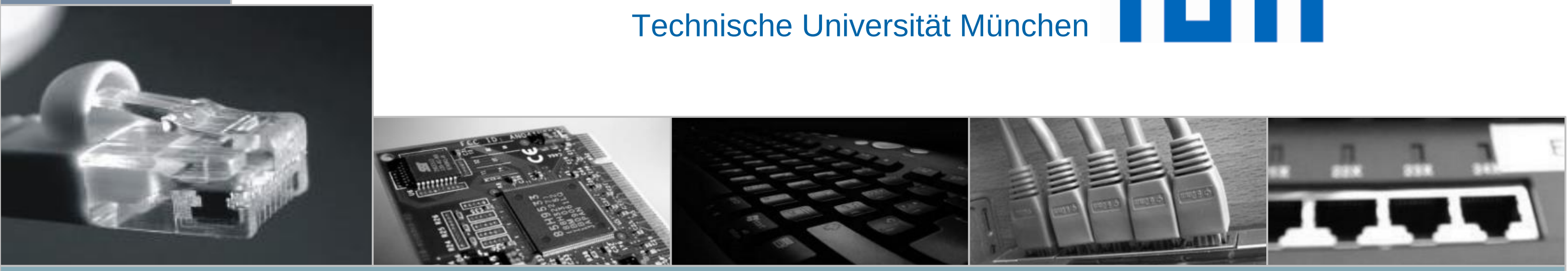
Network Architectures
and Services
NET 2014-05-1

Dissertation

Empirical Analysis of Public Key Infrastructures and Investigation of Improvements

Ralph-Günther Holz

Technische Universität München



TECHNISCHE UNIVERSITÄT MÜNCHEN
Institut für Informatik
Lehrstuhl für Netzarchitekturen und Netzdienste

Empirical Analysis of Public Key Infrastructures and Investigation of Improvements

Ralph-Günther Holz

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Univ.-Prof. Dr. Thomas Neumann
Prüfer der Dissertation:	1.	Univ.-Prof. Dr.-Ing. Georg Carle
	2.	Assoc. Prof. Nick Feamster, Ph.D., Georgia Institute of Technology, Atlanta/USA

Die Dissertation wurde am 18.12.2013 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 05.05.2014 angenommen.

Abstract

Public Key Infrastructures (PKIs) were developed to address the key distribution problem of asymmetric cryptography. Certificates bind an identity to a public key and are signed by a trustworthy entity, called the issuer. Although seemingly a simple concept, the setup of a PKI is not an easy task at all. Trustworthy issuers need to be guaranteed, and certificates must be issued conforming to certain standards. A correct deployment is needed to ensure the PKI is usable for the parties that rely on it. Some PKIs, like the important X.509 PKI for TLS, were criticised from early on for being poor examples with respect to these aspects. The objective of this thesis is to provide a sound analysis of important PKIs and to analyse proposals for improvements for one of them, X.509. The contributions of this thesis are threefold.

In the first part of this thesis, we carry out an analysis of known criticisms of the X.509 PKI and show that they were never addressed well. The approach here is both documental as well as empirical. Furthermore, we provide a survey of incidents in the X.509 PKI, some of which brought it close to failure, and identify their root causes. This analysis allows us to formulate requirements that improvements for the X.509 PKI have to meet. The methodology here is historical-documental.

In the second part of the thesis, we apply empirical methods to analyse the *status quo* for three representative and important PKIs that address different use cases: X.509, the OpenPGP Web of Trust, and the simple key distribution mechanism of SSH. We measure their respective strengths and weaknesses, in particular with respect to deployment, and draw conclusions about the level of security that each PKI achieves in practical use. For X.509, we carried out HTTPS scans of a large number of servers over a period of 1.5 years, including scans from globally distributed vantage points. We also monitored live TLS traffic on a high-speed link of a large network. Our analyses of the thus obtained certification data reveals that the quality of certification lacks in stringency to a degree that is truly worrisome. For OpenPGP, we conducted a graph analysis of the Web of Trust, with a focus on properties such as usefulness and robustness of certification chains. We also analysed the community structure of the Web of Trust and mapped it to social relationships. This allows us to determine for which users, and on which scale, the Web of Trust is particularly useful. For SSH, we carried out several scans over the entire IPv4 address space and collected statistics on host-keys and ciphers used. A number of keys were found to be duplicates, but not due to cryptographic weaknesses. For these, we determined in which network setups they occurred and identified both secure as well as very insecure patterns of use.

In the third part of this thesis, we study five representative schemes to improve the security of X.509. In order to describe each scheme succinctly, we first develop a unified notation to capture the essential protocol flows and properties of each scheme. We then analyse its security properties with particular regard to three threat models that we defined for this purpose. A further particular focus is on the deployment properties of a scheme, i.e., which entities need to make changes to implement it. Based on our findings, we identify the two most promising candidates to reinforce X.509.

However, all schemes fall short in one respect, namely automatic incident reporting and localisation of the position of an attacker. We thus developed and deployed our own solution, Crossbear, to close this gap. Crossbear allows to detect an ongoing man-in-the-middle attack and initiates a distributed but centrally coordinated hunting process to determine the location of the attacker in the network with a fair degree of confidence.

Zusammenfassung

Public Key Infrastructures (PKIs) wurden als Antwort auf das Problem der sicheren Schlüsselverteilung in der symmetrischen Kryptographie eingeführt. Zertifikate von vertrauenswürdigen Ausstellern binden einen öffentlichen Schlüssel an die Identität eines Teilnehmers. Der Aufbau einer PKI ist jedoch kein leichtes Unterfangen. Zum einen muss die Vertrauenswürdigkeit der Aussteller garantiert und Zertifikate müssen nach anerkannten Standards ausgestellt werden. Zum anderen müssen die Zertifikate auf korrekte Weise zum Einsatz kommen, um sicherzustellen, dass alle Teilnehmer sie auch nutzen können. PKIs wie X.509 wurden schon früh dafür kritisiert, dass sie in den genannten Bereichen Schwächen aufweisen. Das Ziel dieser Arbeit ist eine umfassende Analyse wichtiger PKIs und eine Analyse von Verbesserungen für eine davon, X.509.

Im ersten Teil der Arbeit werden frühere, bekannte Kritikpunkte an X.509 daraufhin untersucht, ob seit ihrer Veröffentlichung Verbesserungen erreicht wurden. Es wird gezeigt, dass das nicht der Fall ist. Der Ansatz enthält sowohl dokumentarische als auch empirische Elemente. Weiterhin wird eine Ursachenanalyse für eine Reihe von kritischen Vorfällen in der X.509 PKI durchgeführt, von denen einige die Sicherheit der PKI beinahe außer Kraft gesetzt hätten. Daraus werden Anforderungen abgeleitet, die Erweiterungen von X.509 erfüllen müssen, um Verbesserungen bewirken zu können. Der Ansatz hier ist vor allem dokumentarisch.

Im zweiten Teil kommen empirische Methoden zum Einsatz, um den Status Quo für drei repräsentative und wichtige PKIs zu ermitteln. Diese PKIs decken jeweils unterschiedliche Anwendungsfälle ab. Es handelt sich um X.509, das Open PGP Web of Trust, und die Schlüsselverteilung in SSH. Für jede PKI werden Methoden entwickelt, mit denen die besonderen Stärken und Schwächen der jeweiligen PKI ermittelt werden können. Das erlaubt Rückschlüsse auf die Sicherheit, die die jeweilige PKI bietet. Für X.509 wurde eine große Zahl an Servern über einen Zeitraum von 1,5 Jahren gescannt, zum Teil auch von Beobachtungspunkten rund um den Globus. Zusätzlich wurden Monitoringdaten genutzt. Die Analyse zeigt eine besorgniserregende Qualität der Zertifikate und ihres Einsatzes. Für OpenPGP wird eine Graphanalyse des Web of Trust durchgeführt. Der Fokus liegt auf der Nützlichkeit dieser PKI und ihrer Robustheit gegen ungewollte Änderungen. Zusätzlich wird untersucht, inwiefern sich die Teilnehmer des Web of Trust sogenannten Communities, die sozialen Verbindungen entsprechen, zuordnen lassen. Dies erlaubt es, Rückschlüsse zu ziehen, für welche Teilnehmer das Web of Trust einen besonders hohen Nutzen aufweist. Die Arbeiten zu SSH umfassen Scans, die internetweit durchgeführt wurden. Es werden zum einen Statistiken zu Chiffren und den Schlüsseln erhoben, mit denen sich Hosts in SSH authentisieren. Zum anderen wird eine Analyse von Schlüsseln durchgeführt, die keine kryptographischen Schwächen haben, aber häufig auftreten. Dies passiert vor allem in bestimmten Netzwerkkonfigurationen, von denen einige sehr unsicher sind.

Im dritten Teil werden Systeme zur Verbesserung von X.509 untersucht. Dazu wird eine Notation entwickelt, die es erlaubt, die wesentlichen Protokollflüsse und Design-Elemente eines Systems in einheitlicher Weise zu beschreiben. Darauf baut eine Analyse der Sicherheitseigenschaften des Systems auf. Weiterhin wird eine Betrachtung durchgeführt, welche Schwierigkeiten sich bei der Einführung des Systems ergeben können. Mit Hilfe dieser Betrachtungen werden die zwei aussichtsreichsten Kandidaten zur Verbesserung von X.509 identifiziert. Alle betrachteten Systeme haben jedoch einen Nachteil: Sie erlauben keine automatisierte Meldung von Angriffen oder gar eine Lokalisierung des Angreifers. Daher wurde im Rahmen der Arbeit Crossbear entwickelt: Crossbear erlaubt, Man-in-the-middle-Angriffe auf TLS zu erkennen. Als Reaktion koordiniert Crossbear einen verteilten Prozess, der es ermöglicht, die Position des Angreifers mit hinreichender Genauigkeit zu ermitteln.

Acknowledgements

This work would not have been possible without the invaluable help and steady support of a number of people. I would like to take the opportunity to express my gratitude.

My first thanks go to my advisor Dr. Georg Carle, of course, for making it possible for me to carry out my research at his Chair and providing much support and guidance throughout. He also provided much career advice once this thesis was submitted. I would also like to thank Dr. Nick Feamster for being my second referee and Dr. Thomas Neumann for heading my committee.

I was very fortunate to have some excellent collaborators, research students, and co-authors to work with. A sincere thank you goes to all of them, in particular Lothar Braun, Oliver Gasser, Peter Hauck, Nils Kammenhuber, Thomas Riedmaier, and Alexander Ulrich.

My parents and friends have supported me and believed in me throughout all these years. I cannot begin to say how grateful I am. A very personal thank you goes to my partner Hui Xue for providing moral support and being the wonderful person she is.

Finally, I would like to thank some people who proofread my work or who helped to improve it during many discussions: Lothar Braun, Maren Büttner, Nathan Evans, Christian Grothoff, Michael Herrmann, Nils Kammenhuber, Andreas Müller, Heiko Niedermayer, Stephan A. Posselt, and Stephan Symons.

Contents

Part I.

Introduction and background

1

Chapter 1.

Introduction

It is often said that security is an essential property that should be guaranteed for all electronic communication. With the Internet having established itself as the primary medium of communication, this is now certainly true: the value of communicated information has increased. The need for secure transmission of sensitive financial information (e.g., credit card numbers, bank accounts) is evident. More recently, however, a need to protect messages as a way to ensure personal safety has emerged—the Internet has also become a platform for political activity, for collaboration and information dissemination—with several countries attempting to exercise control over the activities of their dissidents. Networking protocols and applications are thus rightfully expected to protect critical data. Confidentiality, authentication of communication partner and message origin, as well as message integrity, are aspects of security that can be achieved by cryptographic protective measures.

Central to all cryptography is the issue of key distribution, which counts among the hardest problems to solve. Public Key Infrastructures (PKIs) are an important form of key distribution. In a PKI, the goal is to certify a binding between data items. Most commonly, this is the authenticity of a public key, which is expressed as a digital signature on the combination of entity name (identity) and corresponding public key. The issuer of a signature must be a Trusted Third Party (TTP). Analysis of the security of PKIs thus requires to take into account the role and correct functioning of the different entities that are responsible for security-critical operations like identity verification, certificate issuance and safeguarding key material. In this thesis, we analyse the role that PKIs play in protecting Internet communication. Our contributions are threefold.

We begin with a systematic analysis of weaknesses of the PKI that is currently the most important one, namely the X.509 PKI as used for the Transport Layer Security (TLS) protocol [94, 97]. We identify fundamental weaknesses in the organisation and structure of X.509, which revolve mostly around the concept of Certification Authorities (CAs) as TTPs. We provide evidence that critical weaknesses continue to persist in the X.509 PKI to this day, and have not been resolved. This will allow us to draw first conclusions with respect to possible improvements.

The core of this dissertation consists of an empirical analysis of the three PKIs that are most widely used today. Each PKI serves a different use case. Our primary subject of investigation is once again the X.509 PKI, particularly its use in securing the WWW. By active and passive measurement, we determine how well the X.509 PKI has been deployed and the level of security that it provides. The second PKI we chose is the OpenPGP [93] Web of Trust, a PKI where entities are not servers or hosts, but users wishing to secure their private communication. Due to the nature of this PKI, we use graph analysis to determine to which degree it is useful for its users and which security it provides for them. The third PKI is the ‘False PKI’, i.e., a PKI without TTPs, as used in the Secure Shell (SSH) protocol [125]. SSH is an important protocol in network management. Its mechanism of key distribution is different from the other two PKIs, and it is thus a very valuable subject to study. As SSH is closely linked to network management practices, these will be at the focus of our analysis, too.

The third part of this dissertation chooses the X.509 PKI as its sole subject again. Several proposals have been made to reinforce this PKI, with each proposal serving a slightly different purpose. Based on our conclusions in the first part, we provide an analysis of these proposals and assess how robust they are in the face of attackers of varying strengths. As no proposal addresses the open problem of automated attack detection and reporting, the final contribution of this dissertation describes the design, development and deployment of Crossbear, our tool to detect and locate man-in-the-middle attackers.

1.1. Research Objectives

We are now going to outline the Research Objectives of this thesis. There are three overall Research Objectives, which we split into several subtasks.

O1: Identifying weaknesses and historical-documental analysis of incidents

The goal of the first Research Objective is an analysis of the weaknesses of the X.509 PKI by investigating both earlier criticism of X.509 as well as known incidents. This allows us to draw conclusions what requirements mechanisms to improve the PKI have to meet.

In particular, we *analyse earlier academic work and determine which criticisms have been brought forward.* Against this background, we *investigate whether the shortcomings have been satisfactorily addressed and provide evidence for our claims.* Since 2001, several attacks have become known that threatened the security of the X.509 PKI. We use a *historical-documental approach and analyse the reports from incidents.* Our goal is to *identify the root causes and determine how, and by which entity, the attacks were detected.* Finally, we *derive conclusions what kind of improvements would help reinforce the PKI.* Our objective is thus split into three tasks:

O1.1 To identify weaknesses of the X.509 PKI and determine whether these have been satisfactorily addressed.

O1.2 To investigate and analyse incidents, and in particular determine the root causes and how the attacks were detected.

O1.3 To derive conclusions which improvements would help strengthen the X.509 PKI.

O2: Empirical analyses of PKIs

While the previous Research Objective can only provide evidence about a relatively small number of incidents, which furthermore can only be analysed from publicly available sources, the question remains whether systematic weaknesses exist in the PKIs that have been deployed. Having such data would be very valuable as it would allow to mount pressure at entities that operate PKIs to improve the status of their security. Research Objective O2 is thus to carry out *large-scale analyses* to *empirically measure PKI deployment and structure.* As the PKIs to investigate, we choose X.509, the OpenPGP Web of Trust, and SSH.

Different methodologies have to be employed in this endeavour. The X.509 PKI can be analysed by active scans of Web hosts, by analysis of data from TLS monitoring, and by statistical evaluation of the properties of certificates. As X.509 is tree-structured and essentially all information is public, links between cryptographic entities and their properties are relatively easy to trace. The challenge here is in obtaining the data—optimally, over a longer period of time to track changes—and in applying the correct statistical methods on large data sets to derive statistically significant conclusions.

The OpenPGP Web of Trust, however, is not accessible to the above methodology. While data sets can be obtained from so-called key servers, there is very little information stored with such keys. The whole concept of a Web of Trust is based on storing much information locally, and very little publicly. Therefore, the methodology has to be different: only graph analysis can shed light on the links between entities and their properties.

The methodology to analyse the third large infrastructure, SSH, resembles the one for TLS again. However, obtaining data is much more difficult as scans of SSH ports are very often considered hostile in nature by the scanned party. Much care has to be applied to avoid being marked as an annoyance or, worse, a hostile party. Analysis of the data is also slightly different in nature: SSH does not employ certificate chains as X.509 does. Key distribution is practically always managed from a central location. Thus, an important focus must be on identifying issues that arise as a consequence of this particular kind of key distribution and network management.

To summarise, the goal of this Research Objective is to derive empirical data about the three most widely employed PKI technologies, analyse it, and derive conclusions with respect to the level of security each technology provides. We formulate the following three tasks:

O2.1 To analyse and assess the deployment and use of the X.509 PKI for the WWW, with particular focus on security weaknesses and quality of certification.

O2.2 To analyse and assess the OpenPGP Web of Trust, with particular respect to the usefulness and security that the Web of Trust provides for its users.

O2.3 To analyse and assess the deployment of the SSH infrastructure on the Internet, with particular regard to issues that may arise from its method of key distribution.

O3: Improvements to X.509 and attack detection

Research Objective O3 returns to the X.509 PKI for the WWW. We will see from our results in Research Objectives O1 and O2 that there are critical issues to solve, and that the X.509 PKI for the WWW is in a state that makes it infeasible for it to withstand certain attacks, in particular from adversaries on state-level to whom many resources are available.

Several proposals have been made by other researchers to improve (or even replace) the X.509 PKI. These must be analysed with regard to several aspects, in particular the improvements they provide and their robustness against attackers of varying strengths. One also needs to consider whether there are serious obstacles that would hinder their deployment. Optimally, an analysis should be carried out using a common framework to describe the important properties of each proposal—which we will call a *scheme*—in a formalised way. In particular, it should be easy to identify the participants in each scheme, the actions they need to carry out, the interactions with other participants, and the security of the communication channels over which they occur.

Another issue that the research community is facing is a true lack of empirical evidence of ongoing attacks. Such data would be very valuable as it would allow to identify the nature of attacks (local or regional, for example) and possibly help derive the identity of the attackers. As no scheme addresses automated attack detection and reporting, we thus set ourselves the design, development and deployment of such a scheme as a further objective.

Summing up the previous paragraphs, we thus arrive at the following research objectives.

O3.1 To develop a formalised notation to describe schemes to improve the X.509 PKI; to use it to analyse the schemes with respect to the contributions they make to strengthen the PKI, their robustness against attackers, and potential issues with deployment.

O3.2 To design, develop and deploy a scheme that is able to detect attacks against the X.509 PKI for the WWW, and to provide data that allows to analyse the nature of these attacks, in particular the location of the attacker.

Figure 1.1 provides a summary of our Research Objectives for graphical reference, and shows how each part builds on previous ones.

1.2. Structure of this thesis

The structure of this thesis follows the outline of the Research Objectives. We describe it in the following.

Chapter 2 introduces the reader to the **theory and practice of PKIs**. We derive the principal idea from the concepts of authentication and key distribution, both of which we show to be hard problems. This is followed by an overview of the different forms that PKIs may take and present several models and instantiations, particular X.509, OpenPGP, and SSH. Our discussion is structured around the underlying assumptions concerning trust in other entities that are at the core of each model. We also touch on the difficulty of revocation, which is a topic that is often overlooked.

Chapter 3 addresses **Research Objective O1**. We build on previous work to identify critical issues in X.509 and investigate whether these have been addressed (we will see that this was mostly not the case). In the second part of the chapter, we investigate incident reports of the past twelve years in great detail and extract the root causes. Organisational practices, structure of the PKI and technical vulnerabilities will be identified as the primary root causes. Against this background, we derive conclusions which mechanisms may help improve X.509.

Chapter 4 begins the empirical contributions that this thesis makes. It addresses **Research Objective O2.1**. We present the results of our empirical study of the X.509 PKI as it is primarily used, namely in TLS for the WWW. We scanned the hosts on the Alexa Top 1 Million list of popular Web sites and downloaded the X.509 certificates they presented. These scans were carried out over a duration of 1.5 years and also included scans from eight further vantage points on other continents. We complemented our data sets with data we obtained from monitoring TLS connections in the Munich Scientific Network and also included a data set from a third party for reference. We analysed the thus obtained certificates with respect to a number of properties that are crucial for security goals to be achieved. Foremost among these is validity of the certificate chain, especially with respect to information identifying the Web host. We also investigated

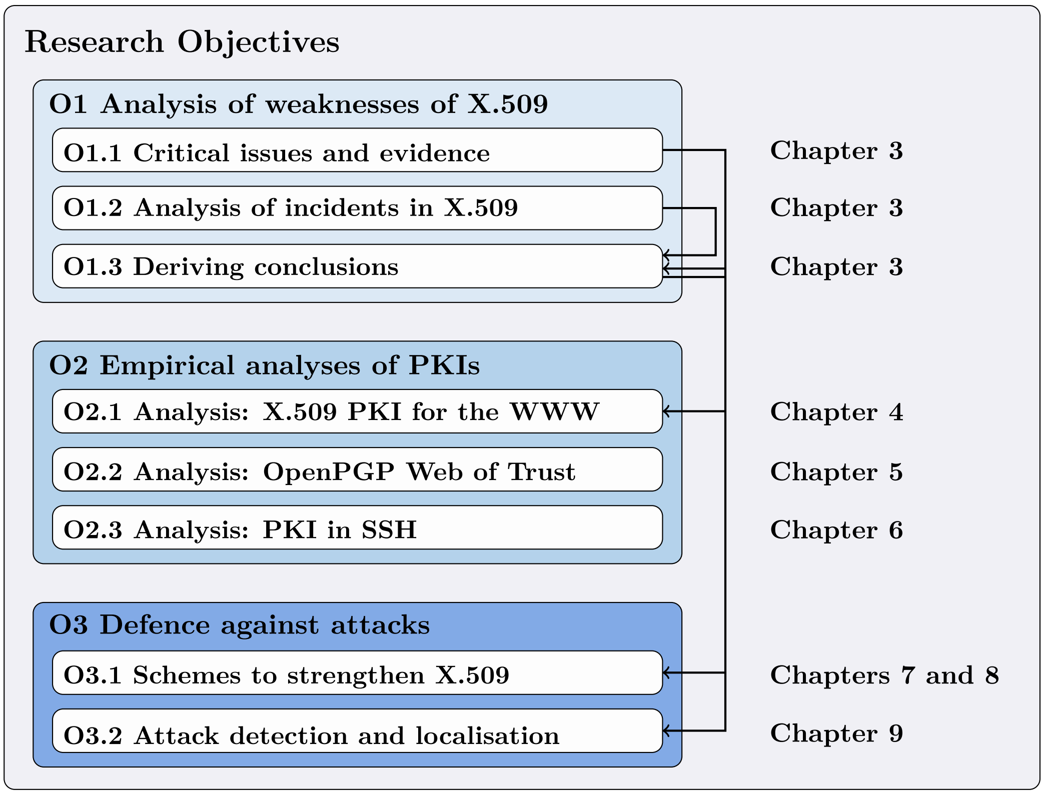

Research Objectives
O1 Analysis of weaknesses of X.509
O1.1 Critical issues and evidence
Chapter 3
O1.2 Analysis of incidents in X.509
Chapter 3
O1.3 Deriving conclusions
Chapter 3
O2 Empirical analyses of PKIs
O2.1 Analysis: X.509 PKI for the WWW
Chapter 4
O2.2 Analysis: OpenPGP Web of Trust
Chapter 5
O2.3 Analysis: PKI in SSH
Chapter 6
O3 Defence against attacks
O3.1 Schemes to strengthen X.509
Chapters 7 and 8
O3.2 Attack detection and localisation
Chapter 9

Figure 1.1. – Research Objectives of this thesis, mapped to chapters. Arrows show which Research Objectives deliver findings on which later chapters are based.

other properties that are relevant for deployment, like validity periods, certificate chain lengths and cryptographic security. Our results show that the X.509 PKI is in a sorry state: only about 18% of certificates offered by Web hosts were completely valid. We also measured the distribution of certificates over multiple hosts, length of certificate chains, use of algorithms, as well as issues like susceptibility to known bugs. Ours was the first academic study on such a large scale and then the only one to track changes in the PKI over time. As our study concluded in mid-2011, Chapter 4 also includes a discussion of academic work that followed our study, confirmed our findings, and added new aspects.

Chapter 5 addresses **Research Objective O2.2**. The OpenPGP Web of Trust is a structure that is entirely different from the X.509 PKI, which is also reflected in its use (e.g., secure email). We present the results of an investigation of the Web of Trust using graph analysis. We first analysed the Web of Trust's major constituents. Although it is almost two orders of magnitude smaller than the Web of Trust as a whole, the so-called Largest Strongly Connected Component is the most meaningful component to analyse as it directly influences the network's usefulness. We evaluated this property by analysing to which degree users are actually able to authenticate other users. The basis of this was an analysis of the length of certification chains, redundant paths, mutual signatures and the Small World effect that the Web of Trust exhibits. We also investigated how robust the network is to changes like key expiration or the targeted removal of keys. As social relations are an important aspect in assessing a key's authenticity, we also applied different community detection algorithms with the goal of mapping identified communities to social relations.

Chapter 6 presents the results of our investigation of the PKI concept in SSH. It addresses **Research Objective O2.3**. This PKI is markedly different from X.509 as its method of key distribution resembles the direct exchange known from a Web of Trust.

However, certificate chains are not supported. For our study, we scanned the entire routable IPv4 space on TCP port 22 and downloaded the SSH host keys that we found. In this analysis, recent work by Heninger *et al.* [35] was our point of departure: the authors had identified critical cryptographical weaknesses and duplicate keys in SSH. We were thus interested whether we could trace such issues to network management practices, i.e., the result of SSH's mode of key distribution. We could indeed identify causes for duplicate keys that are a consequence of network management and provide a list of patterns how such keys may occur. We could also reproduce the results by Heninger *et al.* and show that the situation has slightly improved since their study. Finally, we also studied the use of symmetric cryptography on SSH servers and the occurrence of older servers on the Internet.

Chapter 7 returns to the X.509 PKI and addresses **Research Objective O3.1**. In order to prepare our analysis of schemes to strengthen X.509, we develop a formalised notation to capture the relevant properties of each scheme within a common framework.

Chapter 8 will then make use of the notation and provide the analysis of the proposed schemes. To this end, we define a threat model to describe attackers of varying strength, ranging from a weaker local attacker to a globally active attacker with immense resources at his disposal. The schemes we investigate can be classified by the approach they take. We analyse approaches that are based on pinning, DNSSEC, notaries, and public logs. We assess each scheme with respect to the contributions it makes and its robustness against attackers. A particular focus is on difficulties that may arise when deploying a scheme. Based on our results, we identify the schemes that promise to be the best candidates to reinforce the X.509 PKI.

Chapter 9 addresses **Research Objective O3.2**. Very little is known about man-in-the-middle attacks in the wild, possibly because most users would either not know how to detect an attack, or, if they do, where to report it. Our goal is to develop an infrastructure that allows to collect hard data about such ongoing attacks and report and analyse them in an appropriate manner. To this end, we developed Crossbear, a notary-based tool that can detect man-in-the-middle attacks on TLS and report the attacks automatically. While the underlying notary principle is well understood, Crossbear's novelty lies in its aggressive attempt to also locate the position of the attacker. This is achieved by establishing TLS connections to a victim server and tracerouting from a large number of positions in the network. The interception point between poisoned and clean connections yields the attacker's position with a certain level of confidence. We show that Crossbear can achieve good results in determining a poisoned Autonomous System (AS), although its accuracy decreases significantly at the router level. However, we show that Crossbear functions particularly well against the two attacker types it addresses.

Chapter 10 provides a summary of our findings and reviews them in the context of each Research Objective. We also identify research directions that would help improve PKIs further.

1.3. Note on terminology

Unless otherwise indicated, we use the abbreviation TLS to refer to both the Secure Sockets Layer (SSL) protocol and the TLS protocol. SSL is the predecessor to TLS and uses exactly the same kind of certificates. In the context of our work, there is usually no reason to distinguish between them.

1.4. Publications in the context of this thesis

The following papers were published in the context of this thesis:

- Alexander Ulrich, Ralph Holz, Peter Hauck, and Georg Carle. Investigating the OpenPGP Web of Trust. *Proc. 16th European Symposium on Research in Computer Security (ESORICS)*, Leuven, Belgium, September 2011.
- Ralph Holz, Lothar Braun, Nils Kammenhuber, and Georg Carle. The SSL Landscape—a thorough analysis of the X.509 PKI using active and passive measurements. *Proc. 11th ACM SIGCOMM Internet Measurement Conference (IMC)*, Berlin, Germany, November 2011.
- Ralph Holz, Thomas Riedmaier, Nils Kammenhuber, and Georg Carle. X.509 forensics: detecting and localising the SSL/TLS men-in-the-middle. *Proc. 17th European Symposium on Research in Computer Security (ESORICS)*, Pisa, Italy, September 2012.
- Oliver Gasser, Ralph Holz, Georg Carle. A deeper understanding of SSH: results from Internet-wide scans. *Proc. 14th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Krakow, Poland, May 2014.

2 Chapter 2. Theory and practice of Public Key Infrastructures

This chapter contains text from the sections on related work and background in previous publications. Please see the note at the end of the chapter for further information.

In this chapter, we introduce Public Key Infrastructures (PKIs) and explore the forms a PKI may take. Two terms are at the heart of PKI: authentication and key distribution. PKIs are meant to provide a solution for both, and thus we are going to begin with an introduction to these two fundamental terms. In particular, we will see that although one may understand the term 'authentication' intuitively, it is quite difficult to define formally. Furthermore, we show why the problem of key distribution is a very hard one indeed.

Against this background, we introduce the concept of Trusted Third Parties (TTPs), which is fundamental for PKIs, and then describe PKI concepts that have proved to be relevant, in particular those that we analysed in our own work (X.509, OpenPGP, and SSH). X.509 will be a particular focus as it is a key subject in our investigations and analyses. These sections will give the reader the necessary background for Parts II and III of this thesis.

PKI concepts are often assessed without regard to a problem that is actually essential for their correct operation: key and certificate revocation. We thus dedicate Section 2.7 to this problem and explain revocation mechanisms. This background will prove useful when we analyse compromises of the X.509 infrastructure (Chapter 3).

2.1. The nature of authentication

PKIs play an important role in many cryptographic protocols. Such protocols usually aim to achieve the three goals of confidentiality, authentication and message integrity. Among these, authentication is the essential ingredient—in fact, protocols without authentication are rare. Practically all protocols that computer users deal with today feature at least one mechanism to authenticate the end-points of a communication channel. In these protocols, authentication is a prerequisite to secure key establishment and providing an encrypted and integrity-protected channel.

Intuitively, authentication is a simple concept: the identity of some other entity is ascertained by means of a proof. However, authentication is actually not so easily defined in a mathematically strict sense. The term is frequently used without a clear definition as Boyd comments in [82, p. 33]. Ferguson, Schneier and Kohno, for example, do not define it at all in their book *Cryptography Engineering* [85]. Bishop calls it *'the binding of an identity to a subject'* in his book on computer security [81], and states that this binding may be established by something an entity knows, possesses, etc.

Menezes *et al.* are more precise when they define entity authentication as *'[...] the process whereby one party is assured (through acquisition of corroborative evidence) of the identity of a second party involved in a protocol, and that the second has actually participated [...]'* [87]. This definition states that authentication is a process and that corroboration and evidence are involved. The definition of the evidence remains unclear, however.

In fact, many attempts have been made to give a 'good' definition of authentication, but a great number have also left room for interpretation. As Lowe points out in [51], this has repeatedly led to misinterpretations with respect to the formal guarantees a protocol gives, which is particularly problematic in the formal verification of protocols. It is interesting to note that even the RFCs for TLS and the Internet Protocol Security (IPSec) protocol suite [97, 108] do not indicate what their understanding of the term authentication is.

To facilitate formal verification of protocols, Lowe, and later Cremers, gave formal definitions of authentication and showed that they establish a hierarchy. Lowe, for example, gave a definition of 'authentication as injective agreement' [51]. The idea here is to define authentication between two parties as an agreement on the values in a set of data items, which is the result of a protocol run in which both parties participated. In [16], Cremers shows that 'authentication as synchronisation' is an even stronger form and at the top of the hierarchy.

There is a fundamental link between authentication and the possibility to establish an authenticated and encrypted secure channel between two parties. Consider two entities A and B who wish to establish a secure channel. A necessary condition for them to be able to achieve this goal is that such a secure channel must have been available to them at a previous point in time. In other words, it is not possible to establish a secure channel without already having authenticated credentials. This (intuitively obvious) insight was proved to be correct by Boyd in 1993 [10]. It has an important consequence: the necessary cryptographic material must be distributed between all participants of a particular cryptographic protocol before any authentication can succeed.

2.2. The Key Distribution Problem

For two entities to be able to establish a secure channel, they must be in possession of authenticated credentials. Such credentials are usually shared keys (in symmetric cryptography) or previously exchanged public keys (in asymmetric cryptography). The need to distribute keys to the participants in a protocol is known as the Key Distribution Problem.

2.2.1. Key distribution in symmetric cryptography

In the case of symmetric cryptography, all entities need to have pairwisely exchanged shared keys before they can authenticate. With n being the number of participants, the result would be $O(n^2)$ keys that need to be exchanged, and all of these exchanges would need to be secure. As a consequence of Boyd's theorem, the only way to do this is out-of-band, i.e., by relying on some other, pre-existing secure channel. A typical real-world solution would be delivery in a face-to-face personal meeting. This quickly becomes impractical, especially when a large number of participants is involved (e.g., a bank and its customers). Given that it is also good practice to regularly replace keys, the costs of key distribution in symmetric cryptography rise quickly.

A solution to reduce this overhead is to introduce a Key Distribution Centre (KDC). The KDC is assumed to be trusted by all participants, and it must have securely exchanged symmetric keys with each. The idea is now that the KDC can be queried by an entity A for a symmetric key K_{AB}, to use with another entity B. This implicitly guarantees that A can be sure that only B (and the KDC) will be able to read messages on the secure channel. There are schemes that allow the KDC to create K_{AB} in such a way that A can retrieve it and forward it securely to B. There are a number of details to take care of to make the scheme secure. A well-known example of a scheme that uses a KDC is Kerberos [113].

2.2.2. Key distribution in public-key cryptography

In public-key cryptography, the scalability may seem to be improved as only $O(n)$ public keys need to be published. However, in order to be able to authenticate each other, A and B must have previously exchanged their public keys securely, too, i.e., both must be sure that the respective public key really belongs to the respective other entity. This still requires $O(n^2)$ secure key exchange operations to take place. For small domains with relatively few participants, this may again seem quite reasonable. However, consider the case of email or the WWW: the number of participants is so large that proper key distribution is infeasible. The solution here is to introduce a PKI. The underlying idea is similar to the KDC, but the mode of operation is different.

The key concept is to relieve participants of the need to exchange keys themselves. Rather, they are meant to rely on assertions made by a small set of trustworthy entities—the Trusted Third Parties (TTPs). TTPs make assertions which public key belongs to which entity. An assertion is made by cryptographically signing a tuple (I, k), where I is a meaningful identifier for the entity in question, and k is the entity's corresponding public key. This assertion is commonly called a *certificate*, a term first introduced by Kohnfelder [42].

2.3. Forms of Public Key Infrastructures

Public Key Infrastructures may appear to be a simple concept. However, as we will see in this chapter, creating a *practical* PKI is not an easy undertaking at all.

The essential operation in a PKI is the creation and revocation of certificates. We will call the entity that is responsible for creating a certificate the *issuer*[1]. The issuer's act of applying a digital signature to name and public key creates a cryptographic binding between the two data items that anyone with the issuer's public key can verify. In theory, issuing certificates is a simple (although computationally intensive) task.

The problems begin when theory is turned into practice. Before an issuing entity creates a certificate, it has to verify that the entity who wishes to be certified is indeed the entity it claims to be. Furthermore, it must ascertain that the public key presented to it is indeed owned by this particular entity. Consequently, a central question is which entity, or entities, can be trusted to do this job with due care and act as the TTPs.

Even with TTPs established, however, there is more to take into account. Most PKIs like X.509 or the OpenPGP Web of Trust use so-called certificate chains. A verifying entity—the *verifier*—generally needs to verify a chain from a trusted issuer to the entity whose certificate it wants to verify. In hierarchical PKIs (like X.509), an issuer often wishes to delegate the creation of a certificate for an entity to an intermediate entity. To this end, the issuer creates a special certificate that binds the intermediate's identity, the public key and a boolean value that signals the delegation of certification capacity.

[1]The name for the issuing entity varies from PKI to PKI. In X.509, it is usually called 'Certification Authority'. In a Web of Trust like OpenPGP, the term is often 'endorser'. We use the generic term issuer here.

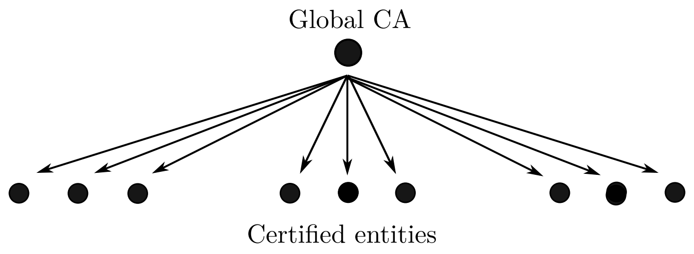


Figure 2.1. – One global CA that directly certifies entities.

The delegated capacity may be delegated again to another intermediate entity, and so on. This process results in a certificate chain that starts at the original issuer and includes one or more intermediate entities. In non-hierarchical PKIs—such as Webs of Trust—certificate chains develop naturally: every certified entity may also be an issuer.

However, it is an interesting question whether (complete) trust into a TTP should also imply trust into the entities to which the signing capacity has been delegated. Assuming full transitivity may often be an unacceptable choice as it does not accurately model real-world trust relationships, and different degrees of trust need to be expressed. This is the realm of trust metrics. A body of literature exists that investigated the different options to compute trust into other entities—the works by Maurer [54] or Jøsang [39] are well-known examples.

A related question concerns the actual structure that a PKI prescribes, i.e., which certificate chains are possible. Most PKIs follow a certain philosophy that governs to which intermediate entities certification rights may be delegated. Perlman gave an overview of the essential constructions in [61].

The simplest form of PKI could be said to be the strictly hierarchical one with one global TTP, which we show in Figure 2.1. The global authority is called Certification Authority (CA). The CA is responsible for verifying the identity of any entity applying for a certificate, for issuing the certificate, and, if necessary, also revoking it. This structure seems rather idealistic on a global scale: there is no universally trusted organisation that could run the necessary infrastructure, and it is hard to imagine that organisations, corporations, and governments, etc. would be willing to rely on the services of a provider that is possibly located outside their legal reach. Even if such an organisation could be agreed on, numerous practical problems would remain. For example, what would be the agreed common standard that the CA would follow in verifying entities coming from very different legislations? What would its chances be to accurately identify entities like owners of Web sites and avoid mistakes? And finally, who would be in actual control of day-to-day activities, and would there be a guarantee that no abuse of power will ever occur?

One way to approach these challenges would be to delegate the task of certification to regionally operating subordinate CAs. This would relieve the global CA from some of its responsibilities. Regionally active entities might also operate more efficiently, with better capacity to carry out tasks like identity verification, due to their knowledge of local practices. As every subordinate CA is a CA in its own right, this would increase the number of attack vectors, however. An alternative would thus be to use so-called Registration Authorities (RAs) instead. These entities cannot sign certificates themselves but are responsible for properly identifying an entity according to local legislation and then for creating a certification request and forwarding it to the global CA. We show this model in Figure 2.2. This restricted form of delegation only removes some of the technical attack vectors, however. The social ones remain: unless the correct operation of RAs can be guaranteed in a meaningful way, attackers may attempt to

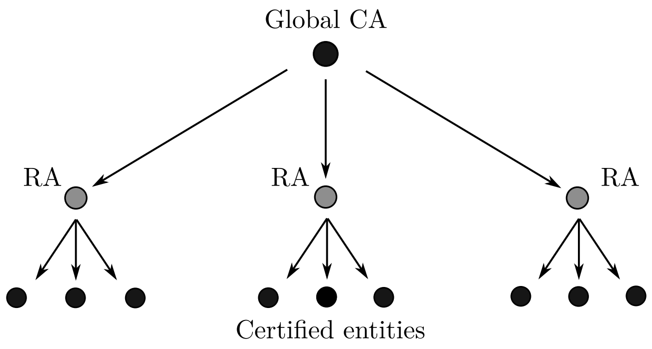


Figure 2.2. – One global CA with several RAs.

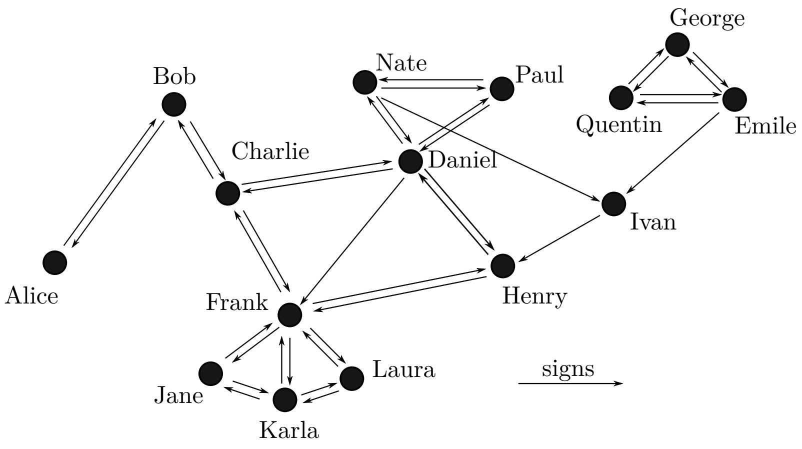


Figure 2.3. – A Web of Trust is an example of a non-hierarchical PKI. Users sign each others' keys and verify the authenticity of other keys using trust metrics.

obtain certificates via RAs that do not execute their duties carefully enough. This shows that the operational factors in a PKI are as important as its technical basis.

Historically, different PKI structures have come into use. The structure used for X.509 on the WWW, for example, is a variant of what we just described above. Perlman calls this structure *'configured CAs plus delegated CAs'* [61]. The concept developed from early browsers beginning to integrate the public certificates of CAs and later introducing programmes to which CAs can be admitted. We elaborate on this in the next section. It is interesting to note that, while Perlman's article is from 1999, it already warns of possible attack vectors in this model. We will show in Chapter 3 that this assessment was quite correct. Yet X.509 is one, if not the, dominating standard on the Internet today.

A notably different model is a Web of Trust as implemented by, e.g., OpenPGP, where participating entities are free to certify any other entity. We describe this in more detail in Section 2.5. The resulting structure is shown in Figure 2.3. OpenPGP enjoys a certain popularity for encrypted and signed email. Thus, we chose it as a further PKI to analyse.

We have so far not elaborated on the simplest possible PKI, which we might call the 'False PKI'. Here, public keys are simply distributed among participants by external means, and there are no TTPs nor CAs. While one might argue that this model does not constitute a 'real' PKI at all, it is actually used very often: it is the structure preferred by the popular SSH protocol, which runs on millions of hosts. The security of the structure depends exclusively on the correctness of the key deployment operations. It is clearly a very interesting PKI, and we thus chose it as the third PKI to analyse.

A variety of other schemes to structure PKIs exist. Some PKIs attempt to introduce very strong technical constraints that rule which entities a CA is allowed certify. The Domain Name System Security Extensions (DNSSEC), for example, construct a PKI with a single authoritative key for the DNS root zone. Certification is then delegated to subzones and their subzones, creating a tree of 'CAs'. As the DNS is itself a tree of names, each zone can be technically constrained to only issue 'certificates' (really signatures on resource records) to domain names that are in its own zone, with delegations to subzones. DNSSEC did not play a significant part in our empirical analyses as it is not widely deployed yet, but we do return to it when we discuss PKI alternatives in Chapter 8.

2.4. X.509 Public Key Infrastructure

X.509 is a tree-structured hierarchical PKI. It is an ITU-T standard that has been adopted by the Internet Engineering Task Force (IETF) as the PKI for several IETF protocols. X.509 certificates are an integral part of the TLS protocol, which secures application layer protocols like HTTP, IMAP and POP3. The most common use case is authentication of a server, but certificates can also be used to authenticate clients.

2.4.1. Historical development of X.509

The history of X.509 is well summarised in, e.g., the work by Gutmann [32] and in RFC 2693[2] [100]. We give a brief summary here.

The roots of X.509 are found in the hierarchical structure of X.500, an ISO/IEC standard of 1988 intended to be used by telecommunication companies. The intention of the standard was to create a global directory of named entities, with data structured in a hierarchical tree. Different organisations would be responsible for different subtrees. The use of certificates was incorporated to allow users to authenticate to this directory and retrieve data. A CA would be an entity responsible for a subtree, i.e., for issuing certificates to users to allow them to access this subtree. One global CA was to be at the top of the hierarchy. X.509 was the standard that defined the certificate format to be used in the X.500 series of standards. As it turned out, however, X.500 never saw much deployment[3]. Furthermore, X.500 required a strict naming discipline. However, as the authors of RFC 2693 [100] note, at the time X.500 was specified, too many different naming disciplines were already in use.

When the Internet, thanks to the WWW, began to grow into its role as a major communications infrastructure, the X.509 format was mandated in the new Secure Sockets Layer (SSL) and later Transport Layer Security (TLS) protocols. As these connected hosts on the Internet, which were referred to by their DNS names, some way was needed to securely associate a DNS name with a public key. The role of CAs was filled by companies that offered X.509 certificates for owners of DNS domains. CAs were (and are) expected to carry out checks of domain ownership with due diligence. However, these CA-internal work processes are difficult to assess from the outside, and thus users need to place their trust into the correct working of a CA. In particular, the verification that a requesting party really owns a domain has always been problematic

[2]RFC 2693 describes the competing SDSI/SPKI standard.

[3]The original X.500 was meant to be used with the ISO/OSI stack of network protocols, which never gained wide support. Adaptations of X.500 protocols to TCP/IP exist, e.g., LDAP [129].

as most CAs rely on insecure protocols to carry out this verification. We return to this in the next chapter.

More importantly, the notion of responsibility for subtrees was removed in this new X.509 PKI: CAs have no association with DNS zones here. Rather, any CA can issue certificates for domains anywhere in the DNS hierarchy. These activities quickly developed into a successful business model that continues to this day. Since sites could buy their certificates from any CA, operating systems and browser vendors began to ship the certificates of a number of CAs with their products to allow client software and browsers to verify certificates.

The inclusion process for new CAs generally depends on the respective vendor. In the case of Mozilla, it is an open forum where inclusion requests of new CAs are publicly discussed. Companies like Microsoft or Apple do not make their decision processes very transparent, although there is some effort to publicly document the outcome, in form of a list of included CAs (e.g., [221]). For quite some time, both the inclusion policies that browsers used as well as the certification practices that CAs followed were largely independently defined. In 2005, the CA/Browser Forum [150] emerged as a collaborative body of browser vendors and CAs, with the goal of releasing normative guideline documents. This forum needed several more years to agree on the first such document, the Baseline Requirements (BR) [148], which took effect on 1 January 2012. This document defines the minimal set of common policies that all CAs with membership in the CA/Browser Forum have agreed to implement.

The number of CA certificates in browsers has grown for a long time, and it continues to grow. Furthermore, CAs have recently introduced new products, among which Extended Validation (EV) certificates are maybe the most significant ones: these certificates are intended for organisations and are supposed to be issued only after a more thorough evaluation of credentials, e.g., by verifying legal documents. They are generally much more costly than normal domain-verified certificates [7].

Although server certificates have remained a pillar of CA business models, CAs can also issue certificates for persons. A popular example are personal certificates for email as defined in the S/MIME standards [118, 115, 116].

2.4.2. Certificate structure

Figure 2.4 shows the simplified structure of an X.509v3 certificate. Version 3 is by far the most common version that can be found today. The most important fields are the issuer, the subject, the public key (algorithm identifier and actual key) and the digital signature (algorithm identifier and actual signature). A serial number is stored as the reference number under which an issuer has created this certificate. The validity is specified in two fields that act as start and end date. In addition to these fields, X.509v3 introduced a certain number of extensions. Among these, we find a flag that indicates whether this certificate belongs to an entity with CA-like properties, i.e., one that will use its private key to sign other certificates. There are also fields for revocation purposes, e.g., Certificate Revocation Lists (CRLs). The EV field signals the EV status of a certificate. This field is expressed as an Object Identifier (OID): just as they maintain a list of CA certificates, browser vendors also maintain a list of OIDs that they associate with EV status. There are several more fields that we do not mention here—e.g., fields indicating the allowed uses of the certificate.

2.4.3. CA hierarchy

Since there is a relatively large number of CAs in the X.509 PKI, it can be viewed as a forest rather than a tree. CAs and subordinate CAs play different roles in the PKI.

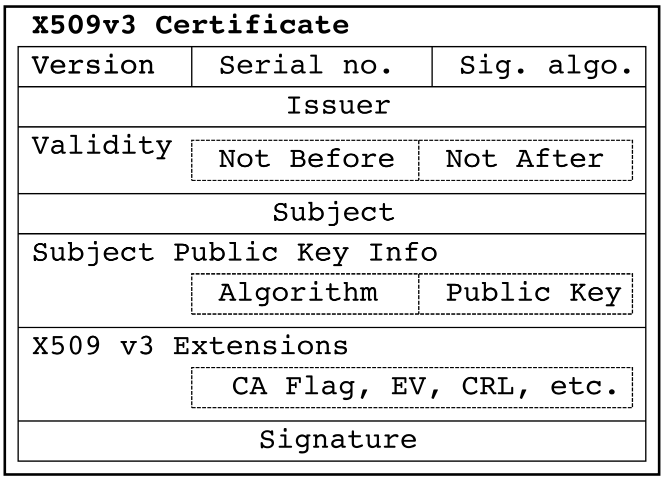


Figure 2.4. – Schematic view of an X.509v3 certificate.

Figure 2.5 shows a minimised example. CAs reside at the top of the hierarchy and are all equally allowed to issue certificates to any domain in the DNS hierarchy. They are identified by their own certificates, which they issue to themselves and sign with their own private keys. These certificates are called root certificates.

Root certificates, indicated as R_x in Figure 2.5, could be used to directly sign end-entity certificates (indicated by E_x), and indeed some CAs do this or have done so in the past. However, this means that the private key must be held online and in memory during a CA's operations. This constitutes an attack vector. Thus, it is considered good practice for CAs to issue intermediate certificates from the root certificates. These certificates have the CA Flag set to true and are used in online operations[4]. We indicate them by I_x. The private key for the root certificate can be stored safely offline in a protected location. The public key certificate can be shipped with browsers and operating systems. If anything should happen to the intermediate certificate's private key, it can be replaced without having to replace the shipped root certificate. In our example, R_1 and R_2 are in the root store (and thus coloured green), while R_3 is from a CA that has so far not been included (coloured grey). Hence a browser with this root store would reject E_7 as an untrusted end-host certificate (and display a warning message to the user).

An intermediate certificate may be used to sign further intermediate certificates. These may or may not be controlled by entities that are part of the same organisation as the root CA. Intermediate certificates always increase the number of private keys that must be protected. If they are issued for subordinate CAs, i.e., organisations outside the direct control of the root CA, this is a particularly sore point as an intermediate certificate has the same signing power as any other certificate. Note that this method also removes some control from the client: it is debatable if users would trust all intermediate authorities, assuming they even knew about them—the existence of subordinate CAs can only be inferred from close inspection of a certificate chain and investigation of the business relationship of the entities specified in the certificate subjects.

An interesting use case for intermediate certificates is cross-signing, where a CA also has a certificate (I_4) that has actually been signed by another CA (in this case, via an intermediate certificate). This is useful if the other CA is contained in a root store in which the cross-signed CA is not contained. The thus certified CA essentially becomes a subordinate CA of the issuing CA. If a CA or any of the subordinate or intermediate

[4]This method is now required by the Baseline Requirements [148].

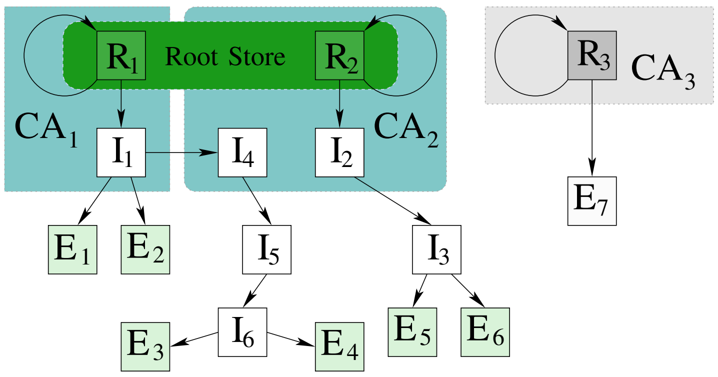


Figure 2.5. – X.509 certificate chains and root stores. Figure adapted from [37].

CAs it cooperates with becomes compromised, the attacker can issue certificates for any domain. Thus, both cross-signing and subordinate CAs can be very dangerous to the PKI system.

2.4.4. Use of X.509 in TLS

The SSL protocol was originally developed by Netscape and later standardised within the IETF as TLS. Since 1999, three versions of TLS have been specified: TLS 1.0 in 1999 [95], TLS 1.1 in 2006 [96], and TLS 1.2 in 2008 [97]. The protocol specifications have been updated over the years to incorporate up-to-date cryptographic algorithms, remove weaknesses, and add extensions. However, the core ideas of the protocol have remained the same.

TLS achieves authentication and session key establishment between a client and a server. X.509 certificates are used to authenticate the communicating parties. In the following, we describe the essential steps of TLS, abstracting from the actual RFC. TLS requires two round-trips between client and server for authentication and key establishment to complete. Although the protocols are subdivided in a number of subprotocols, the exchange between client and server can be described as the exchange of four messages. The principal idea is that information relevant for authentication is exchanged in the first two messages (or three for mutual authentication), and the correctness of the initial exchange confirmed afterwards by exchanging message authentication codes over all previous messages. TLS also supports key establishment via a Diffie-Hellman key exchange, which achieves the important property of Perfect Forward Secrecy (PFS). Like most cryptographic protocols, TLS distinguishes between long-term (public) keys and (symmetric) short-term keys (the session keys). The long-term keys are used to establish session keys at the beginning of a communication. PFS means that an attacker can never obtain the session keys for a communication that he has only recorded and for which he obtains the long-term keys only after the communication has already completed. The Diffie-Hellman key exchange is described in [17]. We include the mechanism here in its form with ephemeral values, i.e., non-fixed values.

Message 1 TLS begins with a client C initiating communication with a server S. C sends a nonce n_C to S, together with a list of cryptographic suites it can support (L_C)[5]. The list contains combinations of symmetric ciphers, cryptographic hash functions and key exchange mechanisms. We denote this first message as:

$$C \longrightarrow S : n_C, L_C$$

[5]The RFC also uses session identifiers for tasks like session resumption. We omit this here.

Message 2 Upon receiving C's message, S will respond to it with its own nonce (n_S). It will select an entry from L_C, which we denote as l_S. It will add its own certificate and the intermediate certificates ($Cert_S$). Adding the root certificate of the CA is optional, as it is assumed that this certificate must already be in a client's root store anyway for authentication to succeed. If a Diffie-Hellman key exchange was requested, the server also includes the necessary Diffie-Hellman values (DH_S). A signature must be added to demonstrate knowledge of the private key (corresponding to the certificate) and signal the correctness of the nonces and the Diffie-Hellman values. The server may also include a request for a client certificate.

$$S \longrightarrow C : n_S, l_S, Cert_S, DH_S, Sig_S(n_C, n_S, DH_S)\ \{, req\_cert\}$$

Message 3 Upon receiving the server's message, C will determine whether the certificate is valid. This includes a number of steps and verification of fields in the certificate, described in RFC 5280 [94]. The following steps are the essential ones. The verifier begins by tracing the certificate towards the root:

- The client verifies that all signatures in the certificate chain are correct.
- It verifies that each issuer of one certificate is the subject of the certificate further up the chain. All intermediate certificates must have the CA flag set to true.
- It verifies that the root certificate is included in its root store.
- It verifies that all certificates in the chain are within their validity period.
- It verifies that the subject of the end-host certificate corresponds to the entity it wishes to connect to.

Verification is often more complicated with many possible paths through the process. For example, certificates may also carry a flag that indicates whether they are intended for securing HTTP or not—the client can inspect this field if it is present[6]. More importantly, the verification of the subject depends to some degree on the application protocol on top of the TLS connection, or even the user: the application or user must verify that the entity named in the certificate subject is the intended communication partner. For HTTPS, this means that the field Subject Alternative Name (SAN) in the certificate must match the DNS name of the host the client connected to (wild cards are allowed). However, mostly due to historical use, this information is often stored and accepted in the so-called Common Name (CN) in the subject, although this is technically wrong. Some certificates (and all EV certificates) indicate the real-world organisation behind the DNS name, which can be displayed by a browser and verified by a user. In practice, users rarely invest this kind of effort.

It should also be noted that there are X.509 extensions whose use requires more verification steps. For example, a certificate of a subordinate CA may be name-restricted, an extension that has only recently come into some use. It means that the certificate carries a restriction that says the corresponding private key may only be used to certify domain names under a certain domain in the DNS hierarchy (e.g., a top-level or second-level domain). The path length extension is a related extension: it defines how many intermediate certificates may occur in a chain.

[6]This field is officially called Extended Key Usage and not mandated by RFC 5280 [94]. It is different from the field Key Usage, which refers to the basic operations allowed for this certificate, e.g., encryption or key agreement.

If the certificate chain could be verified to the client's satisfaction, the client will continue with the handshake. It first sends its own certificate (if it was requested). Then, it proceeds to compute its own Diffie-Hellman value, which it also uses it to derive the symmetric key $K_{C,S}$[7]. The client sends its own Diffie-Hellman values (DH_C). If it was asked to authenticate (i.e., the server requested its certificate), it shows possession of its private key as well as the correctness of the Diffie-Hellman values by adding a signature over all message fields that have been sent so far. Finally, it adds a keyed hash value (i.e., a MAC) over all message fields so far.

$$C \longrightarrow S : \{Cert_C, \}\, DH_C \,\{, Sig(X\_fields)\}, MAC(X\_fields\,\{, Sig(X\_fields)\})$$

Here, the term *X_fields* indicates the concatenated fields n_C, L_C, n_S, l_S, $Cert_S$, DH_S, $Sig_S(n_C, n_S, DH_S)$, potentially *req_cert*, potentially $Cert_C$, and DH_C. The MAC is computed with $K_{C,S}$ as the key.

Message 4 The server can verify the client's certificate in the same way as described above, although the verification of the subject field may take a number of forms, depending on the server's configuration. Using the value DH_C, the server can now compute $K_{C,S}$ and verify the validity of the MAC. This ensures all previous messages were correct. The server confirms that the whole authentication process was correct by sending a final MAC over all previously sent fields, including the signature and MAC sent in the client's last message:

$$S \longrightarrow C : MAC(X\_fields\,\{, Sig(X\_fields)\}, MAC(X\_fields\,\{, Sig(X\_fields)\}))$$

2.5. The OpenPGP Web of Trust

Webs of Trust are a different form of PKI. In this section, we introduce the Web of Trust of OpenPGP, which is a user-centric and self-organised form of PKI. A user in OpenPGP is identified by a user ID, which contains a user-chosen name and email address. Every user ID is associated with a public/private key pair, which is stored locally in a data structure that also contains an expiry date. Users 'issue certificates' by signing another key (i.e., user ID and public key) with their private keys.

In contrast to X.509, certificates are not verified with reference to a root certificate in a root store but by finding a certification path from the own key to the key that is to be verified as belonging to some entity. This allows practically arbitrary paths (although cycles are not considered). Before using someone else's public key, users must determine the key-entity binding and assess whether it is likely to be correct. Consequently, the question arises which intermediate keys (and users) one should trust, and to which degree, in verifying the authenticity of another key.

OpenPGP uses a trust metric to allow users to assess the trust in a key-entity binding. Users store keys of other users in so-called key rings. For these keys, OpenPGP defines two notions of 'trust'. First, 'introducer trust' refers to how much another user is trusted to apply care when verifying an identity. This value is expressed as a string with an associated numerical value, with a higher value signalling more trust. It must be set manually by a user for all keys of other users in his or her key ring. The value is stored locally together with the key (not the user ID). 'Public-key trust' is the degree

[7] In practice, one would derive different keys for encryption and message authentication codes, but we simplify this here.

to which a user claims to be sure of a key-entity binding himself. This value is again expressed as a string with an associated numerical value. It may be stored as part of the signature.

OpenPGP's trust metric is parameterised, and the parameters can be set by the users themselves. The popular implementation GnuPG, for example, uses a default setting that focuses on introducer trustworthiness: this must either be 'full' for all keys on the certification path, or there must at least be three redundant certification paths to the key in question (with all keys on the paths assigned a trust value that is less than 'full'). No certification path must be longer than five keys, either. Trust in OpenPGP relies on social relations for identity verification. CAs are not forbidden in OpenPGP, however—they are merely a very special kind of user. OpenPGP's model can thus be viewed to be more focused on the local 'environment' of a user—it is infeasible for a user to determine introducer trust for everyone in the Web of Trust. A user can only make reasonable assessments about keys to which paths are short and lead over social contacts. When talking about the scalability of the Web of Trust, this is important to keep in mind.

The exact mechanism of creation of OpenPGP's Web of Trust is not exactly known, but it is commonly agreed that personally established contact between users plays a major role, in particular as there are organised events, the so-called Keysigning Parties, at conferences and meetings. To make keys available to the public, users can upload them to a network of key servers, together with the signatures that they have received for their own key or issued for other keys. The key servers use the Synchronizing Keyservers (SKS) protocol for synchronisation. This protocol implements a gossiping strategy to ensure changes to a key server are propagated quickly. There does not seem to be a formal documentation, although a home page exists for the reference implementation [226]. A snapshot of the data stored on a key server contains a complete history of the Web of Trust: keys cannot be deleted from an SKS server and timestamps of key creation, signature creation, and expiry dates are stored.

The open nature of the Web of Trust could lead one to speculate whether large-scale attacks on the Web of Trust are possible, where a malicious entity certifies a large number of keys to trick others. However, this attack is much more difficult than it seems. Assume Alice wants to verify a fake key for the identity Bob, which has only been signed by a number of false identities signed by Mallory. Alice must establish a certification path to the 'fake Bob key' using the faked signed keys. These faked keys would only be used in a path search if Alice manually and explicitly sets a trust value for Mallory *and* the false identities. As setting introducer trust is a manual operation, it is unlikely that Alice would assign the needed trust to unknown entities.

Note that the mapping from human users to user IDs and keys is generally not bijective in OpenPGP. It is not uncommon for users to have multiple keys, for example under pseudonyms or for different email addresses. This can make it difficult to assess who the real person behind a user ID or key is.

2.6. The Secure Shell (SSH) and its PKI model

The SSH protocol started as a replacement for insecure login protocols. It dates back to 1995 [124] and has been a popular tool since its conception. Its versatility and omnipresence make it indispensable for many power users, especially system administrators. SSH is a prime example of using a 'False PKI' without TTPs.

The SSH protocol resides on the application layer and provides authentication (of server and client), encryption, and message integrity. It is primarily used for remote shell access to hosts, including infrastructure devices such as routers. SSH is also notable because it provides support for several other protocols that it runs as so-called subsystems over an encrypted connection. Two particularly important examples are secure remote copy (SCP[8]) and secure FTP (SFTP[9]). Many implementations of SSH also allow to create secure tunnels for arbitrary application layer protocols. In the following, we will give some background on the PKI model that SSH uses as well as discuss some aspects of the protocol flow.

2.6.1. PKI model

Although the use of X.509 certificates is defined for SSH [106], the most popular method to authenticate a server is based on the so-called host keys. A host key is a public/private key-pair of which the public part is sent during connection establishment. The client needs to have been introduced to this key securely *a priori* and out-of-band. This is most commonly achieved by administrative control over both client and server, i.e., system administrators store the server's public key on the clients. There is a second accepted method: when a client cannot authenticate a server by its host key, it will display the host key's fingerprint (a hash value) and leave the decision whether to continue to the user. If the user decides to continue, the fingerprint will be stored locally for future reference and comparison, just like in the predistributed case. In later connections, SSH also displays a warning if the server suddenly presents a different key. This concept is variously known as *trust on first use*, *leap of faith authentication*, *pinning*, or *key continuity.*

It is an interesting and mostly unanswered question to which degree the approach taken by SSH works, and what level of security is achieved. First, the key distribution does not scale well, invites human error, and may lead to misconfigurations. Second, it is not clear if common users of a computer have the necessary knowledge to understand their client software warn about an unknown key (or if they do, whether a relaxed attitude may cause them to continue anyway). Ultimately, such questions are intrinsically linked to network management and the methods used for configuration.

2.6.2. Protocol flow of SSH

The SSH protocol consists of several subprotocols [111, 125, 126, 127, 128]. However, the protocol flow of SSH is not so different from TLS and can be described without referring to each subprotocol. Both TLS and SSH first negotiate the methods for key exchange in a first round of message exchanges, in which the server is also authenticated. Authentication of the client happens in the second round of messages, which is also used to cryptographically confirm the correctness of all previous messages. However, authentication in SSH has some aspects that are not shared by TLS.

First, the protocol is unusual in that it does not define a strict sequence of messages between client and server. The client initiates the connection with a TCP handshake, and client and server send identification strings and version information—but the order of these messages is not defined. In particular, the server may send before the client. This first message exchange is followed by messages to negotiate parameters and cipher suites for key exchange. Again, the order is not defined, but a resolution algorithm ensures that an agreement can be reached. The parties may choose different suites. The server authenticates first with its host-key and proves possession of the private key in a challenge-response exchange.

In the second round of messages, the client (or rather, the user) must authenticate to the server. There are several supported options, like passwords or wrappers for

[8]SCP was never published as an RFC.
[9]SFTP is only specified in an expired Internet-Draft [102].

system-wide mechanisms like Kerberos. Most notably, a user's public key may also have been placed in his home directory by an administrator, which allows login via another (automated) challenge-response process.

It is this latter method that contributes to the second interesting aspect of SSH. It has a profound impact on the feasibility of man-in-the-middle attacks. Thanks to the Diffie-Hellman key exchange, an attacker would have to actively exchange Diffie-Hellman parameters. However, SSH is designed to also use session IDs, which must match between client and server. The message exchange is designed in such a way that an attacker can either swap the Diffie-Hellman secrets, or be able to forward the correct session ID to its victims but not both. Tampering with a connection is thus detectable if clients use public-key authentication.

2.7. Revocation

Public keys and certificates can be marked as unsuitable for further use. This process is known as *revoking a key* or *revoking a certificate*, respectively. There are important reasons to provide this possibility. Key owners might want to replace an older key with a newer, stronger one, and wish others to stop using the old key. For client certificates, the name by which the owner is referred to may have changed, or the owner may not belong to the same organisation any more (if this relationship was indicated in the certificate). There are also more serious reasons: the private key that belonged to a certificate might have been mislaid, lost, or even compromised. This section provides a summary of revocation in PKIs.

2.7.1. Revocation and scalability

Revocation in PKIs is remarkable in that it has a profound impact on the scalability of the entire PKI. This is especially true for hierarchical PKIs, where revocation almost inverses the argument of scalability.

For revocation to be effective, a verifier must be able to retrieve status information about the key or certificate before it uses it. The only entity from which reliable information about a key or certificate's status can be retrieved is the issuer. In the case of hierarchical PKIs, there are comparatively few issuers, namely the root CAs and (possibly) subordinate CAs. The number of verifiers is often several orders of magnitude higher. As every verifier must communicate with the issuer, this means effectively that issuers have to deal with a very high load of requests—the beneficial effect of PKI, scalable key distribution, is countered by poor scalability in revocation checks.

Consequently, a revocation mechanism should attempt to minimise the number of necessary queries. In the following, we are going to present the mechanisms for the forms of PKI we have presented so far, with a focus on the hierarchical X.509 PKI and the Web of Trust that OpenPGP employs. In X.509 at least, there is also a noteworthy difference between theory and practice, which we are going to explore.

Interestingly, revocation is practically no concern in the 'False PKI' of SSH: there are no TTPs that could be bottlenecks. As administrators are assumed to have control over both client and server, revocation can be achieved by simply replacing a host key with a new one.

2.7.2. Revocation in X.509 for TLS

Two mechanisms are defined for revocation checks in X.509. The first one, Certificate Revocation Lists (CRLs), is defined in the same RFC as X.509 [94]. It is an offline mechanism for revocation checks. The Online Certificate Status Protocol (OCSP) is a more recent addition and provides an online way to obtain certificate status information.

Certificate Revocation Lists

The idea behind CRLs is that CAs regularly issue lists of certificates with status information about revoked certificates. The mechanism is defined in RFC 5280 [94]. A CA may issue several CRLs, each with a different scope (e.g., dividing by revocation reason). Certificates in CRLs are identified by the issuer's identity and a serial number. A full CRL is a list of all certificates that have ever been revoked. CAs may choose to issue so-called delta CRLs: these are incremental updates against a base CRL. A CRL must itself be signed by the CA so a verifier can determine its correctness. It also contains information when the next update is going to be published. The CA may choose between different mechanisms how a CRL can be downloaded or obtained, e.g., via HTTP, FTP, etc.

Revoked certificates are listed with additional information. Two items are of particular relevance, although both are optional: invalidity date and reason. The former represents a date from when on a certificate must be considered revoked. This may be a point in time that lies before the actual issuing date of the CRL itself: consider a protocol that involves the use of the private key that corresponds to some certificate, and applies a signature to some timestamped data item as part of the protocol. The invalidity date allows the verifier to determine if the signature was issued while the key was already revoked or not. The reason for revocation can also be given in a CRL; the standard specifies a list of reasons like key compromise or compromise of the CA, but also more mundane reasons like changed affiliations.

One issue with CRLs is generally that they can grow rather large—some CAs may have many thousands of customers for which they issue several certificates over time. With a potentially large number of CAs in a root store, this would mean that clients would waste considerable bandwidth to regularly download CRLs that contain a list of all revoked certificates instead of the ones that the clients really need information about. Furthermore, CRLs can generally not be downloaded when they are needed, i.e., during TLS connection establishment: this would induce a rather high delay in connecting to a Web site. Thus, the user must accept that revocation information is only updated at certain intervals and that there is a certain time window for an attacker until the next update is published.

Online Certificate Status Protocol

OCSP is a protocol to retrieve status information about a certificate in an online interaction. OCSP allows to obtain fresh revocation information. It is defined in RFC 6960 [119]. The protocol describes a request-response mechanism between a client and an OCSP server. An OCSP request message by a client may contain queries for several certificates at once. A queried certificate is identified by a 3-tuple, namely a hash over the issuer name, a hash over the issuer's public key, and the certificate's serial number. The idea here is that a serial number is always logged and stored at the issuer, and that it is unique for the given issuer. A nonce against replay protection is an optional part of the request, as is a client signature.

An OCSP server (often called *responder*) is responsible for a particular issuer. In its response to a client, it identifies itself (by name or hash value of its public key), states the time at which the response was generated, and adds a response for each of the queried certificates. The response is signed by the server. It may also include extensions, like a reference to the CRL on which the response is based.

There are only three values to express the status of a certificate. It may either be *good* (i.e., the certificate is valid to use), *revoked* (the certificate must not be used) or *unknown*. The *revoked* status can be expanded with a reason for the revocation and must contain a timestamp. The allowed reasons are exactly the same as in the RFC for CRLs (see above). The revocation can be temporarily or permanent; this can be expressed in the reason.

The status *unknown* is interesting because it is the answer the responder is supposed to send if the certificate in question is unknown to it, for example because it is not from the issuer for which the responder is responsible. This leaves it open to the client to try another resource like a CRL. The RFC does not require clients to abort the connection attempt—as we will see in Chapter 3, this can be a dangerous weakness.

OCSP Stapling

OCSP, in its basic form, suffers from two issues. One is that an OCSP responder can determine a client's browsing habits, which can be a serious violation of privacy. If a client sends an OCSP request every time it visits a new domain, the responder can track it by its IP address. The second issue is performance. The OCSP default mode of operation puts all load onto the OCSP responder, and the load grows with the number of verifying clients, i.e., browsers. In a blog entry [253], Symantec claimed to serve 3.5 billion OCSP requests per day[10].

An approach to solve this is the TLS Certificate Status Request extension, also often called OCSP Stapling. It is an extension for TLS that can be part of the handshake [99]. With OCSP Stapling, it is the server's responsibility to obtain a proof that its certificate is still considered valid by an authorised OCSP responder. A server is supposed to obtain a valid OCSP response for its certificate and return it as part of the TLS handshake, immediately after its own certificate. The RFC allows the server to reuse the OCSP response to avoid having to query for every single client connection.

OCSP Stapling provides several important benefits. It solves the privacy problem and takes considerable load from the CAs. Web servers can limit themselves to requesting a new OCSP response at certain intervals. Finally, OCSP Stapling also makes the TLS handshake faster as the client does not have to connect to a further party to verify the certificate, saving TCP round-trips and DNS lookups.

However, even OCSP Stapling has some shortcomings. A very important one is that the original RFC does not allow to transport OCSP responses for the intermediate certificates in a certificate chain, only for the end-host certificate. This is very unfortunate as intermediate certificates are very common. RFC 6961 is meant to address this but was only published in mid-2013 [114].

Revocation practices in browsers

Theory and practice are often two different things, and this is also the case for revocation in X.509. If certificates do not carry information where to find proper revocation information, they are effectively irrevocable, i.e., valid for their entire lifetime. In an investigation carried out by Netcraft [182], a number of mistakes made by CAs are listed. A CA operated by American Express was one example where revocation information was completely lacking. Google's CA (a subordinate CA of Equifax) did not include OCSP URLs, which effectively disabled revocation checks by Firefox as this browser only checks CRLs in the case of EV certificates. For all other certificates, it will carry on normally and establish the connection, without warning. Netcraft stated

[10]The blog entry does not specify which CA does this. Symantec owns at least five brand names: Equifax, GeoTrust, TrustCenter, Thawte, and VeriSign.

that this was particularly discomforting as Google's own CRL listed seven certificates with reason 'key compromised'. Google was not the only offender—Netcraft had also found such certificates from Symantec, Verizon, GlobalSign, and Microsoft.

Browsers do not necessarily carry out revocation checks as intended, either. Netcraft demonstrated the shortcomings of browsers in a blog post in May 2013 [181]. The problem they investigated was a revoked intermediate certificate by RSA. The revocation was meant to be effective immediately. However, as the blog post pointedly remarked, *'more than a week later nobody had noticed [and] the certificate was still in place [on the server]'*. The reason was that RSA did not provide an OCSP revocation service and had only put the certificate on a CRL. Even if OCSP had been available, however, Netcraft notes that Firefox does not check the validity of intermediate certificates anyway. Only the browsers Internet Explorer and Opera carried out proper verification checks and prevented access to the Web site. However, browser caching behaviour might still have broken the verification: Netcraft judged most browsers would cache the CRL for six months.

Furthermore, as Google engineer Adam Langley points out in a blog post [208], so-called 'captive portals' make it practically impossible to carry out OCSP checks. The term 'captive portal' refers to network setups as they are common in, e.g., hotels, which require users to register before allowing them unrestricted access to the Internet. As wireless access points often do not use encryption, the registration has to be sent via HTTPS to the registration server (which is initially the only host that can be accessed). OCSP requests are generally not forwarded, and thus users cannot identify a revoked certificate. This unsatisfying situation led Google to be the first vendor to change their revocation support in their browsers. Langley announced that Google Chrome would no longer support CRLs nor OCSP in the future. Instead, Google would maintain a list of revoked certificates and distribute these to the browser via their normal software update mechanism [208]. The list would be obtained by scans of CRLs. The obvious question is whether this concept can scale. It is exceedingly difficult even for Google to maintain a complete list of revocations and distribute them. However, it is safe to assume that most users never access large portions of the Web. This means that Google can focus at first on important Web sites, like those on the Alexa Top 1 Million list of popular Web sites [133] in order to deliver a level of protection that is better than the current one.

2.7.3. Revocation in OpenPGP

Revocation in OpenPGP is different from X.509 revocation. First of all, revocation in OpenPGP cannot be carried out by any entity other than the key owner himself—otherwise, any malicious participant could mark keys as unsuitable for use by others. To revoke a key, the key owner issues a so-called revocation certificate. This is a simple statement that the indicated public key must be considered revoked and not be used any more, signed with the associated private key. The revocation certificate can then be propagated to all interested and affected parties. Optimally, the revocation certificate should be created right after the creation of the key pair itself, and then stored in a secure location until it is needed. It must never be lost, or revocation of the key is impossible. The fact that key servers never delete keys from the Web of Trust makes this worse.

The fact that the revocation certificate must be distributed to affected parties is also a serious problem. While users can inform those they know to use their key, they cannot reach out to unrelated parties who might download the public key from key servers and use OpenPGP's trust metrics to establish a first, marginal trust into the public key. Key servers allow to optimise this as they are able to store revocation certificates, too. Just as in X.509, users would then have to remember to verify that a key has not been revoked every time before using it.

Finally, it is worthwhile to think about the damage that a successful attacker can do in OpenPGP and in X.509. In X.509, domain owners generally must register with a CA out-of-band (usually on a Web site) before they can request certificates. This access is usually protected by password, and thus there is a second secure channel available for the domain owner to request revocation of a certificate. Furthermore, as long as this password remains uncompromised, an attacker cannot obtain further certificates from the CA. He is not able to use the private key to certify other entities, either. This is not true in OpenPGP: an attacker in possession of the private key can spread some damage through the network until he is detected.

On summary, the security of revocation in OpenPGP depends on the security of the revocation certificate and how fast a key owner is able to distribute it. Despite the flaws in implementation, revocation in X.509 can limit the potential damage. However, one should also note that this issue may not be so devastating if a Web of Trust is used within smaller communities: the possibility of personal contact between users may help mediate the problem.

2.8. Text adapted from previous publications

This chapter contains text from the sections on related work and background of the following previous publications:

- Alexander Ulrich, Ralph Holz, Peter Hauck, and Georg Carle. Investigating the OpenPGP Web of Trust. *Proc. 16th European Symposium on Research in Computer Security (ESORICS)*, Leuven, Belgium, September 2011 (reference [73]).
- Ralph Holz, Lothar Braun, Nils Kammenhuber, and Georg Carle. The SSL Landscape—a thorough analysis of the X.509 PKI using active and passive measurements. *Proc. 11th ACM SIGCOMM Internet Measurement Conference (IMC)*, Berlin, Germany, November 2011 (reference [37]).
- Oliver Gasser, Ralph Holz, Georg Carle. A deeper understanding of SSH: results from Internet-wide scans. *Proc. 14th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Krakow, Poland, May 2014 (reference [28]).

The contributions to the papers made by the author of this thesis are listed in Chapter 4 (for [37]), Chapter 5 (for [73]), and Chapter 6 (for [28]).

Section 2.4 is an extended version of Section 2 in [37]. The author added the sections on history and use in TLS. He significantly extended the sections on certificate structure and CA hierarchy. Section 2.5 is a revised version of Section 2 in [73], with clarifications and details added by the author. Section 2.6 is an extended version of Section II in [28]. The author added the protocol flow of SSH and added details on the PKI model.

Part II.

Analyses of Public Key Infrastructures

Chapter 3.

3 Analysis of the weaknesses of the X.509 PKI

The deployment of X.509 for the WWW began in the 1990s with the first browser vendors adding CA certificates to their root stores. Criticism of this approach can be found as early as the turn of the millennium. From about 2008 on, we also find increasing evidence of successful attacks against the X.509 PKI.

This chapter addresses Research Objective O1. The approach we take is historical-documental, with empirical elements. We analyse early criticisms and investigate whether these were addressed later. We then investigate reports of known attacks (both successful and unsuccessful ones) and determine the root causes. From this, we derive requirements that mechanisms to strengthen X.509 have to meet in order to be successful.

3.1. Investigated questions

In keeping with Research Objective O1, we define the following research tasks.

Criticism and mediation We investigate which major criticisms were brought forward, and whether these were addressed. This covers Research Objective O1.1.

Incidents and their causes We investigate incidents in the X.509 PKI and determine their root causes. This task addresses Research Objective O1.2.

Improvements to X.509 Based on the above findings, we derive which kind of approaches can help mitigate the above identified causes for failures. This addresses Research Objective O1.3.

3.2. Criticism of the design

In the following, we describe some major criticisms of X.509 that were brought forward from early on and analyse whether these were addressed.

3.2.1. CA proliferation: the weakest link

The original X.509 PKI envisaged one global CA with many subordinate CAs responsible for clearly outlined portions of the X.500 directory. X.509 for the WWW eliminated this principle: here, any CA may issue a certificate for any domain. When discussing different PKI setups, Perlman identified the critical weakness of this approach in [61]: compromising a single CA is enough to break the entire PKI. Yet CAs in the root store are not the only ones that are trusted. As mentioned in Section 2.3, CAs may employ Registration Authorities (RAs) to identify the entity making a certificate request. Since the identity verification is a crucial step in certificate creation, this means that there are more entities that need to be ‘trustworthy’ than the size of a root store suggests. This was recognised early on in publications, e.g., by Ellison and Schneier [23], and also in academic talks, e.g., by Chadwick [156]. The obvious CA for an attacker to target would be the one with the weakest protections. As a result, the strength of the entire PKI is equal to the strength of the weakest CA.

It is thus an interesting question whether these early criticisms of CA proliferation were addressed or not. One way to measure this is to analyse the development of the root store of a major browser vendor. We chose to do this for the Mozilla Firefox browser, whose source code is publicly available. It takes its root store from the NSS cryptographic library, which it uses for TLS [230]. We downloaded all revisions of NSS from the CVS development branch[1] [233]. This branch holds the root stores between late 2000, when the first certificates were added, and late 2012. It is a very good approximation of the state of Firefox’s root store over the years[2]. For each root store, we determined the number of included root certificates. Our tool suite [200] is an extension of another tool to extract the root store, written by Langley [211]. It allows us to extract the root stores not only from the current Firefox versions, but also from older versions[3].

The result is shown in Figure 3.1, which shows the development of the NSS root store since 2000. As can be seen, the root store was initialised with only about a handful of certificates. Since then, the number has greatly increased over the years. By December 2010, we find the number to have grown to almost 120. By the end of 2012, it had reached 140 root certificates. Recall that each certificate is associated with a private key that must be protected at all times in order to avoid compromise of the entire PKI.

The significance of this count alone is somewhat limited, however. Another interesting question would be how many organisations exist that own root certificates in the root store. This question can be answered with data available from Mozilla—the organisation discloses the owners of the root certificates in a publicly available document [231]. As of 7 November 2013, we find the root certificates belong to 65 organisations, which identify themselves via 90 different names in the ‘organisation’ part in the issuer field of the certificate[4]. We also investigated the requests for inclusion of new root certificates, which are also published by Mozilla [232]. We found 41 owners applying for the inclusion of a new root certificate. Seven owners applied for an update of their certificate’s status in the root store (e.g., to enable EV treatment). Note that this list also contains already-approved requests, which have just not been added into the source code of NSS.

As a note aside, even browser vendors sometimes find it hard to track root certificates across organisational changes (mergers, acquisitions, introduction of new operational procedures, etc.). Mozilla, for example, had to admit in 2010 that they had difficulties determining whether the owner of a root certificate in their root store was still active [273, 238]. The issue could be cleared up within weeks: the company RSA still held the private key, but the certificate had been retired. The case demonstrates the considerable effort that must go into the administration of root stores.

[1] `cvs -d :pserver:anonymous@cvs-mirror.mozilla.org:/cvsroot/ co mozilla/security/nss/`

[2] There is no centrally maintained list which NSS revision was used in which Firefox version, unfortunately. Firefox versions have a different release schedule than NSS and may skip the odd NSS version. However, the root stores are the same and the delay in introducing new CAs into Firefox is too short to be significant for our analysis.

[3] Note that this method differs from the one we used in [37] as we count only root certificates that are declared to be used to issue server certificates here.

[4] We discounted two certificates where the organisation part was empty.

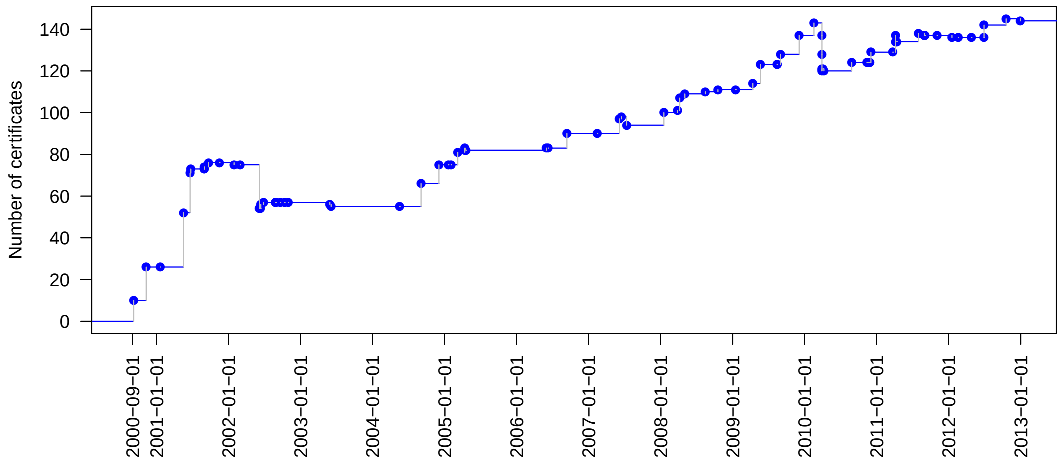


Figure 3.1. – Growth of the NSS/Mozilla root store.

Our method can merely estimate a lower boundary on the number of entities that can sign certificates. The root store does not list subordinate CAs, which may be operated by other organisations. One could be tempted to determine the names of organisations from large-scale scans of the HTTPS infrastructure. However, this method has another caveat: it is impossible to tell from an intermediate certificate whether it is used by a subordinate CA or whether it just indicates that a certain RA verified the identity of the subject named in the certificate. For example, it is known that the German association DFN-Verein, which itself operates as a subordinate CA, employs a large number of RAs. These RAs appear in the issuer strings of intermediate certificates issued by DFN-Verein. However, they do not control the cryptographic material: all certificates are issued by the DFN-Verein subordinate CA itself [190]. It would thus be incorrect to count the RAs as entitites that can issue certificates—only the one subordinate CA should be counted.

Finding The numbers for both certificates and organisations show that CA proliferation has continued over the past decade and there is no sign that it will stop, at least in the short or even medium term. The weakest CA determines the strength of the entire PKI.

3.2.2. Trust and liability

Browser vendors and operating systems decide which root certificates should be included in their root stores. Users as the parties that rely on the correctness of certificates are not involved in these decisions. It is thus an interesting question whether CAs can be held liable for accidents, compromises or mistakes they make. This point was first investigated by Ellison and Schneier in [23] in 2000. At that time, PKI practices were far from being standardised, but CAs did already document abstract duties and responsibilities in Certification Practices Statements (CPSs). Ellison and Schneier analysed the CPSes of several CAs. They found that not one of them accepted liability in any form for an issued certificate. Consequently, trust into CAs could be said to be established by browser vendors without a legal way for users to hold anyone accountable for damages.

Generally, vendors require security audits according to a standard like WebTrust [171]. Some set additional requirements. Mozilla documents their requirements in [229], Microsoft in [223]. The question is if compliance with an audit is enough to constitute trust in a CA, especially as the audited parties are exactly the parties that pay for the audits. However, PKI practices have moved a step further towards standardisation. The so-called Baseline Requirements of the CA/Browser Forum impose policies on the certification practices of the forum members [148].

With such documents in existence, one way to assess the trustworthiness of a CA is to consider the degree of liability they prescribe: a high liability means that the CA will at least pay damages if any incident should harm the customer. This should give it a strong incentive to adhere to good secure practices. We thus inspected the Baseline Requirements. We found that not very much had changed since Ellison's and Schneier's investigation: the standard continues to allow a CA to disclaim any liability. There is not even a need to mention this in the CPS[5]. As of 2013, a separate set of guidelines exists for EV certificates. The guideline document by the CA/Browser forum allows CAs to limit liability to 2000 USD per 'relying party' (e.g., users) or 'subscriber' (e.g., domain owner), and per EV certificate [149, p. 40].

Finding While users are parties that need to rely on the correctness of the PKI, we find that they still cannot hold CAs liable, except in the case of EV certificates. Even then, however, CAs can limit their liability to 2000 USD. Liability is only a small incentive for CAs to operate in a way that makes them more trustworthy.

3.2.3. Strength of identity verification

Ellison and Schneier also expressed concern about what they call a CA's 'authority' [23]—based on the fact that CAs certify public keys for domain names without actually having control over DNS registration, and thus being unable to verify the identity of the party requesting a certificate with enough certainty. We analysed current practices, based on the Baseline Requirements and the EV requirements of the CA/Browser Forum.

The Baseline Requirements distinguish between the authorisation of a domain owner to apply for a certificate and the verification of the identity of the domain owner in terms of name and address. Concerning the authorisation, the document specifically allows CAs to use email to verify that a requesting party is authorised to request a certificate. The email address to use may be either taken from the WHOIS, be a 'standard' address like `hostmaster` prepended to the domain name, or be supplied from the domain registrar as a recognised authority [148, p. 14]. However, email still relies on the insecure DNS, which thus represents a weak link in the chain of verification steps to carry out.

For certificates where the requester also wishes to add name and address, the Baseline Requirements require out-of-band steps. Examples are attestation letters, site visits or communication with a government agency. This establishes a level of verification between the email-based domain-verification and EV certificates. Recall that the latter are intended for organisations only. The respective guidelines for EV certificate issuance are described in a document of their own, and often require contact with entities recognised by a jurisdiction's government for such purposes [149, p. 20–21]. Interestingly, email contact is cited as one way to communicate with such entities, at least in some cases [149, p. 20].

[5] At the time of writing, the liability clauses remain in effect even for the now-updated version of the BR.

Finding The guidelines published by the CA/Browser Forum establish (at least) three different classes of certificates. Some certificate types require a relatively thorough verification, which takes contact with government agencies into account. However, the 'domain-validated' certificates still rely on insecure technology during the verification process.

3.2.4. Summarising view

The previous sections demonstrated that a number of weaknesses were known to exist in the X.509 PKI from an early point on. We showed that at least two of them, namely CA proliferation and liability, were not mediated since the academic publications that described them.

Concerning CA proliferation, we found that the number of root certificates continues to grow as evidenced by an analysis of the Mozilla root store for the time between late 2000 and end of 2012. Furthermore, the number of organisations owning the root certificates is also relatively high, namely about 60—and another 40 organisations are waiting to have their certificates included in the root store.

Concerning liability, we analysed the current guidelines for the issuance of certificates that have been agreed upon by members of the CA/Browser Forum. We found that CAs may disclaim any liability, except for the case of EV certificates. However, the liability even in this case is relatively low.

We found that the third issue, the use of insecure technology to verify identities, has been addressed only partially and only for a certain class of certificates.

On the whole, it is safe to say that developments in the X.509 PKI have been rather slow, with important issues not addressed for many years. In particular, it seems unlikely that the issue of CA proliferation will be addressed in the near future. With CAs having equal certification capabilities, the argument of the X.509 PKI being as strong as its weakest link will continue to hold.

3.3. Incidents and attacks against the X.509 PKI

In this section, we take a documental approach: we compile a survey in which we analyse known attacks and incidents against the X.509 PKI. In particular, we will investigate the root causes. To the best of our knowledge, this is the first academic survey covering this topic.

Figure 3.2 shows a timeline of events that may serve as a graphical reference. Our results are mostly based on the analysis of publicly available sources (like reports from hackers) and incident reports (from CAs themselves or companies hired to carry out an investigation). We also briefly analyse the extent to which cryptographic breakthroughs endangered the X.509 PKI. We conclude with a relatively new danger to X.509 and TLS: the appearance of devices that allow to carry out mass surveillance.

3.3.1. Attacks against the X.509 PKI

The order in which we present incidents will be only roughly chronological: in some cases, incidents occurred before others but were not disclosed until later, or they occurred but it was not realised then that they were linked to other incidents. We thus sometimes deviate from the chronological order to make such links more transparent.

Microsoft code-signing certificates (2001)

The first case we discuss gained a certain prominence as it affected Microsoft. The Microsoft Security Bulletin that describes the incident can still be found online [220].

In 2001, the VeriSign CA erroneously issued certificates to an unknown party that claimed to be Microsoft. These certificates could be used to sign code. They were not issued for domain names, thus a man-in-the-middle attack on traffic was not possible. Microsoft stated their real certificates were issued from a different CA and that the rogue certificates would not be accepted by its Windows operating system.

Finding The incident showed that it was possible to trick a well-known CA into issuing certificates for the wrong entity. The weakness was the insufficient identity check on part of the CA. It remains unclear at which point in the certification process the mistake was made.

Thawte (2008)

Attacks on the X.509 PKI began in more seriousness in 2008, marked by a series of proof-of-concept exploits against major CAs. These were carried out by white-hat hackers, i.e., hackers with benign intent. In summer 2008, Zusman announced that he had been able to obtain a certificate for `login.live.com`, an important site by Microsoft [277]. Combined with a man-in-the-middle attack, this would have allowed him to intercept critical traffic to the site.

The attack vector was simple. Zusman exploited the fact that CAs assume certain email addresses can only be owned by an administrative contact, i.e., someone with a claim to the domain. As mentioned in the last section, this practice is still allowed by the Baseline Requirements. Examples of such addresses are `root@example.org`, `postmaster@example.org`, etc. However, at that time no comprehensive list of such addresses had been agreed upon between CAs and other parties. This was critical in the case of Web mail services, however, as these allow email address to be freely registered under the domain name of the service. In this particular case, Thawte[6] viewed `sslcertificates@live.com` as a valid administrative contact. This address was still available for registration. Zusman acquired the email address and then bought a certificate for `login.live.com`. His blog entry implies that he even spoke to Thawte personnel on the phone, who did not notice the high value of the domain name nor thought of additional verification steps.

Finding The weakness consisted of a lack of agreement between server operators and CAs which email addresses constitute a valid contact. The root of the problem, however, is that email contacts are a poor way to assert an entity's ownership of a domain.

Comodo (2008)

A serious incident in the same year concerned the Comodo CA, a relatively large CA. It was reported by Eddy Nigg, founder of the competing CA StartSSL, in a blog post [236], on 23 December 2008. Comodo operates resellers and Registration Authorities that are allowed to verify ownership of a domain name themselves. Nigg found that one of these resellers sent him unsolicited advertisements to buy certificates from them. He tested whether he could get a certificate for `mozilla.com`, and found

[6]Zusman did not immediately announce the name of the CA; he disclosed this information several months later [276].

that he could buy the certificate without any verification checks applied at all. He reported this violation of a CA's most basic obligation on his blog and in the newsgroup `mozilla.dev.security.policy` [235], where Mozilla carries out its root certificate inclusion process. The issue was also discussed in Mozilla's bug tracking system [146]. The entry contains an incomplete description of the rather complicated system of RAs, resellers and subordinate CAs that Comodo operates.

Comodo's failure to guarantee their RAs' proper functioning raised the question whether their root certificates should be removed from Mozilla's root store. There was no precedent for this. Comodo themselves deactivated the RA. After discussion in the newsgroup and the bug tracker, Mozilla chose to interpret this as sufficient action on Comodo's side. No further steps were taken.

Finding The weakness was an RA failing to comply with their duties in spite of contractual obligations. It shows that the model of RAs is dangerous if the RAs cannot also be technically controlled.

StartSSL (2008)

StartSSL itself was also a victim in 2008. The incident was reported two weeks *after* the previous one, but actually occurred three days *earlier*. On 20 December 2008, an attacker exploited a flaw in StartSSL's Web interface to issue a number of certificates, among them the domain of the online payment operator PayPal and the CA VeriSign. The latter was caught by StartSSL as the CA maintained a list of domains it considered sensitive. Certification requests for these domains were automatically forwarded to human personnel for manual inspection (but after the issuing process had completed). This second line of defence caught the intruder, and all rogue certificates were instantly revoked.

StartSSL contacted the intruder via the email address he had given during the registration process—it was Zusman, who had also previously tricked Thawte. Zusman reported his success in a blog post [279]. StartSSL documented it in a white paper [261]. We inspected both texts. There is no unambiguous description in StartSSL's report how the attacker managed to break the registration process. Zusman cleared it up in a talk at DEF CON 17 [278] and identified the weakness as 'Insecure Direct Object Reference'. This vulnerability type had been in the Top 10 list of weaknesses compiled by The Open Web Application Security Project (OWASP) just the year before. StartSSL also became the centre of debate in the Mozilla newsgroup. Mozilla decided to keep their root certificate in the root store.

Finding The weakness lay in a serious technical vulnerability in the front-end which may have been there for an undetermined period of time. The incident was contained thanks to the CA's defence-in-depth approach, a manual check for sensitive domains.

RapidSSL (2010)

In 2010, Seifried reapplied the method Zusman had used previously to obtain a certificate from Thawte. His target was the RapidSSL CA (owned by VeriSign). Seifried registered `ssladministrator@portugalmail.pt` from the Portuguese Web mail provider of the same name and found he could normally buy a certificate for the domain.

He documented this finding in a post to the newsgroup `mozilla.dev.tech.crypto` and an article in Linux Magazine [257, 256]. A bug in the Mozilla bug tracker was consequently opened [212]. It is worthwhile to note here that the attack vector was long-known and had been exploited before. The CA was even aware of it: an entry in Mozilla's bug tracker documented that the CA had promised one year before to disallow such addresses [262]. Technically, however, they were not at fault—the grace period that the CA had been given to address the issue was not completely over yet when Seifried carried out his certification request.

Finding The weakness was identical to the Thawte incident of 2008. However, it also shows that even security-critical organisations like CAs can sometimes exhibit a slow pace of change.

Comodo (2011)

2011 could be said to be the year when the X.509 PKI was broken the first time with very serious consequences. On 15 March 2011, a Comodo RA was compromised and the attacker was able to issue rogue certificates. Comodo themselves acknowledged this fact about a week later [159]. The intruder himself posted a letter to Pastebin (a site for sharing text fragments, with anonymous access) and identified himself as a hacker from Iran [164], providing details about the methods he had used to issue the certificates and secret information whose authenticity only the RA could confirm—passwords and database names among them. Comodo broadly acknowledged some of these details later [214].

The intruder identified the Italian branch of InstantSSL as the reseller he had broken into. Once on the server, he could decompile a library, which allowed him to determine Comodo's online interface for certificate signing and gave him the access credentials. Note that this means that the reseller was essentially a subordinate CA, not just an RA. The attacker used the credentials to issue nine certificates. High-value domains were among them, e.g., Google, sites of Yahoo, Skype, and Microsoft Live, as well as the Mozilla Add-on site. The latter is of particular note: together with a man-in-the-middle attack, it would allow to trick users into installing forged add-ons, which has the potential to compromise their entire system. The purpose of another certificate, with the subject 'global trustee', was never ascertained. In two separate posts to Pastebin over the next two days, the intruder presented more details [160, 163]. Shortly afterwards, he released the rogue certificate for Mozilla [162] and indicated that he had compromised at least two more Comodo resellers [166]. Comodo confirmed the compromise of two more 'RA accounts' [132], although this leaves open what kind of accounts were compromised. A blog post by Netcraft later identified GlobalTrust as one compromised 'RA' [234].

Comodo noticed the incident themselves and reacted by revoking the affected certificates. According to their report [159], the certificates were not detected in use, apart from one 'test' on a host in Iran. The company claimed that *'the attack came from several IP addresses, but mainly from Iran'* [159]. Comodo also notified other affected parties, in particular browser vendors. The first public notice did not come from Comodo, however, but from Mozilla [240] and the independent security researcher Jacob Appelbaum [137]. He had monitored changes to the source code in the repositories of Chromium (the open-source project behind Google's Chrome browser) and Mozilla and noticed that the two organisations had both blacklisted (roughly) the same certificates in their source code. Upon inquiry, he was told of the incident and agreed not to disclose it until patches had been shipped. Mozilla disclosed the incident on 22 March, Microsoft and Comodo on 23 March. The changes to the repositories had both happened on 17 March. In other words, between the compromise and the fix two days had passed. It took five more days until the incident was finally publicly disclosed.

Browser vendors reacted by marking certificates as untrusted as part of their code. Revocation mechanisms like CRLs or OCSP were not sufficient. First, a root of the

PKI had been successfully attacked. Second, as mentioned in the last chapter, browsers generally do not fail promptly on revocation errors—i. e., they do not abort a connection if they are unable to obtain revocation information. If the certificates had only been revoked, an attacker would still have been able to use them if he had been able to suppress CRL or OCSP traffic, too. Apart from removing Comodo's root certificate, blacklisting certificates in code was thus the only way to make the rogue certificates useless for an attacker.

Finding The weakness here was of a combined technical and organisational nature. First, a weak link existed in the form of an RA. Second, the reseller's public-facing systems were vulnerable, and the attacker could escalate his privileges to the point where he could issue certificates. Third, the attack showed that there is no clear distinction between RA and subordinate CA—the 'RA account' was equipped with access rights to Comodo's signing systems.

DigiNotar (2011)

The series of incidents culminated in the breach of the DigiNotar CA. DigiNotar participated in the Dutch government's PKI, PKIOverheid, allowing it to issue government-accredited certificates. The CA was allowed into the Mozilla root program in May 2008[7][8]. The incident that concerned DigiNotar was investigated by the security company Fox-IT, on whose final report we rely in the following [192].

On 19 July 2011, DigiNotar found a mismatch between their list of issued certificates and records kept in their back office. An investigation yielded that rogue certificates had been issued. In the week that followed, more such rogue certificates were found and also a message from the intruder. DigiNotar revoked the rogue certificates and consulted an external security firm. DigiNotar did not inform browser vendors; the company believed the damage had been contained by revoking all rogue certificates they had found.

It turned out that this assessment was wrong. On 27 August 2011, a user from Iran posted a message to Google's Gmail product forum, stating that he was encountering a rogue certificate [145]. His browser, Google Chrome, had given a warning when trying to access Gmail. This was due to a largely undocumented feature in the browser: it contained hard-coded checksums for valid Google certificates. This allowed it to identify the rogue certificate and warn the user. The post to the forum contained the rogue certificate and was later enhanced with traceroute information showing an unusual path to Google's servers. The initial post was soon followed by other reports from Iran. This showed that the incident was not limited to one ISP. Rather, everything was pointing towards a state-level man-in-the-middle attack.

DigiNotar learned about this only when it was contacted by the Dutch CERT the next day. The rogue certificate was instantly revoked, and other stakeholders notified. Google published a blog post on the same day [130], stating that they had received several reports about ongoing man-in-the-middle attacks with the fraudulent DigiNotar certificate. They announced their plans to remove DigiNotar from their root stores. Mozilla was informed by Google and also reacted on the same day, shipping emergency versions of their software with the DigiNotar CA disabled [241]. Microsoft reacted in the same way [191].

[7]The DigiNotar root certificate appears for the first time in CVS revision 1.48 of the Mozilla source code, which is from 2008.

[8]Interestingly, DigiNotar applied for being allowed to issue X.509 certificates for Web hosts, S/MIME, and code signing. Their verification practices for email addresses did not meet Mozilla's standards for S/MIME, however, and the email trust flags were never enabled in Mozilla.

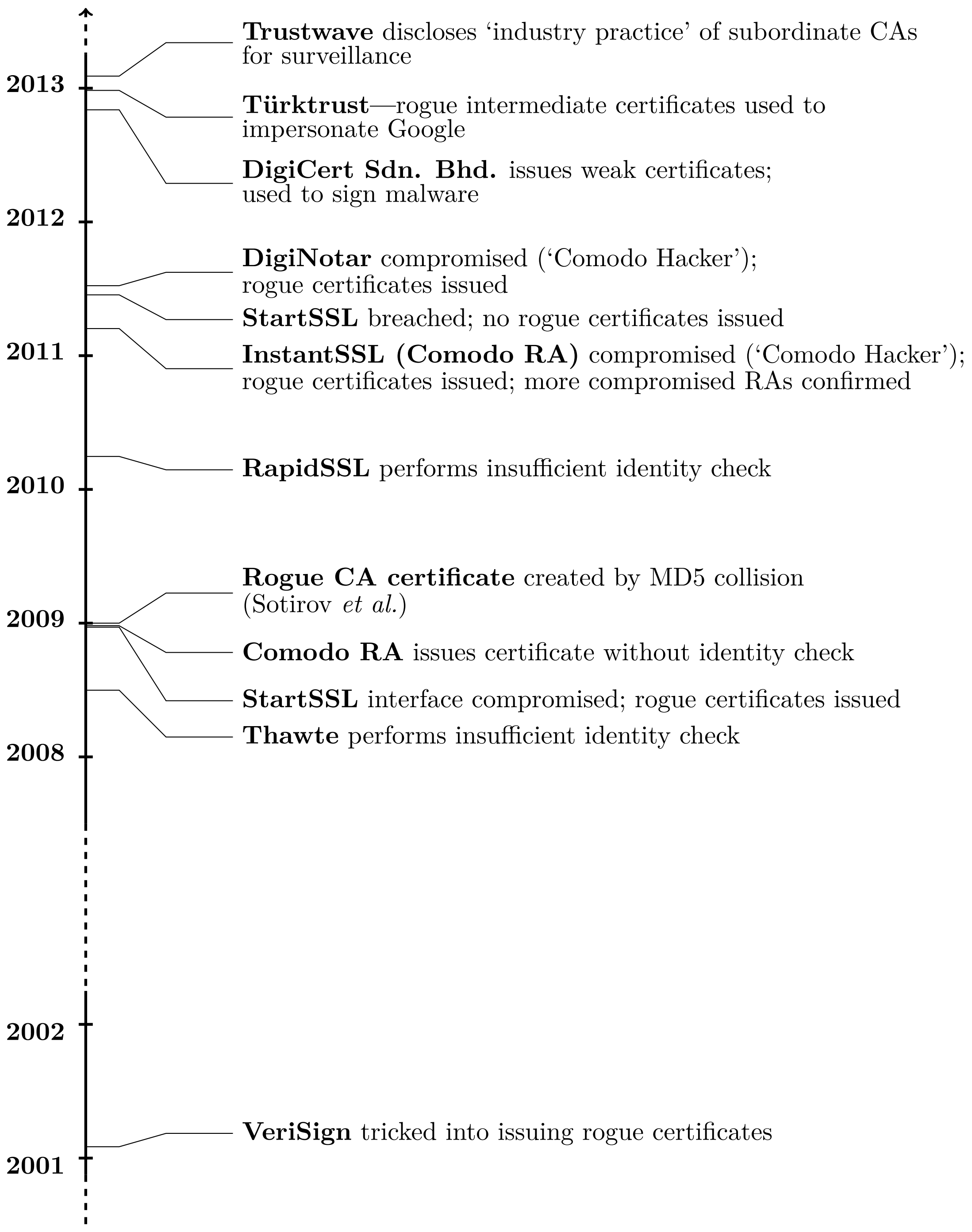

Trustwave discloses ‘industry practice’ of subordinate CAs for surveillance
2013
Türktrust—rogue intermediate certificates used to impersonate Google
DigiCert Sdn. Bhd. issues weak certificates; used to sign malware
2012
DigiNotar compromised (‘Comodo Hacker’); rogue certificates issued
StartSSL breached; no rogue certificates issued
2011
InstantSSL (Comodo RA) compromised (‘Comodo Hacker’); rogue certificates issued; more compromised RAs confirmed
RapidSSL performs insufficient identity check
2010
Rogue CA certificate created by MD5 collision (Sotirov *et al.*)
2009
Comodo RA issues certificate without identity check
StartSSL interface compromised; rogue certificates issued
Thawte performs insufficient identity check
2008
2002
VeriSign tricked into issuing rogue certificates
2001

Figure 3.2. – Timeline of incidents threatening the security of the X.509 PKI for the WWW.

Fox-IT was hired to carry out an investigation on 30 August 2011. The company set up a monitoring system to determine whether the intrusion was ongoing. Fox-IT found that DigiNotar's logs had been tampered with, leaving the company without information about the serial numbers in the rogue certificates, and without a way to assess how many rogue certificates had actually been issued. Initially, this also prevented them from sending correct OCSP revocation responses for the rogue certificates as OCSP is a blacklist-based system. Upon further investigation, Fox-IT found that the attacker had begun to breach DigiNotar's network as early as 17 June 2011. Firewall logs showed connections to IP addresses in Iran.

By 1 July 2011, the sensitive network containing the hosts running the CA software had been compromised. The first successful attempt at creating a rogue certificate was made on 10 July. The investigation showed that all servers that managed CA activities were compromised, with the intruder having administrator privileges and access to databases. Fox-IT could show that the intruder had issued at least 531 certificates, for 140 unique certificate subjects and 53 unique Common Names (e. g., DNS domains). It remains unclear to this day whether more certificates were issued. The last traceable activity of the attacker on DigiNotar's systems was on 24 July 2011. Note that this is a full week *after* DigiNotar noticed the breach *and* consulted a security firm to contain the incident.

Fox-IT found messages left by the attacker on DigiNotar's systems. The attacker claimed to be the same person who had also breached Comodo's RA several months before. He communicated with the public via Pastebin [165] later and for a short time also over a Twitter account [169]. The attacker claimed to come from Iran and act alone [168, 161]. He clarified that he did not work as part of a state-sponsored attack, but acknowledged that he had given rogue certificates to other entities, without stating who they were [167]. The ongoing man-in-the-middle attack was consistent with the attacker's claims, as were Fox-IT's findings of communication with IP addresses in Iran. Among speculations that the Pastebin entries came from an impostor, and not the real intruder, the attacker reacted by using one of the private keys that belonged to a rogue certificate to sign a file [170]. He also claimed to have breached a number of other Comodo RAs, as well as the CA GlobalSign [168, 161]. The last entries on Pastebin are from 7 September and on Twitter from 11 September 2011. GlobalSign later confirmed an intrusion, but stated that it had been limited to their Web server [194], and the attacker had only obtained the private key to the Web server's certificate.

Mozilla reconfirmed their decision to permanently remove DigiNotar from their root store in early September 2011 [239]. Their decision was based on several factors. First, DigiNotar had failed to notify browser vendors of the compromise and the rogue certificates. This was viewed as particularly bad as certificates had also been issued for `addons.mozilla.org`. Second, the scope of the breach remained unknown as DigiNotar's log files had been compromised, too. Third, the certificate had been used in a real attack. This made DigiNotar the first CA ever to be removed from root stores for misconduct.

Finding The DigiNotar incident is of huge interest as so much is known about how the intrusion occurred. The weakness may, at first glance, seem to be of a purely technical nature: the CA had failed to secure its perimeter. However, the effects of intrusion were quite severe. This points at an organisational problem: it took the CA very long before they fully realised the extent of the incident.

StartSSL (2011)

StartSSL became a victim again in 2011. The incident actually occurred before the DigiNotar case, on 15 June 2011, but was found to be connected to DigiNotar. It became publicly known when the StartSSL homepage suddenly announced that the CA would temporarily stop issuing certificates due to a security breach [197]. The CA stated that no rogue certificates had been issued, and the incident was under investigation. In email to a journalist, Eddy Nigg, CTO of the company, said that the attacker had attempted to obtain certificates for the same targets as during the attack on Comodo several months previously. The intruder had unsuccessfully attempted to create an intermediate certificate [195]. In the discussion that followed in the Mozilla newsgroup `mozilla.dev.security.policy`, Nigg maintained that he was currently not allowed to release further details. The discussion led Mozilla to consider changes in their policy, concerning under which circumstances, and how, CAs were meant to notify other parties in case of security breaches.

StartSSL never issued a public incident report. This has remained unchanged at the time of writing and has been confirmed to us in private email. However, details about the incident became known in the context of the DigiNotar case. The intruder mentioned in his post on 6 September 2011 that it had been him who had successfully breached StartSSL. According to his claims, he had already connected to a secure hardware module that was central to certificate issuing. However, Nigg himself had been monitoring the device, and noticed the intrusion. The case was made even more mysterious when Nigg posted an entry on the company blog [237]. He disclosed that his business had ended up in a *'country sponsored cyber-war'*(*sic*). The post mentions the attack on Comodo and cites Iran's involvement, but does not disclose further details. It was not followed up.

Finding The attack was unsuccessful but aimed at a technical weakness. If Nigg's claim of a cyber war should be the truth, this would be proof that the capacities of countries need to be factored into a threat model for the X.509 PKI.

DigiCert Sdn. Bhd. (2012)

In 2012, DigiCert Sdn. Bhd.[9] were found to have issued certificates that were in critical violation of acceptable standards and their own CPS [213]. The CA operated as a subordinate CA for Entrust, Inc. They were found to have issued certificates for very short RSA keys (512 bits), which are not considered secure by today's standards. Furthermore, they had issued certificates without the so-called Extended Key Usage extension, which allows the owner to use them for code signing. The certificates did not carry any revocation information, either. The issue was detected when an attacker cracked the key of one certificate and used it to sign malware.

Entrust reported the issue themselves to Mozilla and announced they would revoke the intermediate certificate. They stated that Verizon had also cross-certified DigiCert Sdn. Bhd., although it was initially not clear what the exact relationship was [189, 213]. Mozilla decided to blacklist the certificate in question [213, 242]. Microsoft and Google followed [222, 206]. DigiCert Sdn. Bhd. responded themselves in a series of statements [177, 178, 176], in which they confirmed the problem and announced they would seek another audit [178]. At the time of writing, they are not in the list of pending requests at Mozilla [232].

[9]Not related to DigiCert, Inc., which is US-based.

Finding The CA did not follow its own operational procedures and issued certificates with weak keys and without a relevant extension. The impact was grave as they operated as a subordinate CA, which once again highlights this particular danger.

Türktrust (2012)

The last incident we discuss concerns a server certificate for `*.google.com`, which was discovered by Google at the end of 2012. This certificate had been issued from an intermediate certificate that had in turn been issued by Türktrust, a CA based in Turkey. Google announced the finding on their security blog [210]. An instance of their Google Chrome browser had blocked the certificate due to the same mechanism that had uncovered the rogue DigiNotar certificate. Google revealed that Türktrust had mistakenly issued intermediate certificates for two organisations instead of what should have been normal server certificates for TLS (this is only controlled by a Boolean flag). Microsoft and Mozilla offered more details [224, 158, 259]. Only one certificate was used to issue the fraudulent certificate for Google's domain. It had been in active use on at least one system. According to Türktrust, the rogue certificates had been issued due to a faulty configuration in August 2011. The CA stated that their logs were complete and that their customers had never requested intermediate certificates. However, one customer had deployed the certificate on their firewall system. The explanation was that this was by accident because the firewall would accept any intermediate certificate automatically and switch to an interception mode. While the claim may seem bizarre, it is not completely implausible and was accepted by the vendors. It was reinforced by Türktrust's statement that OCSP traffic for this particular certificate was predominantly received from IPs associated with the domain in question.

Microsoft, Mozilla and Google disabled use of the intermediate certificates by blacklisting them. Google announced that they would revoke the EV capacity for Türktrust; Mozilla announced it had halted the inclusion of a new Türktrust root certificate meant for the same purpose. Türktrust continues to be included in their root stores, however.

Finding The root causes were the same as in previous incidents— subordinate CAs that were not tightly controlled but able to issue arbitrary certificates. In this particular incident, we would like to emphasise the use of the certificates, however: they were used to monitor encrypted connections.

3.3.2. Surveillance

The case of Türktrust highlighted a particularly severe problem: intermediate certificates used on firewalls and routers for the purpose of inspecting encrypted traffic. It was known well before the Türktrust incident that several companies produce devices for such a purpose—The Spy Files [135] is a collection of documents with the names of companies and products.

It should be noted that some organisations may have a legitimate interest in inspecting traffic that enters and leaves their networks—defence against industrial espionage is one use case. While such practices give rise to ethical concerns, it might be possible to resolve these (e. g., by requiring informed consent from employees). The real concern is a different one: such devices might be used by authoritarian governments to carry out mass surveillance against their own populations. This is particularly troublesome due to the fact that so many CAs exist that are subject to very different legislations. Soghoian and Stamm already warned in [67] that governments might compel a CA registered under its jurisdiction to issue an intermediate certificate for surveillance.

The decisive question is thus how one can ascertain that inspection devices are not used for nefarious purposes. The simplest way would be for a company to create their own root certificate, install it on the border inspection device, and deploy it to client devices. Since the root certificate is only trusted by clients in the organisation's network, no other Internet users are affected. Practical dangers remain, however, when this is done carelessly. In a blog post, Sandvik and Laurie reported a serious flaw in an inspection device by Cyberoam in 2012 [254]—it shipped with a default root certificate that customers could use instead of creating their own one. Unfortunately, certificate and private key were the same across all devices of the same type, and the private key could be extracted as a commenter in the blog post demonstrated [254]. An attacker only had to buy the device in order to be able to read traffic of Cyberoam customers who had not changed the default key. Together with an attack on the Border Gateway Protocol (BGP) as in [199], the attacker would not even need to be in the immediate vicinity of the victim network.

An alternative to a self-created root certificate would be to pay a conventional CA to issue a subordinate certificate. CAs exist that offer such certificates. The CA Trustwave, for example, became the centre of attention when they announced[10] that *'it has been common practice for Trusted CAs to issue subordinate roots for enterprises for the purpose of transparently managing encrypted traffic'* and that *'Trustwave has decided to revoke all subordinate roots issued for this purpose'* [263, 270]. Mozilla, at least, did not welcome such practices and proceeded to explicitly forbid them [274].

Finding Intermediate certificates carry the complete signing power of the root certificate. Instead of compromising a CA, attackers can theoretically just as well buy the certificates they need and then carry out surveillance. Careless deployment of default keys may even enable them to skip this step.

3.3.3. Cryptographic breakthroughs

The number of occasions where the security of X.509 was threatened by cryptographic developments is very small. In fact, there was only one cryptographic breakthrough that had an impact on the PKI: the weakness of the MD5 hash algorithm. Dobbertin showed a way to produce a collision for a modified version of MD5 as early as 1996 [179]. This led to increased efforts to produce collisions in MD5. Wang*et al.* finally presented such collisions in the rump session of Crypto 2004 [76]. These collisions did not constitute practical attacks yet. However, the pace of development increased drastically: Lenstra *et al.* presented two different X.509 certificates with the same hash values in 2005 [47]. This attack on the collision-resistance of MD5 was improved by Stevens *et al.* in 2007 [69]. The authors showed that they could choose input prefixes to MD5 such that two certificates contained different identities yet had the same hash value (a so-called chosen-prefix collision). In 2008, Sotirov *et al.* [260] finally demonstrated that they could create an intermediate CA certificate from a normally issued end-host certificate, which would be accepted by browsers as legitimate. Effectively, this showed that MD5 was no longer a secure algorithm for digital signatures. Work began to phase out MD5 from TLS implementations. Main-stream browsers continued to accept MD5-signed certificates for much longer—e.g., in the case of Mozilla, until 2011 [228].

Finding The number of cryptographic breakthroughs is very small compared to the many incidents that were caused by poor operational practices or technical vulnerab-

[10]The original announcement can no longer be found on Trustwave's homepage. It is quoted here from the post to the newsgroup `mozilla.dev.security.policy` [263], which cites the text from Trustwave's homepage. Reference [270] confirms the correctness of the facts.

Incident	Causes	Detection	Impact
VeriSign (2001)	operational	unknown	minor, if any
Rogue CA certificate (2007)	crypto weakness	disclosure by attacker	none
Thawte (2008)	operational (verification)	disclosure by attacker	Mitm possible, contained
StartSSL (2008)	technology	disclosure by CA	Mitm possible, contained
Comodo (2008)	operational (verification, RA)	disclosure by CA	Mitm possible, contained
RapidSSL (2010)	operational (verification)	disclosure by attacker	Mitm possible, contained
Comodo (2011)	technology (RA)	disclosure by CA	Mitm carried out
StartSSL (2011)	technology	disclosure by CA	none, contained
DigiNotar (2011)	technology	attack detected by Google	Mitm carried out; PKI failure
DigiCert (2012)	operational (issuance, sub-CA)	attack detected	malware signed
Türktrust (2012)	operational (issuance)	attack detected by Google	Mitm carried out
Trustwave (2012)	operational (surveillance, sub-CAs)	disclosure by CA	unknown

Table 3.1. – Summary of X.509 incidents and attacks since 2001. 'sub-CA' is short for subordinate CA. Mitm is short for man-in-the-middle attack.

ilities. It shows that the primary concern for X.509 should be the deployment of the PKI.

3.3.4. Summarising view

The incidents we documented in Section 3.3 exhibited a range of causes. We summarise this in Table 3.1. The impact of the incidents varied greatly, and they were detected by different means. In this section, we are going to provide a summarising view. In reference to Research Objective O1.2, there are two questions we ask. First, what were the most common causes (attack vectors) and their impacts? And second, what were the most common ways an incident was detected?

Going through Table 3.1, we note that *operational practices* were the cause in more than half of all incidents: seven of twelve times. RAs and subordinate CAs proved to be weak links in the overall security of the X.509 PKI. In three cases, CAs or RAs failed to execute their duty of identity verification with due care. In two cases, the problem lay in the issuance process (after verification): CAs accidentally issued certificates with wrong values that allowed the certificates to be used for attacks. The impact of these incidents was always low, however: while man-in-the-middle attacks would have been possible, the incidents were mostly contained. Only the rogue certificates that were due to mistakes in issuance had a measurable impact: in one case, malware was signed; in the other, a localised man-in-the-middle attack was carried out for the purpose of surveillance.

Technical vulnerabilities were responsible for four incidents: StartSSL (2008), Comodo (2011), DigiNotar (2011), and again StartSSL (2011). Of these, two had a very serious impact: Comodo (2011) and DigiNotar (2011). In the case of DigiNotar, the entire PKI was disrupted to the point that browser vendors had to ship emergency updates as X.509 mechanisms, including revocation, failed completely. Notably, only the attack on StartSSL in 2008 was carried out by a white-hat attacker. The other three were carried out with malicious intent. At least in the case of DigiNotar, a large-scale man-in-the-middle attack was detected, which hinted that a country at least profited of the attack, although there is no proof it initiated it.

Finally, we find that *PKI structure* was a cause in four incidents: in the cases of Comodo (2008), Comodo (2011), DigiCert (2012), and Trustwave (2012), either RAs or subordinate CAs made mistakes or failed to comply with their obligations. At least in the case of Comodo (2011), a man-in-the-middle attack seems to have been the result. The revelations made by Trustwave point at a grave problem concerning mass-surveillance, however. It remains unclear how many such subordinate CAs already exist or, worse, are continued to be sold.

Concerning *incident detection*, we find that CAs themselves disclosed incidents and vulnerabilities surprisingly often, namely on five occasions (although they often did not provide many details). However, in the two incidents that had the worst impact, Comodo (2011) and DigiNotar (2011), the disclosure came either late and after independent researchers had already found evidence (Comodo) or the attack was not disclosed at all (DigiNotar). In two cases, ongoing attacks were detected by Google, thanks to their internal mechanism of comparing the certificate in a TLS handshake with internally stored references. In the third event, DigiCert (2012), the attack was not discovered by the CA, either.

3.4. Reinforcements to the X.509 PKI

We address Research Objective O1.3 in this section: based on our historical analysis from Research Objectives O1.1 and O1.2, we draw conclusions what effective reinforcements for the X.509 PKI should look like. In keeping with Research Objective O1.2, we are going to structure our discussion around the following two aspects:

- Resistance against the attack vectors determined in the historical analysis
- Detecting and reporting an incident

3.4.1. The impossibility of direct defence

The primary attack vectors we determined were operational practices and technical vulnerabilities. This was made worse by the existence of subordinate CAs and RAs in the X.509 PKI. The first conclusion we would like to offer here may sound surprising at first: it is impossible to defend against these vectors *directly*. We elaborate on this in the following.

Operational practices The current CA system is built on root programs run by vendors of browsers and operating systems. These programs generally require audits and compliance with policies in order for CAs to be included in a root store. Yet we found that CAs repeatedly violated policies they had agreed to. As subordinate CAs and RAs are, in contrast to CAs, not documented in the public documents of a root program, they are an obstacle in the effective detection of incidents. A report by Netcraft that was published very recently supports our argument: Netcraft found that one year after the Baseline Requirements of the CA/Browser-Forum had come into effect, a number of CAs were still issuing certificates that were in violation of the agreed practices [180]. Netcraft reported keys that were too short, certificates that missed revocation information, or showed wrong use of the CN field. In total, Netcraft reported 2500 cases.

Some of the largest CAs were among the offenders. Against this background, and our previous findings, the position we take is that operational practices will continue to be a problem, at least in the medium term.

PKI structure The above argument is intrinsically linked to the issue of the *X.509 PKI structure*. The compromise of one authority means the compromise of the entire structure. RAs and subordinate CAs can be useful for better identity verification, but they remain the weakest links in the PKI structure. Even if they were entirely removed, however, we remain in doubt whether this improvement would have a larger impact: a large number of CAs with equal signing rights continues to exist—our analysis of the Mozilla root store has shown that. Any improvement in the short or medium term must take the existence of a large number of equal authorities into account.

Technical vulnerabilities Concerning *technical vulnerabilities*, we take the view that flaws in software will always exist, and thus also exploitable vulnerabilities. CAs, by their very nature, must operate public-facing systems that accept input from untrusted sources. This attack vector is next to impossible to close.

3.4.2. Improving defences

The second conclusion we offer is the following: the above does not mean at all that we are defenceless, nor does it mean we need to abandon X.509 immediately. There are at least three ways to address the shortcomings of today's X.509 PKI.

Out-of-band mechanisms can help First, if we cannot address the problems of the X.509 PKI directly, we must design *mechanisms that raise the security for users considerably*, even against very strong attackers. These mechanisms must be designed to work out-of-band, i.e., without relying on X.509 mechanisms themselves. The reason is that users in the above described cases had no way to assess the authenticity of a certificate other than evaluating the certificate itself. The compromise of a root CA made it impossible to use revocation mechanisms. Some out-of-band mechanisms to improve X.509 have been proposed. They fall into several categories. We analyse the most important contributions in Chapter 8.

Incident detection to aid containment Second, we suggest to return to the old wisdom of computer security that good defences are not only concerned with thwarting an attack. Rather, one should expect that defences are breached and have established powerful reactive mechanisms to contain the damage. This should not be left to single CAs as the compromise of one is damaging for all. We thus suggest that mechanisms to *detect incidents in the PKI* are the most important step in containing damages. We saw in the last section that some of the more dangerous incidents were not disclosed by the CAs themselves (or only belatedly) but by third parties, which had a legitimate interest in thwarting the attack. Designing mechanisms for detection from the outside is a key factor in improving the X.509 PKI. As part of this thesis, we designed a detection system, which we present in Chapter 9.

Outside pressure Finally, we believe it *is* possible to exercise pressure on CAs to cause them to improve their operational practices—after all, a bad reputation is a poor basis for economic success. One way to mount this pressure, and at the same time increase the security of the X.509 PKI as a whole, is to *monitor the deployed PKI*. One case where this would have helped are the certificates with short key lengths issued in the case of DigiCert Sdn. Bdh. Monitoring the deployed PKI has a further benefit: it makes it possible to detect configuration problems on servers and notify administrators. In the next chapter, we show that monitoring can indeed unveil poor (security) practices, both on the side of servers as well as on the side of CAs.

3.5. Related work

Our analysis of X.509 criticism is mostly based on the work by Perlman [61] as well as that of Ellison and Schneier [23]. Both publications provide significantly more detail than we discussed here. Perlman discusses a wider range of PKI setups and trust models; Ellison and Schneier also discuss issues related to usability (for users) as well as end-host security (for client computers and client authentication). We chose these two publications as they can be said to be highly accurate where they identify issues in X.509. They also cover all aspects that are important in our discussion. Naturally, there are many other publications available. Gutmann, for example, discusses improved PKI setups in [32], focusing on the revocation issue. In [33], the same author makes suggestions to improve X.509 based on interviews with programmers, administrators and managers. Chadwick also identified several issues in a talk at EuroPKI 2004 [156]. In [22], Ellison addresses the usability aspect again and references principles established within SPKI [100], a PKI that never saw wide deployment but took an entirely different approach by replacing the concept of global identifiers with that of local names that can be chained.

There is very little related related academic work available that gives a historical assessment of incidents in the X.509 PKI and analyses the root causes. Asghari *et al.* reported on several CA incidents in [7]. They also identified the 'weakest link' in the PKI structure as a serious flaw. Their work focuses on an analysis of the interplay between economical incentives and market forces and thus does not discuss systemic vulnerabilities beyond PKI structure. Notably, however, the authors show empirically that certificate pricing is not a driving market force. Roosa and Schultze [65], in a legal publication, also identified the same weaknesses in the CA trust model as Perlman and Ellison and Schneier and cite some incidents as examples.

Ian Grigg compiled a list of incidents from 2001 to 2013, which he announced in the `mozilla.dev.security.policy` newsgroup [196] and published in a wiki at [151]. The text gives only a brief description of each incident, but contains most of the incidents discussed by us. A similar list was published in another wiki by The Hacker's Choice at [265]. The list is significantly shorter than both the one at [151] as well as ours. It was compiled in the wake of disclosures about mass surveillance carried out by government agencies.

We know of two books that are work-in-progress and contain sections on historic CA incidents. The first is [86, p. 39–44] by Gutmann. The cited section provides an entertaining summary of CA-caused incidents. It does not analyse the root causes systematically. The other book is [89, p. 11-34] by Ristić. Ristić gives an (often shorter) overview of many of the same CA incidents we discussed, but does not analyse root causes, either.

3.6. Key contributions of this chapter

This chapter addressed Research Objective O1. We conclude this chapter with a list of the key findings.

Research Objective O1.1

We analysed historical criticism of the X.509 PKI. Three major criticisms were:

CA proliferation The number of CAs is high, and growing. This increases the number of weak links.

Lack of liability CAs have very little liability towards relying parties (customers, users), if any at all.

Strength of identity verification CAs rely on insecure communication to carry out identity verification or may lack the proper means to do so.

We investigated whether these issues were addressed. We found that no solutions had been introduced for the first two, CA proliferation and liability. On the contrary, the number of root certificates continues to grow, as does the number of organisations that control them. The liability of CAs is only given for EV certificates, and even then the sum for which a CA is liable is relatively low. On the bright side, the introduction of the Baseline Requirements and EV guidelines by the CA/Browser Forum has introduced common standards and defined several classes of certificates. Certificates for domain names may still be issued by relying on insecure mechanisms like email or DNS. However, certificates with name verification of the subject and EV certificates require CAs to carry out a much more thorough validation. Our conclusion here is that weaknesses in the X.509 PKI—mostly on an organisational, but also on a technical level—were known from early on, but never satisfactorily resolved. The proliferation of CAs, in particular, constitutes a persisting danger.

Research Objective O1.2

We documented incidents related to the X.509 PKI in a survey. To the best of our knowledge, this is the first academic survey to cover this topic. Our goal was to identify the root causes for the incidents. Table 3.1 provides a summary. We found that the most common causes of incidents were:

Poor operational practices Some CAs were tricked into issuing certificates for the wrong entity. Other made mistakes when issuing certificates and signed keys that were too short or accidentally issued intermediate certificates instead of server certificates.

Operational setups The weakest link argument we mentioned above holds. The high number of CAs, subordinate CAs and RAs was involved in several incidents where the mistake was not made by the primary CA, but by an organisation acting in its name. Control over the latter was not pronounced enough in some cases (e.g., Digicert Sdn. Bhd. or Comodo in 2011).

Vulnerabilities in the technology used The gravest incidents were all caused by compromises of CAs, subordinate CAs or RAs. In several cases, the intruders managed to issue rogue certificates. In at least one case, this resulted in a real man-in-the-middle attack. As a country may have been involved in the latter, it seems necessary to include an attacker of this strength in the threat model for X.509.

Furthermore, we found that incidents were often detected by third parties, not by the CAs themselves. Those parties had a legitimate interest in thwarting attacks.

Research Objective O1.3

Based on our findings from Research Objectives O1.1 and O1.2, we derived conclusions with respect to possible improvements to the X.509 PKI. Our primary conclusion is that the attack vectors that were exploited are hard or impossible to defend against *directly*. Consequently, the focus for improving X.509 should be on

Out-of-band mechanisms Several of the attacks were successful because no mechanism existed to verify the correctness of certificates other than the certificates themselves. Establishing further channels over which users can verify the authenticity of a certificate seems crucial to improve the security of X.509.

Strong mechanisms to detect incidents early Several of the attacks were detected very late, when the damage was already done. Furthermore, the attacks were often not disclosed by the CAs themselves. Early detection of ongoing attacks would allow to take measures to contain the damage.

Monitoring It is possible to detect problems in X.509 deployment by monitoring issued certificates (e.g., via active scans). This also allows to mount pressure on CAs to only issue certificates that comply with recognised security standards (e.g., the Baseline Requirements) and to improve their security practices in general.

Chapter 4.

4 Analysis of the X.509 PKI using active and passive measurements

This chapter is an extended version of our previous publication [37]. Please see the note at the end of the chapter for further information.

As shown in Chapter 2, the X.509 infrastructure consists of a rather large number of organisations and entities. These may be linked by contractual obligations, but their operations often show a high degree of independence. CAs, subordinate CAs and RAs are required to carry out their work in conformance with certain certification practices that they agree to follow. We showed in Chapter 3 that several incidents were caused by failure to follow such operational practices.

In this chapter, we take one step back and shift our focus away from incidents to the state of the X.509 PKI as a whole. Although the concrete implementation of work processes within a CA is not directly observable, it is possible to determine the outcomes of these processes: they are evident in the deployed X.509 PKI. Our overall research question is: can we assess the quality of this PKI, with the ultimate goal of determining where weaknesses occur due to poor operational practices?

To find an answer to this question, we carried out a thorough large-scale analysis of the currently deployed and practically used X.509 infrastructure for TLS and examined its security-related properties. We used two empirical methods. Data obtained from active scans allowed us to draw conclusions with respect to the X.509 PKI as it is deployed on servers. Data obtained from passive monitoring allowed us to draw conclusions with respect to the X.509 PKI as normal users encounter it during their Internet activities.

4.1. Investigated questions

The following are the research questions we investigated.

Fundamentals: secured connections We set ourselves the goal to determine how many servers offer TLS for HTTP, i.e., HTTPS. This can be achieved with active scans.

Security of deployment Certificates need to be deployed in a correct way in order for authentication and key agreement in TLS to succeed. Most importantly, certificate chains need to be correct and without errors, and the certificate's subject must indicate the host on which the certificate is used. Cryptographic material should not be deployed on many different physical machines as this increases its exposure and thus the attack surface. We investigated these deployment factors.

Certification structure The number of intermediate certificates and the number of certificates they sign are indicators of the PKI's certification structure. On the one hand, too many intermediate certificates increase the exposure of cryptographic material. On the other, use of at least one intermediate certificate is important as it allows to keep the private key for the root certificate safely offline. We investigated these factors in the deployed X.509 PKI.

Cryptography The cryptographic material used to establish TLS connections should be strong enough to provide the desired level of security. We decided to investigate the properties of public keys and hash algorithms in certificates as well as the symmetric ciphers used to secure the actual connections.

Issued certificates versus encountered certificates Active scans can only detect certificates as they have been issued and then deployed on servers. However, they cannot provide us with statistical information about certificates that users encounter during their Internet activities. We thus decided to investigate the latter case by using passive monitoring to extract certificates from TLS connections of users.

Development over time One should expect the security of the X.509 PKI to improve over time. We decided to investigate this by observing the state of the PKI over 1.5 years.

Influence of geographic factors It is well known that many services on the Internet are provided in a geographically distributed fashion—Content Distribution Networks (CDNs) are one example. We were interested whether the different locations would be configured in different ways, despite offering the same service. We thus decided to carry out additional scans from several geographic vantage points, distributed across the globe.

4.2. Measurements and data sets

We present our measurement methodology and the data sets we obtained in this section. We first describe the methodology for active scans, then for passive monitoring, before we give a description of the data sets themselves. For the purpose of comparison, we also included a third-party data set from the Electronic Frontier Foundation (EFF) in our analysis. Table 4.1 gives an overview of all data sets. The data sets that we acquired by active scans have been released to the scientific community at [249].

4.2.1. Methodology for active scans

We collected X.509 certificates with active scans between November 2009 and April 2011. We used primarily vantage points from Germany, first from University of Tübingen and later from Technische Universität München[1]. In April 2011, we also added several measurement points in other countries using PlanetLab [248] hosts. These countries were China, Russia, USA, Brazil, Australia, and Turkey.

We chose the hosts on the Alexa Top 1 Million Hosts list [133] as the hosts to scan. For each scan, we chose the most up-to-date list. The Alexa list contains the most popular hosts on the WWW as determined by the ranking algorithm of Alexa Internet, Inc. We chose this list as it contains sites that many users are likely to visit. Although its level of accuracy is disputed [84], it nevertheless is appropriate for our need to identify popular hosts. The list's format is somewhat inconsistent: usually, domains are listed without the `www.` prefix, but this is not always the case. We thus decided to emulate browser-like behaviour by expanding each entry into two hostnames: one with the prefix `www.`, and one without.

[1]This change became necessary due to our group's move to Munich.

Short Name	Location	Period	Type	Certificates (distinct)
Tue	Tübingen, DE	Nov 2009	Scan	834k (207k)
Tue	Tübingen, DE	Dec 2009	Scan	819k (206k)
Tue	Tübingen, DE	Jan 2010	Scan	817k (204k)
TUM	Tübingen, DE	Apr 2010	Scan	817k (208k)
TUM	Munich, DE	Sep 2010	Scan	829k (211k)
TUM	Munich, DE	Nov 2010	Scan	827k (213k)
TUM	Munich, DE	Apr 2011	Scan	830k (214k)
TUM	Munich, DE	Apr 2011	Scan, SNI	826k (212k)
Shanghai	Shanghai, CN	Apr 2011	Scan	799k (211k)
Beijing	Beijing, CN	Apr 2011	Scan	797k (211k)
Melbourne	Melbourne, AU	Apr 2011	Scan	834k (213k)
İzmir	İzmir, TR	Apr 2011	Scan	826k (212k)
São Paulo	São Paulo, BR	Apr 2011	Scan	833k (213k)
Moscow	Moscow, RU	Apr 2011	Scan	831k (213k)
S. Barbara	S. Barbara, USA	Apr 2011	Scan	834k (213k)
Boston	Boston, USA	Apr 2011	Scan	834k (213k)
MON1	Munich, DE	Sep 2010	Monitoring	183k (163k),
... grid certificates:				47% (52%)
MON2	Munich, DE	Apr 2011	Monitoring	989k (102k)
... grid certificates:				5.7%(24.4%)
EFF	EFF servers	Mar – Jun 2010	IPv4 scan	11.3M (5.5M)

Table 4.1. – Data sets used in our investigation of X.509.

Every TLS scan was preceded by `nmap` scans of TCP port 443 (three probes) to filter out hosts where this port was closed. We did not scan aggressively; the actual certificate scans thus started two weeks after obtaining the Alexa list. Our TLS scanning tool itself is a wrapper around OpenSSL [267]. It takes a list of hostnames as input and attempts to conduct a full TLS handshake on port 443 with each host. We let it run with 128 processes in parallel; thus a single scan took less than a week. Where a TLS connection could be established, we stored the full certificate chain as sent by the server, along with all data concerning TLS connection properties.

PlanetLab hosts received a slightly different treatment. We omitted the `nmap` scans entirely. The primary reason was that we wanted to scan TLS with the same host list as in our main location so we would be able to compare differences between locations. Furthermore, we wanted to scan from the remote location at roughly the same time as from our primary location. Thus, we imported the list of hosts with open port 443 from our main location in Germany and used it as a basis for the TLS scans.

4.2.2. Passive monitoring

As mentioned, we used data obtained from passive traffic measurement to complement our view on the deployed X.509 PKI with a view that corresponds to how users experience the X.509 PKI during their Internet activities. Our passive traffic measurements were carried out in the Munich Scientific Network (MWN) in Munich, Germany. We were able to monitor all traffic entering and leaving that network. The network is relatively large as it encompasses three major universities and several affiliated research institutions. The MWN provides Internet access to users of these institutions via a 10 Gbit/s link. At the time we carried out our measurements, the network served about 120,000 users and about 80,000 devices. The peak load that we measured on the link was about 2 Gbit/s inbound and 1 Gbit/s outbound traffic, observed in April 2011.

We carried out two monitoring runs, both during semester breaks. The setups we used were different, although the hardware in both setups remained the same (Intel Core i7, with hyper-threading; Intel 10 GE network interface with 82598EB chipset). The sampling algorithm was also the same in both runs: we sampled the first n bytes of each bi-flow. This method was described by Braun *et al.* in [12].

The difference between both runs lay in the processing. In the first run (September 2010), we used an approach that is similar to the one used by the *Time Machine* [43]. We dumped all sampled packets to disk into files of size 10 GB. As soon as one file was complete, we started an offline extraction process to obtain the certificates. We had to cope with hard drive limitations (both I/O and space). Thus, we sampled only the first 15 kB of each bi-flow.

For our second monitoring run (April 2011), we switched to an online analysis of TLS traffic. The method used was based on TNAPI [26], with an improvement presented by Braun *et al.* in [11]. With six instances of our monitoring application running in parallel, we could analyse up to 400 kB of traffic data for each bi-flow, with packet loss of less than 0.003%.

We used Bro [60] in both monitoring runs to process TLS. Thanks to its dynamic protocol detection, described in [19], we could identify TLS traffic without being limited to certain ports.

4.2.3. Data properties

Table 4.1 summarises the locations, dates and number of certificates in the data sets. Table 4.2 provides additional details for the monitoring runs. Our data sets can be divided into four groups.

The first group consists of the scans from hosts located in Germany at the University of Tübingen (Tue) and at TU München (TUM). These scans were carried out between November 2009 and April 2011, thus spanning a time interval of about 1.5 years. In April 2011, we performed an extra scan with Server Name Indication (SNI) enabled. SNI is a TLS extension to address virtual HTTP hosts. In such setups, a single Web server is responsible for a number of WWW domains. The name of the domain an HTTP client wishes to access is part of the HTTP headers it sends. Without SNI, a Web server is unable to determine which certificate it is supposed to send: the TLS handshake takes place before the client can inform the server via the HTTP header. SNI solves this by letting the client indicate the name of the domain as part of the TLS handshake. We carried out our scan with SNI enabled to determine if there were any significant differences in the certificates a server would send.

The second group of data sets was obtained in April 2011. We employed Planet-Lab [248] nodes from different countries, in order to obtain a geographically distributed picture of the TLS deployment. Our goal was to determine whether a client's location would result in different certificates received from a server. A typical example where we expected this to happen were CDNs. These often use DNS to route clients to different computing centres, depending on the geographic location of the client. Different computing centres might use different certificates, which would give us relevant data on the state and practice of certificate deployment in CDNs. Comparison between locations also has a desirable side effect: it can show whether certain locations actively interfere with TLS traffic by swapping end-host certificates during connection establishment (a man-in-the-middle attack).

The third group of data sets was obtained by passively monitoring live TLS traffic. There is an important difference between what can be learned from active scans versus passive monitoring. Active scans show which certificates are deployed for which host, and statistics like percentages of valid certificates thus refer to the state of the PKI as it is *deployed*. This might differ from what a user experiences: a user might access well-protected sites more frequently than poorly protected ones with invalid certificates. Evaluating the certificate chains we encounter in passive monitoring thus yields a picture of the state of the PKI as it is *encountered* by users.

We extracted all observed certificates in TLS traffic over each two-week period of a run. In September 2010, we observed more than 108 million TLS connection attempts, resulting in over 180,000 certificates, of which about 160,000 were distinct. Our second run observed more than 140 million TLS connection attempts, which were responsible for about 990,000 certificates, of which about 100,000 were distinct. We found that most TLS connections are due to HTTPS, IMAPS and POPS (see Table 4.2 for details). One of the difficulties we encountered was that the MWN is a research network that hosts several high-performance computing clusters. We consequently observed much grid-related traffic secured by TLS and found many grid certificates. They had short life-times and were replaced fast (their median validity period was just 11 hours). Grid-related certificates cannot be reasonably compared to certificates for WWW hosts. We thus filtered them out in our analysis of certificate properties. We explain this in the next section.

Finally, we included a data set from the EFF in our analysis. This data set was obtained using a different scanning strategy. The EFF data set is based on scanning the entire assigned IPv4 address space, which took a few months. According to the source code[2], it seems an attempt was made to accelerate the scanning by assuming a /8 subnetwork would not contain any Web servers if the first 655,360 probes did not hit one[3]. The method used by the EFF results in a higher number of observed certificates, but does not provide a mapping onto DNS names. Hence, the data set cannot be verified in terms of a matching subject for a hostname, since information about which domain was supposed to be served with a given certificate is not contained. In contrast, our own data sets provide this information. Since a sizeable part of IP addresses are assigned dynamically [78, 34], the long scanning period also impacts the accuracy of the mapping from IP addresses to certificates. The results from this data set must thus be taken with a grain of salt. However, due to its sheer size, the data set remains valuable for comparisons.

An important distinction that we sometimes make is whether we analysed certificates of the full set or just the *distinct* certificates in the data set. The difference is that the former refers to the deployment of certificates and thus duplicate certificates can occur, i.e., certificates that are used on more than one host. The latter is a pure view at certificates as they have been issued.

4.2.4. Data preprocessing

The International Grid Trust Federation [266] operates an X.509 PKI that is separate from the one for HTTPS and other TLS-related protocols. Their root certificates are not included in the Firefox root store, but are distributed in grid setups. Our setup

[2]`https://git.eff.org/public/observatory.git`; file `ControlScript.py`, lines 41–45, revision `ffc10e4e1876a855f0ce24f29ae0ac80d38a99dc`. There is a second scanner in later revisions that uses roughly 850,000 probes; file `scan/cs3.py`, lines 77–85, from revision `73403a74d23e604fa6b9ae918245f0ccaad62a2f`, added in 2012.

[3]A later IPv4-wide scan was carried out by Heninger *et al.* [35]. They found about twice as many certificates as the EFF.

Property	**MON1**	**MON2**
Connection attempts	108,890,868	140,615,428
TLS server IPs	196,813	351,562
TLS client IPs	950,142	1,397,930
Different server ports	28,662	30,866
Server ports ≤ 1024	91.26%	95.43%
HTTPS (port 443)	84.92%	89.80%
IMAPS and POPS (ports 993 and 995)	6.17%	5.50%

Table 4.2. – TLS connections in recorded traces.

stored certificates without a reference to the (potentially many) TLS connections during which it was observed. Although we cannot filter out grid traffic in our analysis of TLS connections to hosts, we were still able to identify some properties by correlating the encountered IP addresses with those of known scientific centres.

For our analysis of certificate properties (Section 4.4), we used a rather coarse filter mechanism: we simply checked whether or not a certificate contained the word 'grid' in the issuer field. We tested our certificate filter by checking the certificate chains in the data set containing the thus determined grid certificates. Indeed, 99.95% of them could not be found to chain to a CA in the Firefox root store, and not one of them had a valid certificate chain (as is to be expected due to the use of a different PKI setup). At the same time, the values for the validity of certificate chains of non-grid certificates was either in the same range as in our active scans (MON1) or even higher (MON2). We thus conclude that our filter removed most of the grid certificates and the remaining bias is tolerable.

We describe results from our analyses now, going item-wise through each question that we address.

4.3. Host analyses

Our first analyses are host-based: we investigate properties of the TLS hosts that we either contacted through active scanning, or whose connections we monitored passively.

4.3.1. Host replies with TLS

A central question was how many hosts actually support TLS and allow TLS connections on ports that are assigned to these protocols.

To begin, Figure 4.1 shows the results of our `nmap` probes for the scans in November 2009 and April 2011. It groups host from the expanded Alexa list by rank: the top 1000, the top 10,000, the top 100,000, and all hosts (top 1 million). We plot the cases of open ports vs. closed or otherwise non-reachable ports. We did not distinguish between unreachable ports due to network configurations at the destination systems and unreachable ports due to network failures in intermediate systems. We did count failures due to DNS resolution (no IP address for domain name) and `nmap` timeouts—these never occurred in more than 2-3% of cases. The most interesting case is, naturally, where a host showed an open TCP port 443, i.e., where at least one `nmap` run determined an open port. These were the candidate hosts we tried to connect to with OpenSSL later.

Figure 4.1 reveals the fraction of hosts with an open port 443 is higher over the entire range of the Alexa list than just for the top 1000. There is very little change

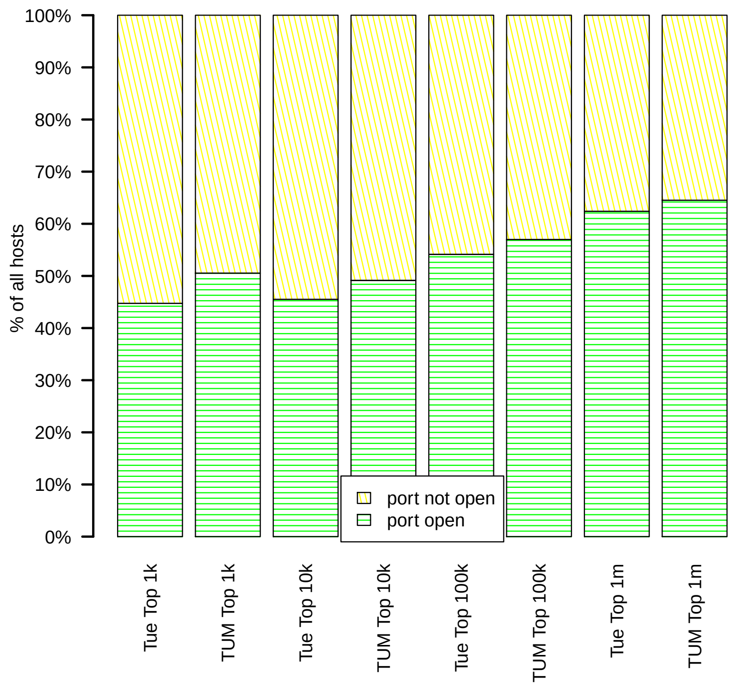


Figure 4.1. – Hosts with open port 443 vs. hosts without open port 443 for scans in November 2009 (Tue) and April 2011 (TUM), in relation to Alexa rank.

over time—we find a trend towards enabling TLS only among the higher-ranking hosts, but even there it is not pronounced too strongly. This seems surprising as one might assume that the more important sites have more reason to protect sensitive data than those on the lower ranks.

Without more data (and possibly more intrusive scanning), it is hard to find compelling reasons here. However, one might speculate that two factors might have an influence. First, it is possible that hosts on the lower ranks use out-of-the-box default configurations more often, or at least do not optimise their Web server configuration much as operators expect less traffic anyway. In these cases, TLS would be enabled by default. Hosts on the higher ranks, on the other hand, are more likely to use optimised configurations as they have to cope with different amounts of traffic—and they might disable TLS for performance reasons. If this is the case, however, the result is a rather poor deployment: only about half of the hosts among the top 1000 offer TLS (or at least have the corresponding port open).

For the remainder of this analysis, we focused only on the hosts that had reported an open port 443. We evaluated replies to our OpenSSL scanner. Figure 4.2 shows the results for these hosts. As can be seen, the numbers do not change significantly over time, neither for the overall picture nor for the top 1000. Two thirds of all queried hosts offer TLS on port 443, and more than 90% of the top 1000 do so, too. We found a surprising number of cases where OpenSSL reported an ‘unknown’ protocol on the HTTPS port. We therefore rescanned all hosts that had shown this result and analysed samples of the rescan traffic manually. In all samples, the servers in question offered plain HTTP on the TLS port. As can be seen, this unconventional configuration is less popular with highly ranked servers. The number of cases where a connection attempt failed (due to refused connections, handshake failures, etc.) also showed a correlation to rank, but was generally low.

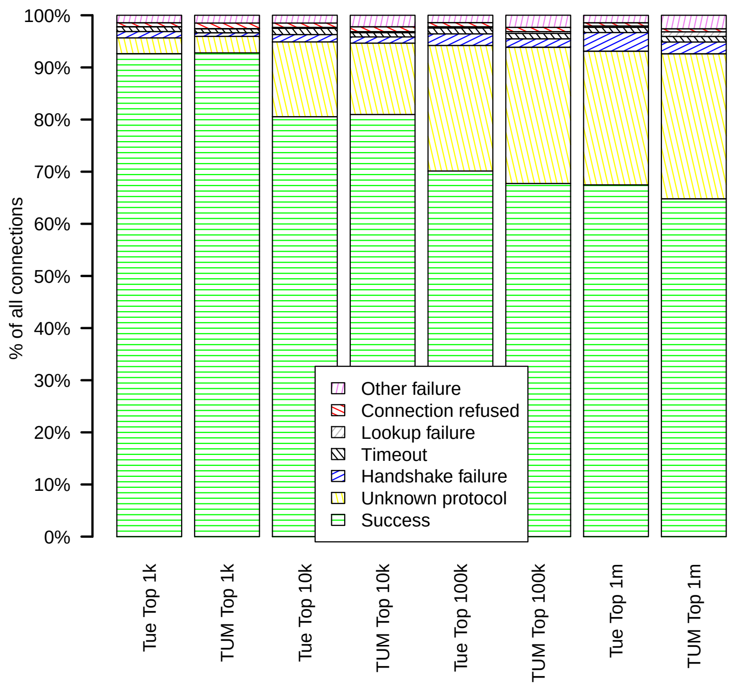


Figure 4.2. – TLS connection errors for scans in November 2009 (Tue) and April 2011 (TUM), in relation to Alexa rank.

On the whole, only about 800,000 hosts from the expanded Alexa list allowed successful TLS handshakes on the HTTPS port. Since 2010, several industry leaders like Google, Facebook, and Twitter have switched to TLS by default, thus bringing the importance of secure connections to the attention of a larger public. It seems, however, their example was not widely followed, at least during our observation period.

The corresponding data from our monitoring also showed a large number of TLS-enabled servers. Recall, however, that we captured Grid traffic, too, but could not filter it during our monitoring runs (live analysis of the certificates was not possible). Table 4.2 shows a summary of the monitored traffic's properties. We particularly point out the distribution of TLS connections to the well-known ports for HTTPS, IMAPS and POPS. These three protocols make up for most TLS traffic—however, there is a relatively large number of ports that are used in total. Also note the increase in observed TLS servers: this increase is also the main factor responsible for the increase in the observed certificate numbers in Table 4.1.

4.3.2. Negotiated ciphers and key lengths

The strength of cryptographic algorithms and the length of the involved symmetric keys determine the cryptographic protection of a TLS connection. We used our monitoring data to obtain statistics on ciphers and key lengths. We did not use data from active scans: it would have only limited validity as the negotiated ciphers depend on what a client supports. Hence, all results from active scans would be biased.

Figure 4.3 presents the most frequently negotiated cipher suites, key lengths, and hash algorithms encountered in the monitoring runs.

The first keyword in the cipher suite, e.g., RSA or DHE-RSA, refers to the method of key exchange. DHE indicates the Diffie-Hellman Ephemeral Key Exchange, the only mode that provides Perfect Forward Secrecy (PFS). PFS is an important security

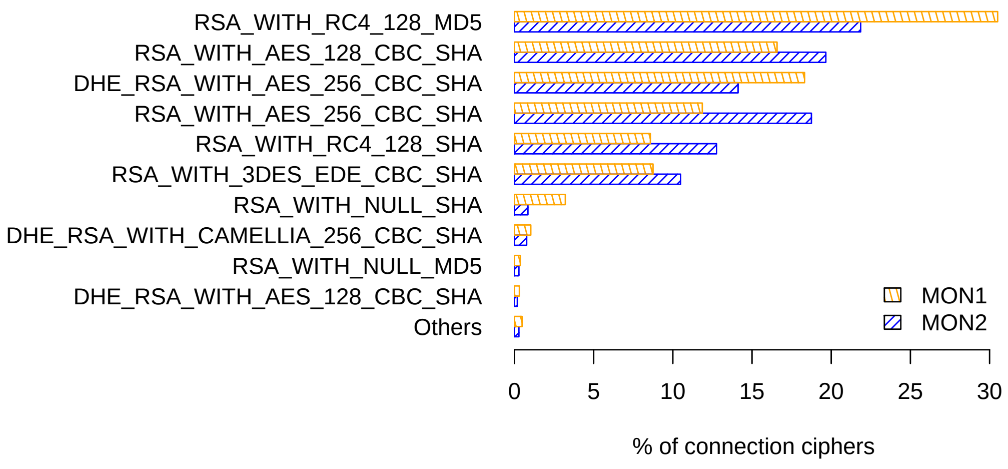


Figure 4.3. – Top 10 chosen ciphers in passive monitoring data.

property. An attacker who is able to record non-PFS-protected communication and later obtains the private keys used in the handshake (e. g., by compromise of a host) can decrypt the session key used in the TLS session and thus decrypt the entire conversation. PFS avoids this attacker by requiring that an (active) attacker must be able to break or compromise the private keys at the time of the handshake in order to be successful.

As we can see, only one cipher suite with PFS enjoyed significant popularity in our monitoring runs, namely DHE with RSA, together with AES, at a key length of 256 bits. The need for PFS may not have been as pronounced in 2011 as it is now—in the light of powerful attackers who can store entire encrypted sessions and who may have the power to obtain the necessary private keys, DHE is a wise choice.

We can also see that ciphers that were considered (very) strong at the time were most commonly selected: RC4 and AES in Cipherblock Chaining mode (CBC). These were sometimes also used with a very good safety margin (256-bit modes). 3DES was still used, but not in the majority of cases. MD5 occurred relatively frequently in Message Authentication Codes (MACs). In MAC schemes like HMAC, this is not considered problematic at this time, but it is not encouraged either [97]. In April 2011, we found an increased share of SHA-based digest algorithms, a positive development.

Lee *et al.* had analysed ciphers in TLS in 2007 [45]. Comparing our results to theirs, we found that while the two most popular algorithms remained AES and RC4, their order had shifted. In 2007, AES-256, RC4-128 and 3DES were the default algorithms chosen by servers, in that order. In our data, the order was RC4-128, AES-128 and AES-256. It is difficult to determine a compelling reason here. It could be that more clients enabled support for TLS, and with different ciphers supported; but it could also be that more servers supported TLS at the time of our monitoring and that their default choice was the very fast RC4 at lower key length.

Furthermore, we can see that some of the connections, albeit a minority, chose no encryption for their user traffic during the handshake. Such `NULL` ciphers were observed for 3.5% of all connections in MON1, and in about 1% of all connections in MON2. We were able to trace the corresponding IP addresses to computing centres. Our hypothesis is that TLS was only used for authentication and integrity of grid traffic, whereas encryption was omitted for performance reasons.

In summary, our results showed very good security as far as ciphers, key lengths and hash algorithms were concerned—at least at the time of our observations. In retrospect, we need to point out that CBC mode and RC4 in TLS have come under some pressure since our analysis. In summer-autumn 2011, a new attack on TLS was published: BEAST allows an attacker to derive HTTP cookies without actually breaking the encryption [244]. It works only in CBC modes and in TLS version 1.0 and the even older SSL 3. For a while, it seemed that using a stream cipher like RC4 was a good way to mitigate the attack. However, it was long known that this setup requires great care to be taken to avoid certain attack vectors [40]. In 2013, AlFardan *et al.* presented two attacks on RC4 in TLS that showed the security of the algorithm is far less than suggested by the keylength [4]. The attacks are not yet practical on a larger scale; however it is old wisdom in cryptography that attacks only become better over time. The authors suggested to move away from RC4 entirely. New scans and monitoring runs will need to take this into account and monitor the situation closely.

4.4. Certificate analyses

We now present results concerning properties of certificates and certification chains in X.509. For this analysis, we filtered out Grid-related certificates in the data from passive monitoring.

4.4.1. Certificate occurrences

Ideally, every WWW domain should have its own certificate for TLS connections. However, in practice it is common to use so-called virtual host setups to serve a number of domains from the same server. This method is based on an HTTP header, and it has a disadvantage: plain TLS does not indicate the domain that is to be accessed. This means that a Web server cannot determine the domain that a client wishes to access until after an encrypted connection has already been set up. At this time, the server must already have sent its certificate. There are two ways to deal with this problem. Either one uses the same certificate for all domains on a server, and adds all domain names to the certificate. This is not very elegant, but works with all setups. The second way is to use the SNI extension: this allows to send the required domain name as part of TLS. Since the SNI extension was only introduced several years after TLS [92], it is not universally deployed. It is thus quite common that a certificate is issued for several domain names. Note, however, that there are further reasons for a domain owner to buy just a single certificate with several hostnames included. One may be cost: this solution might be cheaper than separate certificates. It may also be less time-consuming for the operator as the configuration of the Web server is simpler. However, as the private key for every certificate must also be stored with the public key, this method can increase the attack surface if a private key needs to be stored on more than one physical machine.

We checked how often the same certificate was reused on several hosts. Figure 4.4 shows the complementary cumulative distribution function (CCDF) of how often the same certificate was returned for multiple different hostnames during the active scan of September 2010, in the middle of our observation period. We can see that the same certificate can be used by a large number of hosts (10,000 or more), although the probability for this is significantly less than 1:10,000. However, the probability that a certificate is reused rises quickly for a smaller number of reuses. We found it not uncommon (about 1% of the cases) that 10 hosts share the same certificate.

We investigated which hosts are particularly prone to reuse a certificate. To this end, we evaluated the domain names in the certificates' subject field. Despite being a violation of the specification in the Baseline Requirements [148] (also see Chapter 3, p. 34),

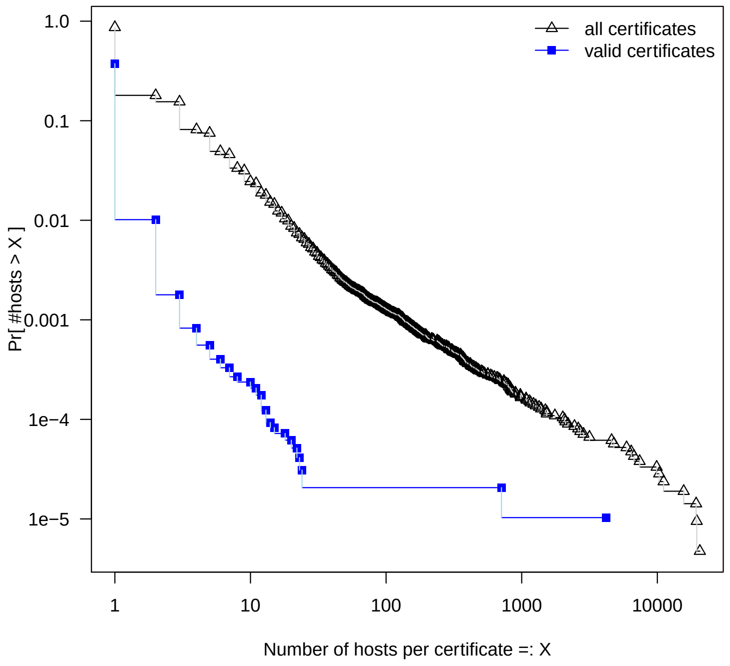


Figure 4.4. – Certificate occurrences: CCDF for number of hosts per distinct certificate.

this is the field that is practically certain to hold the intended hostname. Figure 4.5 shows our results for the certificates that we found most often. Most of these are identifiable as belonging to Web hosters—but only the certificates for `*.wordpress.com` were at least sometimes found to be valid (in 42.5% of cases). This is a rather poor finding, considering how popular some of these hosters are.

`www.snakeoil.dom` is a placeholder certificate for the Apache Web server, possibly appearing here because of Apache servers that run a default configuration (but probably are not intended to be accessed via HTTPS). We continued to investigate hostnames for those certificates that occurred at least 1000 times, and found that these seemed to be primarily Web hosting companies. We explore the issues of domain names in the subject fields and correctness of certificates in the next sections.

4.4.2. Correctness of certificate chains

To avoid confusion, we use the following terminology for the remainder of this work. A certificate is *verifiable* if it chains to a recognised root certificate in a root store, and the signatures in the chain are all correct. Furthermore, we require that no certificate in the chain must be expired or have a validity date ('not before' field) in the future, and intermediate and root certificates must have the CA flag set to true. When we speak of a *valid* certificate, we assume that an *additional* semantic check takes place: the certificate must have been issued for the correct hostname. Note that we do not yet address the issue of hostnames—we defer this to the next section.

There are a number of reasons why a certificate may not be verifiable due to errors in the chain. Hence, we verified chains with respect to the Firefox root store from the official repositories at the time of the scan or monitoring run[4]. Note that we determine

[4]These are currently called Mozilla Central. The current URL is `https://mxr.mozilla.org/mozilla-central/source/`.

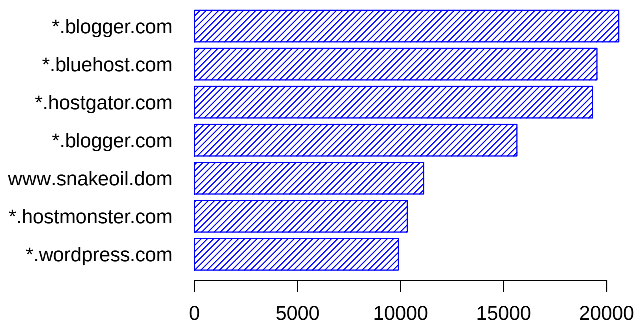


Figure 4.5. – Domain names appearing in subjects of certificates most frequently used on many hosts. The double entry for `*.blogger.com` is due to two different certificates with this subject.

verifiability with respect to a particular root store—the same certificate that is not verifiable due to a missing certificate in one root store may theoretically be verifiable with another root store. We chose the Firefox root store, however, as it is a publicly accessibly resource of a popular WWW client. We used OpenSSL's `verify` command to check certificate chains. Our check verified the full certificate chain according to OpenSSL's rules.

We examined which errors occurred how often in the verification of a certificate chain. Note that a single chain may contain multiple errors. Figure 4.6 presents our results for a selection of our data sets. The error codes are derived from OpenSSL error codes. Commonly, a browser would display a warning to the user (usually with the option to override the alert). Depending on the vendor, additional user clicks are necessary to determine the exact cause. In the following, we use error codes as employed by OpenSSL to indicate problems with the certificate chain.

Error code 10 indicates that the end-host certificate was expired at the time we obtained it. Expired certificates can be considered in two ways: either as completely untrustworthy or just as less trustworthy than certificates within their validity period. The latter would account for human error, i.e., failure to obtain a new certificate. We determined expiration in a post-processing step, which we show in Listing 1. This was necessary as we did not store timestamps during our monitoring runs, due to disk space limitations. Furthermore, some (rare) certificates had missing timestamp fields, and we wanted to process these in a lenient way that allowed for such errors in the certificates.

Error code 18 identifies an end-host certificate as self-signed. This means no certificate chain at all is used: a user has to trust the presented certificate and can only verify and validate it out-of-band (e.g., by checking hash values manually).

Error codes 19 and 20 indicate a broken certificate chain. Error code 19 means a correct full chain (with root certificate) was received, but the root certificate was not in the root store and thus untrusted. This error can occur, for example, if a Web site chooses to use a root certificate that is not included in a major browser. Certain organisations (like some universities) sometimes use root CAs of their own and expect their users to add these root certificates manually to their browsers. If this is not done securely and out-of-band, its value is very debatable—we have anecdotal experience of university Web sites asking their users to 'ignore the following security warning', which would then lead them to the 'secured' page. Error code 20 is similar to 19: there was

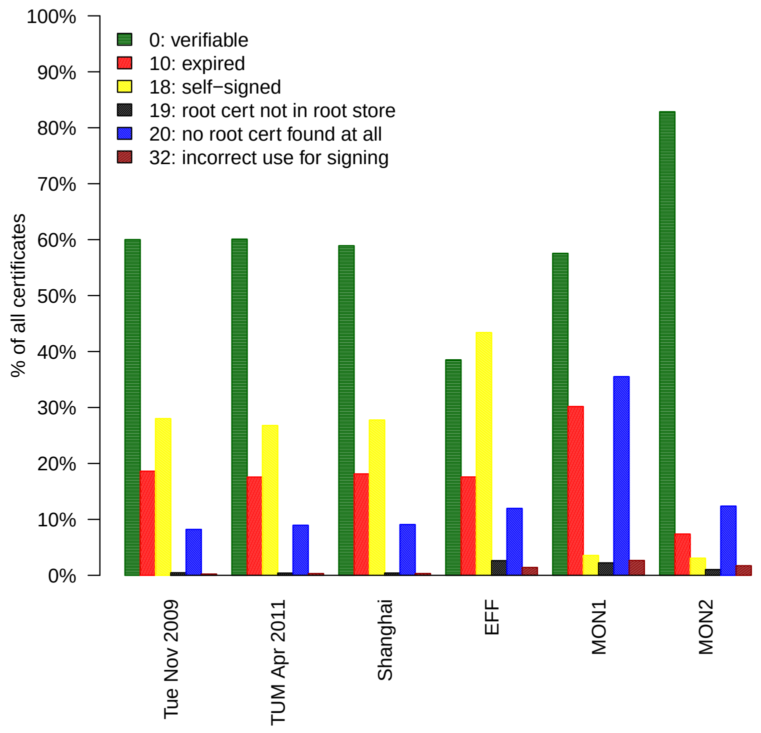


Figure 4.6. – Error codes in chain verification for various data sets. Multiple errors can cause the sums to add up to more than 100%.

a certificate in the sent chain for which no issuing certificate could be found, neither in the sent chain nor in the root store.

Error code 32 means that one or more certificates in the chain are marked as not to be used for issuing other certificates. Interestingly, cases where signatures were *wrong* were rare: just one certificate in November 2009, nine in the second monitoring run and 57 in the EFF data set.

Figure 4.6 reveals that trust chains were correct in about 60% of cases when considering the active scans and over all certificates. Expired certificates (about 18%) and self-signed certificates (about 27%) are by far the most frequent errors found. The number of verifiable certificates does not change significantly when considering only distinct certificates. Between November 2009 and April 2011, these numbers also remained practically constant. Note that the entries on the Alexa list had greatly differed by then; more than 550,000 hosts in April 2011 had not been on our expanded list in November 2009. We can thus say that even while the popularity of Web sites may change (and new sites likely enter the stage), the picture of the PKI with respect to verifiable certificates remains the same. This is a rather poor finding as no improvement seems to occur.

Error codes 19 and 32 occurred very rarely. For our scans from Germany, we can thus conclude that expired certificates and self-signed certificates are the major cause for chain verification failures. We compared our findings to the vantage points in Santa Barbara and Shanghai (Figure 4.6; only Shanghai data shown) and found the perspective from these vantage points was almost the same.

Figure 4.6 also shows the findings for MON1 and MON2. Here, the situation is somewhat different. Although the number of correct chains was similar in MON1 (57.55%), it was much higher in MON2 (82.86%). We can also see that the number

Listing 1 Algorithm to compute expiration of a certificate.

```
Algorithm Compute expiration
    Requires: Certificate fields not_before, not_after, time of TLS handshake
    Output: true when expired, else false
    Procedure expirycheck(not_before, not_after, cert_grab_time):
        if not_after = ∅:                                    ▷ Field empty
            return true                                      ▷ Count as expired
        else if cert_grab_time > not_after:                  ▷ Expired cert
            return true
        else if not_before = ∅:                              ▷ Field not set
            return false               ▷ Count as not expired: not_after is OK
        return (not_before > not_after)
```

of errors resulting from expired certificates has decreased, and so have the occurrences of Error 20. This does not mirror the findings from our scans. We cannot offer an absolutely compelling explanation for this phenomenon, but one plausible explanation is that the increased use of TLS by major Web sites (especially Google and social networks) contributed as these are extremely popular among users.

We also compared our results with the full data set of the EFF, which represents the global view for the IPv4 space. We found differences. First of all, the ratio of self-signed certificates is much higher in the EFF data set. This is not surprising, given that certification costs effort and often money—operators on not so high-ranking sites may opt for self-issued certificates or just use the default certificates of common Web servers like Apache. Samples showed us that these are not uncommon among self-signed certificates.

It should be noted that our observation period ended before incidents like the DigiNotar case (see Chapter 3) became known. Further scans might show whether there has been improvement in the quality of certificate chains since then. On the whole, however, we found the fact that about 40% of certificates in the top 1 million sites showed broken chains to be rather discouraging.

4.4.3. Correct hostnames in certificates

The second major factor in determining whether a certificate should be accepted is the correct identification of the certified entity. Only the application that uses a given TLS connection can know which subject to expect. The common practice on the WWW (but also for IMAPS and POPS) is that the application (i.e., the Web browser) verifies that the subject certified in the certificate matches the DNS name of the server. In X.509 terminology and syntax, the subject field must be a *Distinguished Name (DN)* . A *Common Name (CN)* is part of the DN. Very commonly, a DNS name is stored in the CN, although the *Subject Alternative Name (SAN)* field would be the correct place to put it. In our scans, however, the SAN was the less common practice. Technically, a user should also check manually that the other fields in the subject indicate the intended entity or organisation (e.g., the user's bank), but this is rarely done. Extended Validation (EV) certificates, which we discuss later, facilitate such assurances.

In our investigation, we checked if the CN attribute in the certificate subject matched the server's hostname. We also checked if the SAN matched. Where the CN or SAN fields were wild-carded, we interpreted them according to RFC 2818 [117], i.e., `*.domain.com` matches `a.domain.com` but not `a.b.domain.com`. One exception was to count a single `*` as not matching, in accordance with Firefox's behaviour. Note that this investigation can be carried out only for data sets from active scans of domains as neither data from monitoring nor the EFF data contain an indication of the requested hostname.

In the scan of April 2011 (no SNI), we found that the CNs in only 119,648 of the 829,707 certificates matched the hostname. When we allowed SANs, too, this number rose to 174,620. However, when we restricted our search to certificates that also had correct chains, the numbers are 101,070 (correct hostname in CN) and 149,656 (correct hostname in CN or in SAN). This corresponds to just 18.04% of all certificates. We checked whether the picture changed for the data set of April 2011 where we had SNI enabled. This was not the case. The number of certificates with both valid chains and correct hostnames remained at 149,205 (18.06%). We deduce from this that client-side lack of SNI support is not the problem. We also determined the numbers for the scans in November 2009, April 2010, and September 2010: they are 14.83%, 15.83%, and 16.84%, respectively. This indicates a weak but positive trend.

Our findings mean that only up to 18% of certificates can be counted as absolutely valid according to the rules implemented in popular browsers—in a scan of the top 1 million Web sites. More than 80% of the issued certificates would have led to a browser warning. In this light, we are not surprised it is often said that users choose to ignore security warnings. These are major shortcomings that need to be addressed. However, we also have to add a word of caution here: while a poor finding, it is not implausible that many of these hosts are actually not intended to be accessed via HTTPS and thus neglect this critical configuration. Normal users may therefore never encounter the misconfigured site, even in the case of very popular sites. Still, omitting support for TLS does not increase security, either.

4.4.4. Unusual hostnames in the Common Name

We encountered a few unusual hostnames. In the data set of April 2011, with SNI enabled, we found 60,201 cases of the string ‘plesk’ as a CN. Our hypothesis was that this is a standard certificate used by the Parallels/Plesk virtualisation and Web hosting environment. We tested this by rescanning the hosts, hashing the HTTP reply (HTML) and creating a histogram that counted how often which answer occurred. Just eight kinds of answers were alone responsible for 15,000 variants of a Plesk Panel site, stating ‘site/domain/host not configured’ or the like.

A further favourite of ours were certificates issued for ‘localhost’, which we found 38,784 times. Fortunately, neither certificates with ‘plesk’ nor ‘localhost’ were ever found to have valid chains.

4.4.5. Hostnames in self-signed certificates

Server operators may opt to issue a certificate to themselves. Hence, no external responsible Certification Authority exists. This saves the costs for certification, but requires users to accept the self-signed certificate and trust it. The value of self-signed certificates is debatable: some view them as useful in a Trust-On-First-Use security model as used with the SSH protocol; others view them as contrary to the goals of X.509. Our own view is that self-signed certificates can be useful for personally operated servers, or where it is safe to assume that a Man-in-the-middle attack in the first connection attempt is unlikely and a form of pinning can be used later (see Section 8.2).

In the data set with enabled SNI (April 2011), we checked if the self-signed certificates had a subject that matched the hostname. The result was sobering: 99.4% of CNs did not match. Subject Alternatives Names matched in 0.13% of cases.

Location	**EV status**
Tue Nov 2009	1.40%
TUM Sep 2010	2.10%
TUM Apr 2011	2.50%
Shanghai	2.56%
Santa Barbara	2.49%
Moscow	2.51%
İzmir	2.50%

Table 4.3. – Deployment of EV certificates over time and from Germany; and the same for April 2011 from Shanghai, Santa Barbara, Moscow and İzmir.

Interestingly, very few typical names account for more than 50% of the different CNs. The string 'plesk' occurred in 27.3% of certificates as the CN (without any domain). Together, we found either `localhost` or alternatively `localhost.localdomain` in 24.7% of CNs (without any further domain part). The remaining CNs in the top 10 all had a share of less than 3%. Yet the bulk of CNs is made up of entries that do not occur more than 1–4 times—i.e., names are assigned (possibly automatically), but they do not match the hostname. Our conclusion is that self-signed certificates are not maintained with respect to hostname. This may make them puzzling to the average user.

4.4.6. Extended Validation

Technically, a user should not only be able to verify that the domain in the CN or SAN matches, but also that other information in the certificate correctly identifies the entity on an organisational level, e.g., that it is really the bank he or she intended to connect to—and not a similar-looking phishing domain. This is the purpose of the so-called EV certificates, which were introduced several years ago. EV certificates are meant to be issued under the relatively extensive regulations described by the CA/Browser-Forum [147]. An identifier in the certificate identifies it as an EV certificate. A browser is meant to signal EV status to the user (e.g., via a green highlight in the address bar).

We analysed how often EV certificates occurred. Table 4.3 shows the results. One can see that there is a slight trend towards more EV certificates. We inspected the top 10,000 hosts in the scan of April 2011 (no SNI) more closely and found that the ratio of EV certificates was 8.93%. For the top 1000 hosts it was 8.11%, and for the top 100 8.33%. Surprisingly, for the top 50 it was 5.17%. We found two explanations for this. First, Google sites dominated the top 50 of our Alexa lists (almost half of all hosts), and Google does not use EV[5]. Second, a number of well-known Web sites (e.g., Amazon and eBay) use different hosts to let users log in. These are not in the top 50, but use EV.

To summarise, we found that EV certificates are not very wide-spread, even though they can be very useful for sensitive domains. Since they are commonly more expensive than standard certificates, this is somewhat to be expected.

4.4.7. Signature Algorithms

Cryptographic hash algorithms have come under increasing pressure in the past years, especially MD5 [69]. Even the stronger SHA1 algorithm is scheduled for phase-out [243].

[5]This is even true at the time of writing.

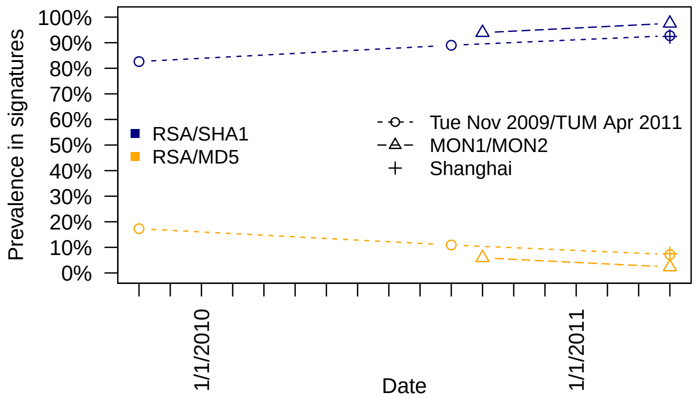


Figure 4.7. – Popular signature algorithms in certificates.

When researchers announced in 2008 they could build a rogue CA using MD5 collisions [260], a move away from MD5 was expected to begin.

We thus extracted the signature algorithms in certificates. Figure 4.7 shows the results for November 2009 and April 2011, the monitoring runs, and the vantage point from Shanghai. We omitted the rare cases (less than 20 occurrences altogether) where we found the algorithms GOST, MD2 and SHA512. The algorithms SHA256 and SHA1 with DSA were also only found rarely, in 0.1% of cases or less.

In our active scans, 17.3% of certificates were signed with a combination of MD5 and RSA in 2009. In 2011, this had decreased by 10% and SHA1 had risen by about the same percentage.

The view from the monitoring data was slightly different. Most importantly, MD5 usage numbers were lower, both for all certificates and only for distinct ones. Between September 2010 and April 2011, the number had fallen even further. Our conclusion here is that MD5 does not play an import role in this PKI any longer.

4.4.8. Public key properties

It is quite evident that the ciphers used in certificates should be strong, and keys should be of a suitable length to achieve the desired security. Otherwise the certificate might be crackable by an attacker: RSA with 768 bits was factored in 2009 [41]. With the security margin of RSA-1024 shrinking, a move towards longer ciphers has been recommended in 2011 [142].

We thus investigated the public keys in certificates. Concerning the ciphers, the result was very indicative. In the active scans of November 2009 and April 2011, which span 1.5 years, the percentage of RSA keys on all queried hosts was always around 99.98%. DSA keys made up the rest. Counting only distinct certificates, the percentages remained the same. The values for the monitoring runs were practically identical.

In these two scans, we also found a movement towards longer RSA key lengths: the percentage of keys with more than 1024 bits increased by more than 20% while the percentage of 1024-bit keys fell by about the same.

The general trend towards longer key lengths can be seen in Figure 4.8: the newer the data set, the further the CDF graph is shifted along the x axis. This shows that the share of longer key lengths increased while shorter key lengths became less popular, with the notable exception of 384-bit keys that were found in the crawls from 2010

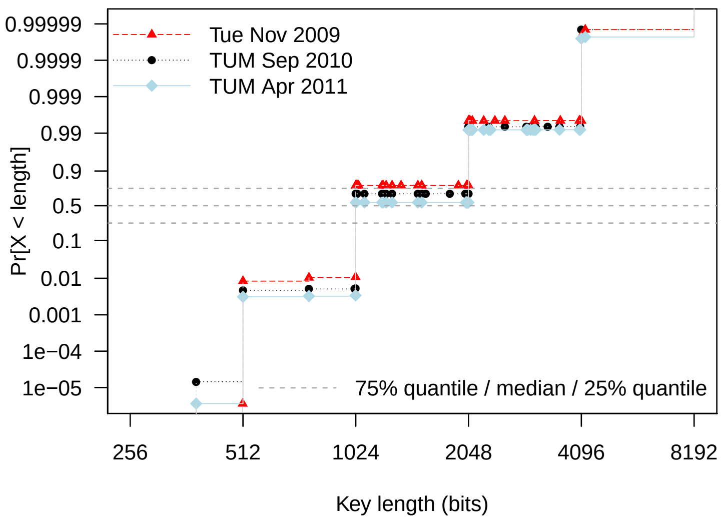


Figure 4.8. – Cumulative distribution of RSA key lengths. Note the unusual atanh scaling (double-ended pseudo-log) of the y axis.

and 2011, but not in the 2009 data. These were RSA keys, not keys for elliptic-curve algorithms, where such lengths might be expected. The small triangles/circles/lozenges along the curves indicate the jumps of the CDF curves; hence they reveal furthermore that there is a significant number of various non-canonical key lengths, i.e., key lengths that are neither a power of 2 (e.g., 2048) nor the sum of two powers of 2 (e.g., 768). However, their share is extremely small as the CDF lines do not exhibit any significant changes at these locations. It is not a particularly security-relevant finding[6], either.

An exponent and modulus make up the public key together; and there is only one private key for every public key. Concerning RSA exponents, the most frequent RSA exponent we found in November 2009 was 65,537, which accounts for 99.13% of all exponents used. The next one was 17, which accounts for 0.77% of exponents. The value 65,537 is the minimum value recommended by NIST [141], which is also fast to use in computations. It seems to be a preferred choice by software to create certificates.

There are two caveats to watch out for in public keys. The first refers to a now well-known bug of 2008: the Debian distribution of OpenSSL had removed a source of entropy and caused very weak randomness in key generation. It was therefore possible to precompute the affected public/private key combinations. Such keys must not be used as the private keys are publicly known. We determined the number of certificates with weak keys of this kind by comparing with the official blacklists that come with every Debian-based Linux distribution. Figure 4.9 shows the results for our scans. We can see the number of affected certificates become less over time. Interestingly, the percentage of Debian-weak keys was higher when we investigated it for just the distinct certificates. This means Debian-weak keys are more likely to occur on a single host, as opposed to being reused on several hosts. Our numbers fit well with another investigation of 2009 [79]—they essentially continue where that investigation left off.

Finally, the percentage of affected certificates was four times less in our monitoring data (about 0.1%)—this is a very good finding as it shows that the servers that most

[6]Keys are the products of two primes, e.g., 1024-bit keys are expected to be the product of two large primes whose length is on the order of 512 bits (there are a number of rules how to choose primes to make RSA secure). The product can, in rare cases, require a few bits less to encode.

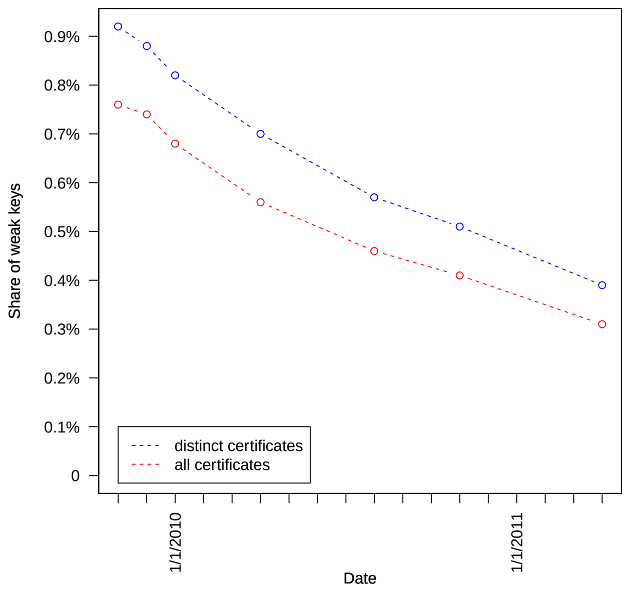


Figure 4.9. – Debian weak keys over the course of 1.5 years in the scans of the Alexa Top 1 Million.

users deal with are generally unlikely to be affected. However, we also found some weak yet absolutely valid certificates (correctly issued by CAs, correct hostnames, etc.) that were still in use, but the number was very small: such certificates were found on about 20 hosts in the last scan.

The second caveat is that no combination of exponent and modulus should ever occur in different certificates. However, we found 1504 distinct certificates in the scan of April 2011 that shared the public/private key pair of another, different certificate. In the scan of November 2009, we found 1544 such certificates. The OpenSSL bug could have been a cause for this, but we found that only 40 (November 2009) and 10 (April 2011) of the certificates fell into this category. We found one possible explanation when we looked up the DNS nameservers of the hosts in question. In about 70% of cases, these pointed to nameservers belonging to the same second-level domain. In about 28% of cases, these domains were `prostores.com` and `dnsmadeeasy.com`. A plausible hypothesis here would be that some Web space providers issue certificates and reuse the public/private keys—either intentionally or due to some flaw in a device or software. In either case, it means that some private key owners are theoretically able to read encrypted traffic of others. They can even use a scanner to find out which domain to target. However, our hypothesis does not explain the large number of remaining certificates with duplicate keys. We could not give a reason ourselves at the time we carried out the scans. Later, Heninger *et al.* found evidence that flaws in key generation could have been involved. We return to this in Section 4.5.

Our own conclusion here is that shorter key lengths were still too frequent, but the overall trend was very positive. RSA keys should not be issued any more at less than 2048 bits. The Debian OpenSSL vulnerability has become rare in certificates.

4.4.9. Validity periods

Certificates contain information for how long they are valid. This validity period is also a security factor. If a certificate is issued for a very long period, advances in cryptography (especially hash functions) can make it a target for attack. Alternatively,

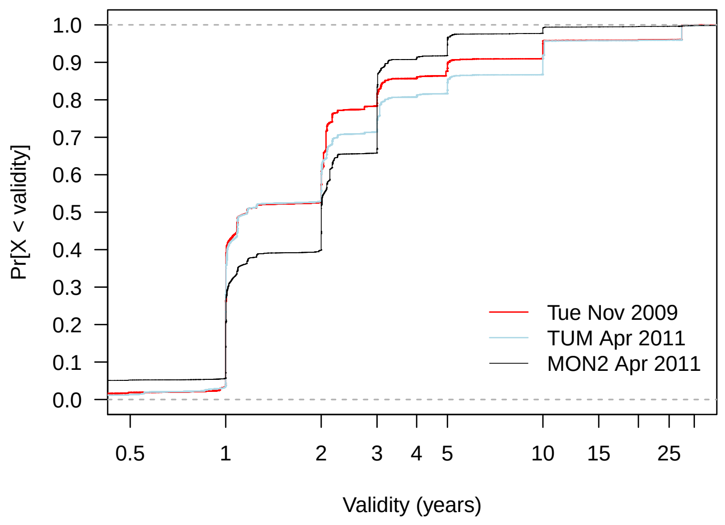


Figure 4.10. – Comparison of certificate validity periods, all certificates.

an attacker may attempt to reuse a stolen certificate (with stolen private key) and rely on the fact that certification revocation is not very effective in any case (see Section 2.7).

When we analysed the validity period for the certificates we encountered in our scans, we found that the majority of the certificates was issued with a life span of 12–15 months, i.e., one year plus a variable grace period. Other popular lifespans were two years, three years, five years, and ten years. This can be seen from the cumulative distribution function of the certificate life spans depicted in Figure 4.10. Comparing the data of November 2009 with the data of April 2011, we can see that the share of certificates with a validity period of more than two years increased, in particular the share of certificates issued for ten years. The curve for the second monitoring run reveals that the life spans typically encountered by users are either one, two, or three years, plus some grace period. In particular, certificates with life spans of more than five years seem to be only rarely used.

What the figure does not show are the extremal values for the certificate life span: we encountered life spans on a range from two hours up to an optimistic 8000 years. This was particularly evident in the monitoring data from April 2011. Figure 4.11 shows the same kind of plot, but this time for the distinct datasets. The number of certificates in the second monitoring run with extremely short validity periods (2 hours or less) is striking. We thus inspected these certificates manually: they all had random-looking issuers; and all issuer strings were unique with the exception of just six certificates. We found a plausible solution when we showed these certificates to a representative of the Tor project [269], who acknowledged the issuer strings as typical for certificates used in Tor circuits.

While there is a trend towards slightly longer periods rather than shorter ones (which is slightly worrisome), the otherwise reasonable validity periods seem to give little reason for concern.

4.4.10. Length of certificate chains

As explained in Chapter 2, intermediate certificates—kept within the same CA—allow to store the root certificate offline and carry out all online operations from the inter-

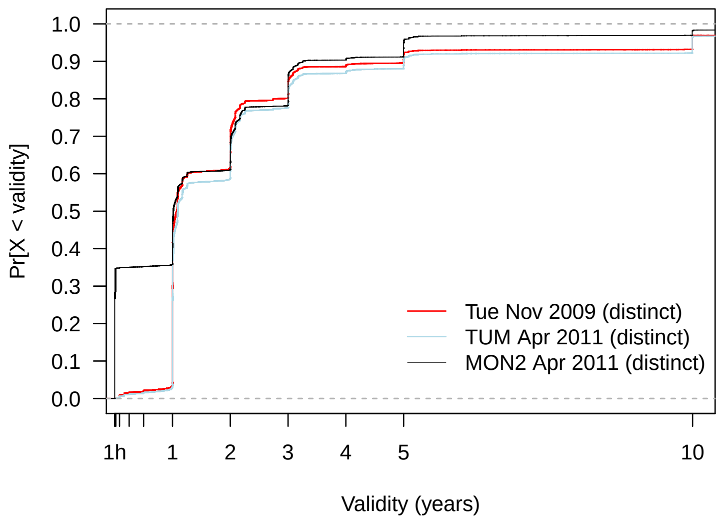


Figure 4.11. – Comparison of certificate validity periods, distinct certificates.

mediate certificates. At least one intermediate certificate per chain is thus a sensible setup. One should not use too many, however: it is not implausible that the probability for negligence increases with the number of intermediate certificates. This is even more problematic in setups where intermediate certificates are *not* kept within the same CA, but used for subordinate CAs outside the direct control of the root CA. Although not our focus, it is worthwhile to note that the verification of long chains has an impact on performance (especially if the verifying device is a hand-held computer).

We thus proceeded to investigate the length of certificate chains. We did this for the data sets from November 2009, April 2011 (no SNI), the EFF data set and the first monitoring run. We derive the length of a chain as the number of all intermediate certificates in the chain that are not self-signed. The rationale is that OpenSSL constructs chains by attempting to find a combination of certificates that yields a chain leading to a certificate in the root store. Consequently, self-signed intermediate certificates in the chain are either root certificates themselves, and we can safely discount them as not being intermediate certificates[7]. Or they are certificates 'outside' a chain, i.e., they cannot contribute to the verifiability of a certificate as they are not contained in a root store. In this case, they only impact performance, and we can discount them for our security-oriented investigation here.

There is a minor caveat here: it is possible in X.509 to construct more than one valid chain. One way to do this is to use the fields *Authority Key Identifier* (AKID) and *Subject Key Identifier* (SKID) from RFC 5280 [94], which provide (supposedly) unique identifiers for issuer and subject. The alternative is to use the fields for subject and issuer directly (these may sometimes not be unique). As a result, the corresponding intermediate certificates use the same public key, but chain to a different root certificate. We tested how often two chains may have been used in the mentioned datasets. We found the method via AKID and SKID was used in 0.06% of certificates in November 2009, and only about 0.01% further cases where the second method may have been used (we only checked matches on issuers and subjects, not public keys). The total number

[7]TLS allows to send root certificates as part of the chain, although there is no real advantage for clients, except where clients display authentication decisions for unknown root certificates to the user. Most users would probably not know how to make this decision.

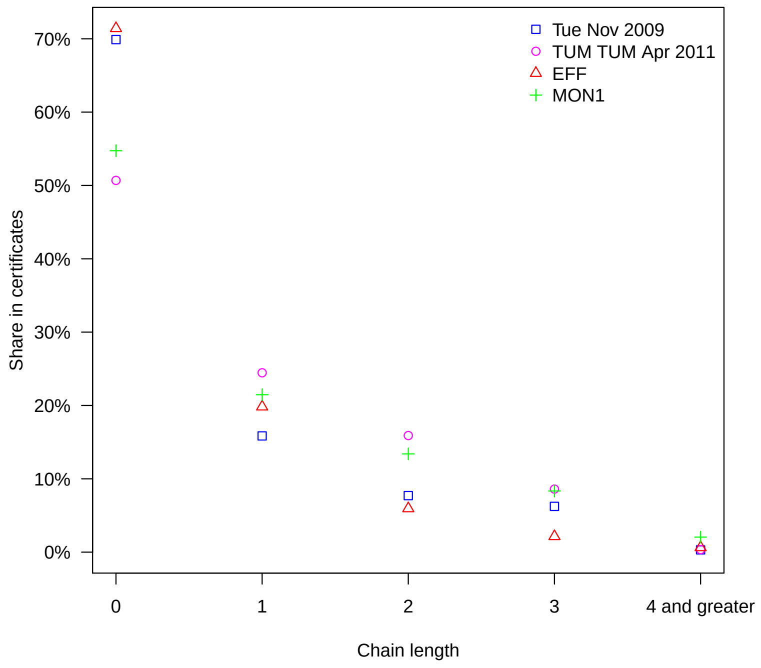


Figure 4.12. – Certificate chain lengths for two scans, one monitoring run, and the EFF data. If no intermediate certificates are sent, the chain length is 0.

across all scans was between 0.02% and 0.06%. The numbers for the monitoring run were similar. As they were so low, we decided to disregard the special case of multiple chains in our analysis.

Figure 4.12 shows the results. We see that the vast majority of certificates is verified via a chain of length three or less. At the other end, more than half of the certificates have a chain length of zero. The natural explanation here is the high number of self-signed certificates that we observed. When comparing the scan from November 2009 to the one in April 2011 (no SNI), we see that the share of chains of length zero greatly decreased while the share of chains with length one or more significantly increased by about 20%. The graph, as well as the increased average chain length (0.52 in November 2009 vs. 0.67 in April 2011), point to a weak tendency in the past 1.5 years to employ more intermediate certificates, not less. We double-checked this by plotting the chain lengths again, with self-signed end-host certificates excluded. The result can be seen in Figure 4.13. The overall numbers are smaller, as is to be expected. The shift towards using intermediate certificates is very pronounced when considering the changes from November 2009 to April 2011.

Interestingly, our results for the second monitoring run were entirely different. Chain lengths of 0–3 occurred at almost equal percentages, around 25–30%. We checked why this might be the case by analysing the subjects in the certificates. We found that they indicated high-profile sites in all cases, which we often knew to have enabled TLS since the first monitoring run, which were very popular among users, and which had appeared at different frequencies in our first run. The most common certificates were from Akamai and Dropbox (both chain length 0); Google services (chain length 1), Apple and Microsoft (chain length 2); and Rapidshare, Cloudfront, and Foursquare (chain length 3).

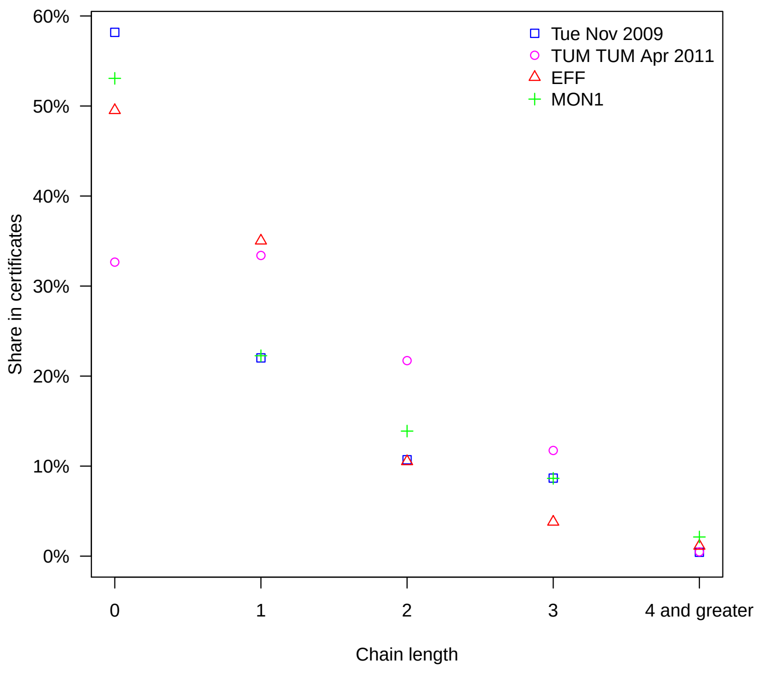


Figure 4.13. – Certificate chain lengths for the same data sets, with self-signed end-host certificates excluded.

Overall, we can say that certificate chains are generally remarkably short, and the move towards using intermediate certificates has not introduced chains that would be too long. Considering the trade-off of too many intermediate certificates versus the benefits of using them, this is a positive development.

The maximum length of chains found in the scans and EFF data set is 17. In the monitoring data, it is only 12. The scans thus detected hosts with very unusual chain lengths (outliers), whereas most certificate chains are actually relatively short. We briefly checked whether the high values of 17 down to 12 were cases of multiple chains; this was not the case.

4.4.11. Certification structure

While the use of intermediate certificates can be beneficial for security, it is also true that if too many intermediate certificates are used in a chain, the undesired result can be that the attack surface increases slightly as there are more targets. The number of distinct intermediate certificates versus the number of distinct certificate chains built with them is thus an indication whether a small number of intermediate certificates is used to sign a large number of certificates (good for security) or whether this is done with a larger number of intermediate certificates (bad for security).

We investigated the number of distinct intermediate certificates and the distinct certificate chains that are built with them. Figure 4.14 shows the development of the intermediate certificates. For the active scans from Germany, we see about 2300 distinct intermediate certificates in the data set, with a trend to increase. Compared to the size of the root store in Firefox, this means that, on average, every CA would use more than ten intermediate certificates. However, we already know that average chain lengths are short, so this points to a very skewed distribution. The number of distinct intermediate certificates in the EFF data set is even higher, almost grotesque: 124,387. The ratio

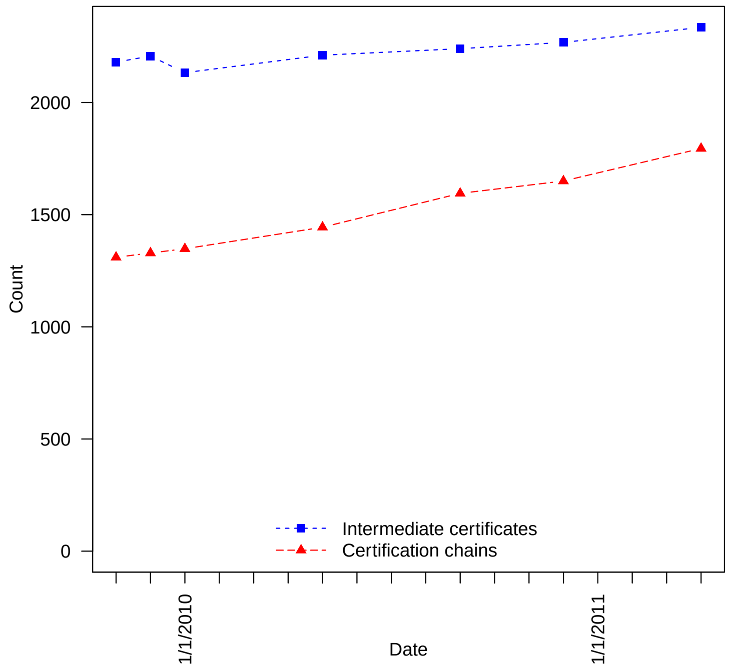


Figure 4.14. – Temporal development of the number of distinct intermediate certificates (squares) and the number of distinct certificate chains (triangles) across the scanning data sets.

of certificates/intermediate certificates for the top 1 million hosts is about 335 (scan of April 2011); in contrast, it is about 91 for the whole IPv4 space. This means that the top 1 million hosts use less intermediate certificates than hosts in the whole IPv4 space do.

To analyse chains, we computed a unique ID for every distinct certification chain we found. For this, we discounted the end-host certificate and any self-signed intermediate certificate in a chain as a potential root CA, as we did before. The remaining certificates were sorted, concatenated and hashed. Recall that the number of intermediate certificates is very small compared to the number of end-host certificates. Correspondingly, we found only a small number of certificate chains: about 1300 in November 2009 (compared to over 2000 intermediate certificates). This means that the X.509 certification 'tree' (end-hosts are leaves) shrinks rapidly from a wide base to very few paths leading to the root CAs. In the EFF's data set, we find an unexpected number of different chains: 17,392, which is much higher than the number in our scans. The ratio of distinct intermediate certificates to distinct certification chains in our own scans was between 0.6 (November 2009) and 0.76 (April 2011, no SNI). The numbers are plotted in Figure 4.14. For the EFF data set, it was 0.13. This indicates that the convergence to very few certification paths is much more expressed for the global IPv4 space than for the popular Alexa hosts.

Overall, our finding here is that too many intermediate certificates are encountered (leading to an unnecessary attack surface). The dimension of the problem is not huge, however. In the top 1 million hosts, the situation is better than in the IPv4 space as a whole.

Scan	Suspicious certificates	Differences to TUM, April 2011
Santa Barbara	1628	5477
São Paulo	1643	6851
Melbourne	1824	7087
İzmir	2069	7083
Boston	2405	5867
TUM, April 2011	3245	—
Shanghai	10,194	9670
Bejing	10,305	9901
Moscow	10,986	11,800

Table 4.4. – Occurrences of suspicious certificates per location, and number of certificates different to those seen from TUM.

4.4.12. Different certificates between locations

We investigated how many downloaded certificates were different between locations. Our motivation was twofold. A Web site operator may offer duplicated or different content depending on geographic region; CDNs are a large-scale example of this. In this case, it is possible that certificates are bought from different CAs. It is also possible that an operator switches CAs but fails to propagate the move to all sites in a timely fashion.

Another possible reason can be of a malicious nature: a router can intercept traffic and swap certificates transparently on the fly (man-in-the-middle attack). We labelled hosts as ‘suspicious’ when their certificate was identical from most locations and only differed in one to three locations. Table 4.4 shows the results from each vantage point.

Although the suspicious cases seem to occur particularly often when scanning from China and Russia, this might be simply the result of localised CDN traffic. We thus examined differences between the 2011 scans from Shanghai and Germany. The number of different certificates between these two locations was 9670, which is about 1% of all certificates in the data set for Germany. Only 213 of the corresponding sites were in the top 10,000; the highest rank was 116. We can surmise that if operators of high-ranking sites use CDNs for their hosting in these regions, then they deploy the same certificates correctly. From manual sampling, we could not find a man-in-the-middle interception.

We checked how many of the differing certificates from Shanghai were actually valid (correct chain, CN, etc.). This yielded just 521 cases—and only in 59 cases, the certificate was absolutely valid in the data set for Germany but not in the one for Shanghai. We checked the corresponding domains manually; not one of them could be identified as highly sensitive (e.g., politically relevant, popular Web mailers, anonymisation services, etc.). About a quarter of certificates were self-signed but different in the two locations. The reason for this is unknown to us.

While we are reluctant to offer compelling conclusions here, we do wish to state the following. First, there are not many differences between locations. High-ranked domains using CDNs seem to properly distribute their certificates. Maybe this is the most interesting finding, given the overall administrative laxness that is revealed by the many badly maintained certificates. Second, the number of different certificates was significantly higher from the vantage points in China and Russia. However, we found no indication of an attack. Third, we emphasise that we did not manually investigate all domains—the hope that other researchers would carry out their own investigations was one of our reasons to publish our data sets.

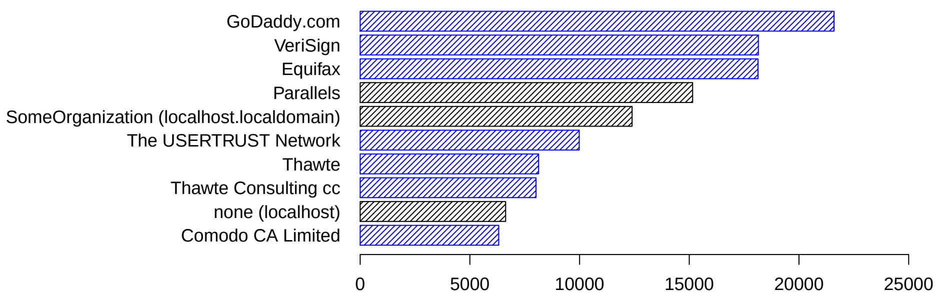


Figure 4.15. – Top 10 of issuers in data set of April 2011 (no SNI), distinct certificates.

4.4.13. Certificate issuers

We were interested to see which issuers occur most commonly. This is interesting for two purposes: first, it gives some insight into market share. Second, the respective CAs may be at higher risk of attacks as they may be perceived as more valuable (e.g., an attacker might hope his attack remains unnoticed for a longer time because the issuing CA for his rogue certificates has not changed, or he might want to steal the larger customer database).

We therefore determined the most common issuers for certificates in the data set from April 2011. We first investigated the case of the distinct certificates—this allows us to gain an insight into how many certificates were issued by different entities. We found 30,487 issuer strings in total. However, many different issuer strings may actually represent issuing entities that belong to the same organisation. As the subordinate CAs or even intermediate certificates used by CAs are generally not known (root stores do not keep track of them), we used the following method to obtain an *approximation* of the top 10 most common issuers. We extracted the 'Organisation' part of the issuer field—in most cases, this will already identify the organisation that runs the issuing entity. We then grouped by this field and counted all occurrences in end-host certificates. Figure 4.15 shows the result.

We find GoDaddy as the top issuer, followed by VeriSign and Equifax. At the time of writing, the latter two are actually both owned by Symantec. We find a high number of issuers identifying themselves as 'Parallels'—this is a virtualisation software by Plesk. Two issuers, 'SomeOrganization' and 'none', are found in self-signed certificates and are likely a form of default certificates as their CNs (in the issuer) always indicated 'localhost' and 'localhost.localdomain', respectively. The USERTRUST Network is a CA that is part of Comodo.

We then proceeded to investigate the issuers over all certificates, i.e., allowing for certificates reused on hosts. The picture was somewhat different. Figure 4.16 presents this top 10. The USERTRUST Network was suddenly at the top, with GoDaddy second, and Comodo ranking much higher. This means certificates from these CAs are reused on several hosts much more often. We also find GeoTrust, another CA owned by Symantec, and Google, who run a subordinate CA certified by Equifax. The high number of Google certificates is likely due to the use of Google certificates on Google's many services.

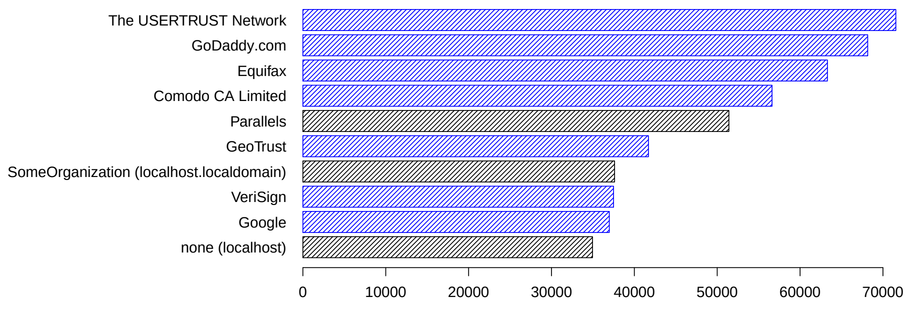


Figure 4.16. – Top 10 issuers in data set of April 2011 (no SNI), all certificates.

4.4.14. Further parameters

We inspected the serial numbers in valid certificates and looked for duplicates by the same issuing CA. We did not find any in the last three scans. This is a good finding as a certificate's serial number is intended to be a unique identifier (recall that blacklisting of certificates in revocation lists is done via the serial numbers).

We also investigated a certificate property that is of less relevance for security, but interesting nonetheless: X.509 version 1 has been outdated for years. The current version is X.509 version 3. We investigated the data sets of November 2009 and April 2011 in this regard. In November 2009, 86.01% of certificates were version 3, 13.99% were version 1. In April 2011, 85.72% were version 3 and 14.27% version 1.

Although our first guess was that this was an artefact of measurement, we found that 33,000 certificates with version 1 had not been seen in any previous scan. None of them had valid certificate chains, however, and 31,000 of them were self-signed. Of the others, the biggest issuer was a Russia-based company. We investigated all issuers that had issued more than two certificates and found that all of them seemed to employ a PKI of their own, but without root certificates in Firefox. The reasons for this use of version 1 are unknown to us, but plausible causes are software default settings or an arbitrary choice to use the simpler format of version 1. Version 1 lacks some extensions that are useful to limit the ways in which a certificate can be used, however, so this is not a wise choice.

4.4.15. Certificate quality

We conclude our analysis with a summarising view of certificate quality. Figure 4.17 shows a classification of certificates in three categories, which we termed 'good', 'acceptable' and 'poor'. *Good* certificates have correct chains, correct hostnames, exactly one intermediate certificate[8], do not use MD5 as a signature algorithm, use non-Debian-weak keys of at least 1024 bits, and have a validity of at most 396 days (a year plus a grace period). For *acceptable* keys, we require the same but allow two intermediate certificates, and validity is allowed to be up to 25 months (761 days). *Poor* keys represent the remainder of keys (with correct chains and hostnames, but otherwise failing to meet our criteria). Figure 4.17 shows certificate quality for the scan in November 2009 and for the scan in April 2011 (no SNI).

First of all, the figure reveals that the share of valid certificates (total height of the bars) is negatively correlated with the Alexa rank. This does not come as a surprise, since operators of high-profile sites with a higher Alexa rank can be expected to invest

[8]As before, we do not filter out sites that send more than one chain.

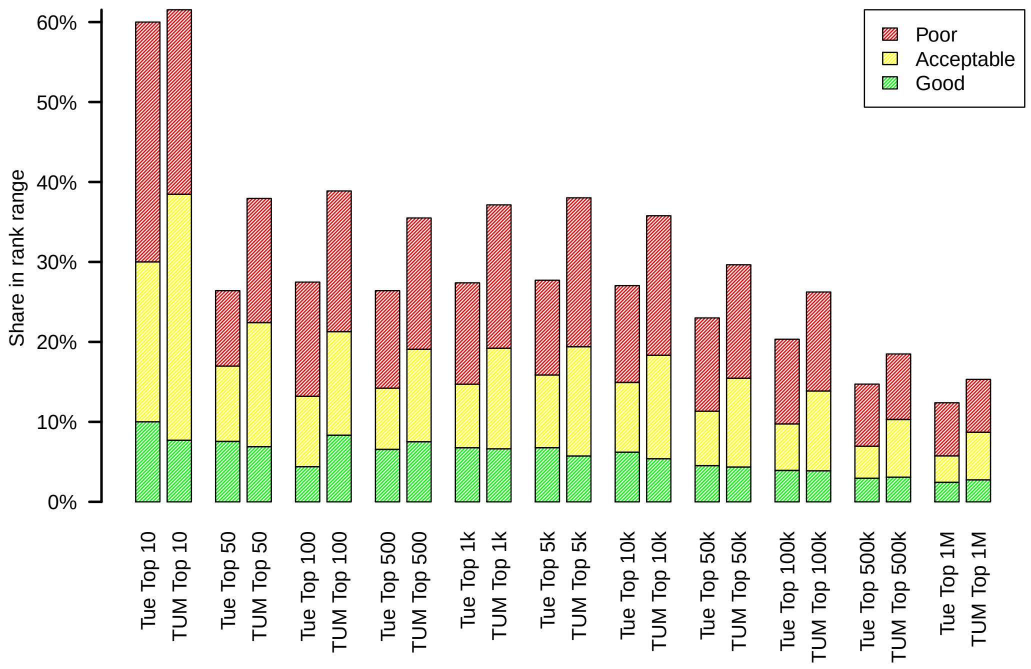


Figure 4.17. – Certificate quality in relation to Alexa rank for the data sets of November 2009 (Tue) and April 2011 (TUM). Note that the figure shows only *valid* certificates, and thus the numbers do not add up to 100%.

more resources into a working HTTPS infrastructure. What is surprising, however, is that even in the top 500 or 1000—i. e., truly high-ranking sites—only just above 40% of certificates are absolutely valid. Only the top 10 sites seem to be an exception here. Interestingly, although the more high-profile sites are more likely to deliver valid certificates, the share of poor certificates among their valid certificates is higher when compared to the entire range of the top 1 million sites.

Concerning development over time, we find interesting trends. Overall, the fraction of sites with valid certificates increased over our observation period. While the difference in the top 10 is marginal, we see a consistent development over the entire range. The relative fractions (good versus acceptable versus poor) do not seem to change much; only the fraction of good-quality certificates shrank slightly. A possible explanation may lie in our criterion for the number of intermediate certificates in chains: we know that this number increased over time. We do not view this as a critical finding: compared to other flaws in certification, the impact of using one versus two intermediate certificates is marginal.

4.5. Related work and aftermath

Since our analysis of the X.509 PKI covers the years 2009–2011, we group our discussion of related work into two groups: publications before our own, and publications after ours that, in part, built upon it.

4.5.1. Previous work

We are aware of two previous contributions on certificate analysis for TLS. Both were given as talks at hacking symposia, but were not published as articles. Between April and July 2010, members of the EFF and iSEC Partners conducted what they claimed to be a full scan of the IPv4 space on port 443 and downloaded the X.509 certificates. Initial results were presented at DEF CON 2010 [184] and 27C3 [245]. The authors focused on determining the certification structure, i.e., number and role of CAs, and several noteworthy certificate properties like strange subjects (e.g., `localhost`) or issuers.

Ristić conducted a similar scan like the EFF in July 2010 and presented some results in talks at BlackHat 2010 [251] and again (now including the EFF data) at InfoSec 2011 [202]. The initial scan was conducted on 119 million domain names, and additionally on the hosts on the Alexa Top 1 Million list [133]. Ristić arrived at about 870,000 servers to assess, although the exact methodology cannot be derived from [251, 202]. However, the information about certificates and ciphers collected is the same as in our scans, and together with the EFF data set our data sets provide a more complete coverage.

Vratonjic *et al.* presented a shorter study of a one-time scan of the Alexa Top 1 Million list [75], which was published while our own contribution was under submission. Their results confirm ours.

Lee *et al.* also conducted a scan of TLS servers [45]. In contrast to our work, they did not investigate certificates but focused on properties of TLS connections (symmetric ciphers, MACs, etc.) and the cryptographic mechanisms supported by servers. The number of investigated servers was much lower (20,000).

Yilek *et al.* investigated the consequences of the Debian OpenSSL bug of 2008 [79]. The authors traced the effect of the error over a time of about 200 days and scanned about 50,000 hosts.

The problem with the above scans, particularly of the IPv4 space, is that more hosts are included that are likely not intended to be accessed with TLS and thus provide invalid (and often default) certificates. The percentages given in [184, 245, 251, 202] thus need to be treated with caution. This is particularly true for the scan by the EFF as this scan covered IP ranges used for dynamic IP address allocation. Combined with the long duration of the scan, this leads to an inaccuracy. Our actively obtained data sets concentrate on high-ranked domains from the Alexa Top 1 Million list, and observe these domains over a long time period. Note that high-ranked domains can be assumed to be aimed more at use with TLS. This should at least be true for the top 1000 or top 10,000.

Our monitoring does not suffer significantly from the problems mentioned above, either. Thanks to it, we were not only able to estimate the deployment of the TLS infrastructure, but were also able to analyse which parts of the PKI are actively used and therefore seen by users. Furthermore, our view on the TLS deployment is not a single snapshot at an arbitrary time, but includes changes that operators have conducted in 1.5 years. By analysing TLS data that has been obtained from all over the world, we could also estimate how users see the TLS-secured infrastructure in other parts of the world.

4.5.2. Later work

A number of publications carried out investigations that were similar to ours and, in some cases, built upon it.

Lenstra *et al.* published an analysis of the strength of DSA and RSA keys in 2012 [46]. They built a data set of public keys from three sources: their own collection of public keys, the EFF data set and our data sets. They investigated the properties of the keys. One of their primary findings was that more RSA than DSA keys showed weaknesses. Their conclusion was that generating RSA keys carries significantly higher risk than generating DSA keys.

A similar analysis was carried out by Heninger *et al.* and published very shortly afterwards [35]. In contrast to the work by Lenstra *et al.*, Heninger *et al.* created their database of keys from their own IPv4-wide scans. They confirmed the finding of a higher proportion of weak RSA keys, but came to an entirely different conclusion. Thanks to their active probing, they could show that a number of weak keys are the result of devices with poor entropy during key generation—they even noted that we had found such keys in our own analysis in the form of 'duplicate' keys in different certificates. The authors could also show that devices with poor entropy are at extreme risk of revealing their private key when using DSA: here, the entropy must be high enough every time the algorithm is used for signing, not just on key generation. The conclusion by Heninger *et al.* was thus that device properties were responsible for the weak keys, and that DSA is in fact the more dangerous algorithm to use. The team also found duplicate keys that were caused by other issues—we return to this in our own discussion of SSH public keys in Chapter 6.

Durumeric *et al.* presented their tool to scan the entire IPv4 space at line speed in [21]. Among other things, they also presented early results on HTTPS adoption over one year and found an increase of almost 20%. Durumeric *et al.* followed this up with a larger study in [20]. The focus of the latter study was on the certification structure as created by CAs and subordinate CAs, and to a lesser degree on the properties of end-host certificates as we investigated them. Among other things, the authors investigated the number of trusted CAs and found a large number of intermediate certificates, on the same order as we had found in the EFF data set. Disturbingly, they find that a very large number of certificate chains contain the same intermediate certificate. This would mean a compromise of this certificate would cause a major key rollover on the Internet. The authors draw a picture of the distribution of subordinate CAs and find many non-commercial entities. They also found unexpected security-relevant issues, like locally scoped names in CNs (e.g., `localhost`), which were signed by CAs (recall we did not find any of these for the Alexa Top 1 Million list of domains).

Asghari *et al.* presented a study of HTTPS from a security-economical point of view in [7]. Based on lessons learned from CA compromises, they also conclude that the weakest CA is the 'weakest link' in the HTTPS authentication model. Their primary contribution is an in-depth analysis of CA market share, with a surprising finding: the more expensive CAs hold a significantly larger market share than the cheaper CAs. They conclude that the market is not primarily driven by price. Using a qualitative approach (interviews with buyers), they conclude that buyers are aware of the weakest link argument, but continue to buy from more expensive vendors. One of the reasons is that large CAs are considered too large to be removed from browser root stores—i.e., buyers do not have to fear that their sites are suddenly without HTTPS access in case of compromise. Another reason is that larger CAs often offer accompanying services like extended support.

Akhawe *et al.* provided a study [1] in which they used passive monitoring to obtain a large number of certificate chains, based on measurements in networks with a total of over 300,000 users and over nine months. One of their contributions was a better understanding of how browser libraries make authentication decisions, which was so far very poorly documented—among other things, they found that the NSS library as used by Firefox is more lenient in certificate verification than OpenSSL. Where OpenSSL accepted 88% of chains, NSS would accept 96%. This means that Firefox users are slightly less likely to see warnings than previous studies, including ours, suggested. The authors also measured the frequency of TLS errors in their study and gave recommendations on how to design the authentication decision in browsers better and with less false-positive warnings for users.

Amann *et al.*, finally, presented an analysis of the trust graph in the HTTPS ecosystem [6], together with an analysis of known man-in-the-middle attacks, where the certificate chain of the rogue certificate was different. Their goal here was to determine whether malicious certificates are well detectable, e. g., by noting the sudden changes in the certificate chain, for example due to the change of the root CA or an intermediate certificate. They found that this is not the case as too many such changes exist and are of a routine nature rather than an attack. The authors discussed the implications for one of the concepts to improve X.509 security, namely Certificate Transparency (CT). We return to this aspect in Chapter 8.

4.6. Summarising view

By combining and evaluating several actively and passively obtained data sets, which were in part also obtained over 1.5 years, we were able to derive a very comprehensive picture of the X.509 infrastructure as used with TLS. Our analysis supports with hard facts what has long been believed: that the X.509 certification infrastructure is, in great part, in a sorry state.

The most sobering result for us was the percentage of certificates that a client using the Mozilla root store would accept without a warning: just 18%. This can be traced to both incorrect certificate chains (40% of all certificates exhibit this problem), but even more so to incorrect or missing hostnames in the subject or SAN. With self-signed certificates, where conscientious operators would have an incentive to use the correct hostname, the situation was much worse. The only positive point is that the percentages of absolutely valid certificates increased since 2009, but then again only very slightly. Recall that these numbers refer to the top 1 million hosts—the percentage of certificates where the chains are correct was lower for the full IPv4 space than for the top sites as we found by examining the EFF data set.

Moreover, many certificate chains showed more than one error. Expired certificates were common, and so were certificates for which no root certificate could be found. A further problematic finding is that all our data sets revealed a high number of certificates that were shared between a large number of hosts. This was even the case for high-profile Web hosters—and often, the hostnames did not match the certificates there, either. Although offered by several CAs, EV certificates do no seem to be in wide use.

This truly seems a sorry state. It does not come as a surprise that users are said to just click away warnings, thus adding to the general insecurity of the WWW. As few CAs are responsible for more than half of the distinct certificates, one should think the situation should be better or at least easier to clean up.

There are some positive tendencies that should be mentioned, however. Our evaluation shows that the more popular a given site is, the more likely it supports TLS, and the more likely it shows an absolutely valid certificate. On the whole, key lengths seem not to constitute a major problem. The same is true for signature algorithms. Keys with short bit lengths are becoming fewer, and the weak MD5 algorithm is clearly being phased out. Over the 1.5 years, we also found an increase in the use of intermediate certificates while chain lengths remained remarkably short. This is a good development as end-host certificates should not be issued by a root certificate that is used in online operations.

Concerning our passive monitoring, the data we obtained allowed us to evaluate negotiated properties of TLS connections, which cannot be obtained with active scans. We were able to determine the negotiated ciphers and digest mechanisms. At the time, most connections used ciphers considered secure, at acceptable key lengths, with a good security margin. However, recent developments have put pressure on some of the used algorithms, and it will be interesting to carry out a similar monitoring run to determine whether TLS implementations choose different ciphers now.

With the above mentioned problems in certificates, however, we have to conclude that the positive movements do not address the most pressing problems, which are the certification structure and deployment practices.

4.7. Key contributions of this chapter

In this chapter, we addressed Research Objective O2.1. We analysed the properties of the X.509 PKI using active and passive measurements. Our general finding was that the X.509 PKI is not well deployed. Our key contributions were as follows.

Long-term distributed scans We scanned the servers on the Alexa Top 1 Million list over the course of 1.5 years, between November 2009 and April 2011. In addition, we scanned the servers from eight further vantage points across the globe in April 2011. This allowed us insights into the X.509 PKI as it is deployed on HTTPS servers. At that time, ours were the largest and longest scans of this kind.

Passive monitoring We extended our data sets with certificates won from monitoring TLS connections in the Munich Scientific Network (MWN). This yielded insights into the X.509 PKI as it is encountered by users accessing sites according to their browsing habits. It also allowed us to determine the symmetric ciphers used in TLS connections.

Third-party data set As a reference, we also analysed data from a third party, namely the EFF, who had carried out a three-month long scan of the IPv4 space and collected roughly 11 million certificates. This data set allowed us to compare the X.509 PKI as used for the most popular Web sites with the PKI as a whole. However, due to the nature of the EFF's scan, the results for their data set must be treated with some care (no domain information, loss of accuracy due to IP address reassignments).

Secured connections We found a clear correlation between a site's rank on the Alexa list and the probability it would offer TLS-secured access. Interestingly, the percentage of sites that have an open HTTPS port is higher for the top 1 million than for the top 1000 or even top 10,000 sites. The explanation we can offer are default configurations for hosts on the lower ranks of the list and conscious decisions to disable TLS for the higher-ranking sites, possibly for performance reasons. However, when connecting to the sites with an open HTTPS port, we found that only about 60% offered HTTPS, although this percentage was much higher for the 1000 most popular sites (over 90%) and even the top 10,000 most popular sites (roughly 80%). While a good finding for the top sites, it is a rather poor result as a whole.

Security of deployment The majority of certificates chained to a root certificate in the Mozilla Firefox's root store and was not expired nor showed other errors in the chain. However, almost 20% of certificates were expired, showing insufficient deployment practice. We found that only 18% of certificates were verifiable and issued for the correct hostname. Worse, the trend showed only slight improvements over time. This result is very disappointing and shows little care is applied in deploying certificates. Almost 30% of certificates in our data sets were self-signed (and more than 40% in the EFF's data set). We could show that the majority of them were issued without regard to hostnames: less than 1% were issued for the right hostname. Although we found some extreme cases, validity periods for certificates were mostly sufficiently short and no reason for concern. We also found that many certificates are reused on many domains. Where domains are hosted on different machines, this is a security weakness as it increases the attack surface. This is one area where improvements are needed.

Certification structure We could show that most certificates are issued via a rather short chain of intermediate certificates. This is a relatively good finding: short chains reduce the attack surface, but allow to keep the root certificate offline. However, the number of intermediate certificates is high, pointing to a rather skewed distribution of the length of certificate chains.

Cryptography Our findings concerning the length of public keys were mostly positive. Short key lengths were still too common, but the trend showed a clear movement towards longer keys. Vulnerabilities like the Debian bug were already very rare. We could also show that MD5 is in the process of being phased out. This is a good finding as this algorithm cannot be relied on any more. Our monitoring showed that cryptography is generally not a weakness as both strong ciphers and long keys are used. At the time of our monitoring, the most common symmetric algorithms were AES in CBC mode and RC4. In the light of new attacks on both CBC and RC4, however, these are not optimal choices. It would be insightful to carry out monitoring again and determine whether clients and servers have moved to other block modes and moved away from RC4.

Issued certificates vs. encountered certificates Where sensible, we compared our findings for the data sets from active scans versus the ones from passive monitoring. The differences were rarely drastic. However, the data from our second, later monitoring run showed a clear increase of correct certificate chains. Furthermore, we found a number of very short-lived certificates in our data, which we attributed to the Tor network.

Development over time For several of the issues we investigated, we traced the development of the X.509 PKI over time. We found little development in fundamental TLS connectivity or properties of certificate chains. However, we found a trend to use more intermediate certificates (and short chains) and a movement towards longer key lengths. Unfortunately, the really critical issues (like correct chains, correct hostnames) did not show significant improvements. We note that the more dangerous attacks on the X.509 PKI happened after our investigation period (see Chapter 3). It would be interesting to determine whether the attacks caused a move towards more security in HTTPS configurations.

Influence of geographic factors We could show that results from our other vantage points did not differ crucially. The operators of CDNs seem to carry out certificate deployment with due care.

Correlation to rank We could determine that the quality of certificates (correct chains, hostnames, sensible values for validity and key length) correlated with the Alexa rank, but in a surprising way. We determined three categories of *valid* certificates: 'good', 'acceptable' and 'poor'. While the ratio of *valid* certificates was *higher* for the top-ranking sites, the fraction of 'poor' certificates among them was also *higher.*

Our overall conclusion is that the X.509 PKI is in a poorly maintained state, with a high fraction of certificates not being absolutely valid for the host where they are used. There is also very little improvement over time. The primary efforts in improving the X.509 PKI thus need to focus on sensible deployment and certification practices.

4.8. Statement on author's contributions

This chapter is an improved and extended version of the following paper: R. Holz, L. Braun, N. Kammenhuber, G. Carle. The SSL landscape—a thorough analysis of the X.509 PKI using active and passive measurements. *Proc. 11th ACM SIGCOMM Internet Measurement Conference (IMC)*, Berlin, Germany, November 2011 (reference [37]).

The author of this thesis carried out all scans and carried out the post-processing of the thus obtained data. He also carried out the post-processing of certificates in the data sets from monitoring. The author made major contributions to all results that concern evaluation of host properties (Section 4.3) and evaluation of certificates (Section 4.4). The author made major contributions to the paper.

For this chapter, the author reanalysed the data sets. Deviations to the numbers in the paper are due to this improved methodology. In the original publication, we used a simpler script to extract root stores that would include certificates in its output that were not meant to issue server certificates (e.g., root certificates for S/MIME) or were blacklisted (the demonstration MD5 certificate from [260] and Mozilla test certificates). Server certificates are not expected to be issued from such certificates. For this chapter, the author extracted all root stores again with the more precise script by Langley [211], which the author extended into a tool suite to work on older root stores, too [200]. The author's re-evaluation using this new method produced only marginal differences.

In the original paper, our results showed minor deviations due to two oversights. First, we accepted expired intermediate certificates as valid. In the new methodology, the author corrected this. The differences were again marginal (the re-evaluation of our data sets with respect to expired certificates showed differences of only 0.1%–0.7%). Second, an undocumented behaviour in OpenSSL had caused our instance to fall back on root certificates installed by the Linux distribution *in addition* to the correct root store. This means we slightly overestimated some numbers, e.g., the number of verifiable certificates. The re-evaluation showed the differences to be marginal. Furthermore, this OpenSSL behaviour actually emphasised our message as we *overestimated* the number of *correctly* issued certificates.

Static properties of certificates and hosts (i.e., such properties that do not depend on verification steps) were not analysed again. Using the new methodology, the author made a number of additions to the original results, and also added some corrections:

Section 4.3.1 The author added an analysis how many hosts (correlated to rank) had an open port 443. The author put this in relation to how many hosts actually offered TLS on this port.

Section 4.3.2 The author reanalysed the results in the lights of cryptographic developments since 2011, in particular with respect to Diffie-Hellman key exchanges, AES in CBC mode, and RC4.

Section 4.4.1 A mistake in the number of Wordpress certificates has been fixed.

Section 4.4.2 The author corrected a mistake in the plot.

Section 4.4.3 Marginal deviations in the percentages were fixed (less than 0.1%).

Section 4.4.5 Minor errors in the numbers were fixed by the author (deviation of about 2% in the case of matching CNs).

Section 4.4.8 The author added a comparison between distinct and non-distinct certificates and found that Debian-weak keys are less likely to be reused on multiple hosts.

Section 4.4.9 The author added a comparison with distinct certificates, where a number of extremely short-lived certificates could be associated with probable use of Tor.

Section 4.4.10 The author added results for multiple chains. He also analysed chain lengths a second time, with self-signed end-host certificates excluded, and compared the results. This analysis provides even stronger evidence for the move towards using more intermediate certificates. The author also added an analysis of the differing chain lengths for the second monitoring run and determined the reason to be the very different distribution of popular sites.

Section 4.4.11 The author fixed a mistake in the text describing the plot and clarified the statement on the convergence of certification paths from end-host certificates towards root certificates.

Section 4.4.13 The author reanalysed the data with a new method for counting and added a comparison to issuers in distinct certificates. The new results show that certificates from certain CAs are more likely to be reused on multiple hosts.

Section 4.4.15 The author reanalysed the data with improved criteria for the categories and added a comparison between earlier and later scan.

The author changed the above sections to reflect the additions and results from the new methodology. The following sections are also adapted from the paper. For Section 4.2, the author added details about the scans and added a discussion of the inaccuracy in the third-party data set. He also rewrote the section on the monitoring setup. For Section 4.4.2, the author added the algorithm to determine expired certificates. In Section 4.4.8, the author added a statement concerning the later work by Heninger *et al.*, who found an explanation for the duplicate keys we had found. Section 4.5 is an extended version from the paper, too. The author split the section into earlier and later work, and added related work for the latter category.

The following sections are from the paper, with only stylistic changes: 4.4.6, 4.4.7, 4.4.12, 4.4.14, and 4.6. Section 4.4.4 is a shortened version from the paper. All plots in this chapter are by the author.

Chapter 5.

5 Analysis of the OpenPGP Web of Trust

This chapter is an extended version of our previous publication [73]. Please see the note at the end of the chapter for further information.

Webs of Trust are an approach to PKI that is quite orthogonal to the hierarchically structured X.509 PKI that we analysed in Chapter 4. In a Web of Trust, every entity can certify any other entity. One particularly important Web of Trust is the one established by implementations of the OpenPGP standard (RFC 4880, [93]), e.g., Pretty Good Privacy (PGP) and the GNU Privacy Guard (GnuPG). In this chapter, we describe the results of a thorough investigation of the Web of Trust as established by users of OpenPGP.

5.1. Introduction and investigated questions

OpenPGP was conceived primarily as an encryption tool for private users, to be used in applications like email. Email constitutes one of the primary uses of OpenPGP to this day[1]. This use case is entirely different from the WWW: OpenPGP is not intended to be used to certify myriads of Web sites. Instead, OpenPGP means to establish cryptographic links between real persons and to exploit relationships between them to make it possible to assess the authenticity of their public keys. Due to its entirely different use case, the research questions to ask are different from those for the X.509 PKI—they have less to do with problems that are due to deployment and more with the relationships between keys. The latter are reflected in signatures.

OpenPGP does not need central entities that act as dedicated issuers of certificates. A 'certificate' in OpenPGP is simply a signature on a name (with email address), public key, expiry date, and optionally an indication of how thoroughly an entity's identity was verified by the signing party. Thus, one often speaks of 'signatures' and 'keys' instead of certificates when discussing OpenPGP.

The key to understanding OpenPGP lies in analysing the structure of the graph that the OpenPGP Web of Trust constitutes. It is possible to state certain properties that a 'good' Web of Trust must exhibit, and these are accessible to graph analysis. Using this form of analysis, we investigated the following research questions.

Fundamental statistics The first question to ask is what the size of the Web of Trust is, and how many keys with certification paths (i.e., signature chains) exist between them. This gives an insight for how many users the Web of Trust is potentially useful in the sense that they can use it to authenticate other keys or have their keys authenticated by others.

[1]OpenPGP is also often used in Linux distributions to sign software packages.

Usefulness of the Web of Trust A good Web of Trust must allow to find certification paths from one key to many others, otherwise it is not useful. This degree of usefulness is the next question we investigate. The length of the paths is also important: short paths reduce the number of entities on the path that a user has to trust and thus increase a user's chances of accurately assessing a key's authenticity. Giving and receiving many signatures is important, too: it increases the chances of several redundant paths between keys, which is beneficial for GnuPG's trust metrics.

Robustness Keys in the Web of Trust are stored on key servers, from where they can be retrieved. They are subject to the usual life-cycles in a PKI, i.e., they may expire, be withdrawn, and possibly be compromised. The effect would be that certification paths would break. We thus decided to investigate how the Web of Trust reacts to the random and targeted removal of keys, i.e., to which degree keys remain connected via (redundant) certification paths.

Social relations A good Web of Trust should model social relations and social clustering well: where communities of users exist, the chances of being able to accurately assess trustworthiness of users within the same community increase. We decided to investigate community structures in the Web of Trust and attempt to map them to social relations. Furthermore, we decided to investigate whether the Web of Trust shows the so-called Small World effect, which is common in networks of social origin.

Cryptography Just as with X.509, the cryptographic algorithms and keys in use in OpenPGP should be sufficiently strong. We decided to investigate whether this is the case.

Historical development The Web of Trust graph is unique in the sense that all certification information in the Web of Trust is preserved on key servers. Hence, it is possible to derive the historical development of the Web of Trust since its conception in the 1990s. As previous studies of the Web of Trust dated back almost ten years, we were interested to see to which extent the Web of Trust had changed.

5.2. Methodology

We used primarily methods from graph analysis to determine properties of the Web of Trust. In this section, we describe how we extracted the graph topology and summarise the metrics we used.

5.2.1. Graph extraction

As mentioned in Section 2.5, a system of publicly accessible key servers allows users to upload their keys together with the signatures so other users can retrieve them. The keys servers synchronise via the Synchronizing Keyservers (SKS) protocol. Keys are never removed from a key server—consequently, a snapshot of a key server's database contains the history of the entire Web of Trust. We modified the SKS software to download a snapshot of the key database as of December 2009.

Table 5.1 shows properties of our data set after the extraction. The data set contains about 2.7 million keys and 1.1 million signatures. Of these, about 400,000 keys were expired, another 100,000 revoked. About a further 52,000 keys were found to be in a defective binary format. The *actual* Web of Trust, which consists only of valid keys that have actually been used for signing or have been signed, is made up of 325,410

Total number of keys	2,725,504
Total number of signatures	1,145,337
Number of expired keys	417,163
Number of revoked keys	100,071
Number of valid keys with incoming or outgoing signatures	325,410
Number of valid signatures for the latter set of keys	816,785

Table 5.1. – Our data set for the analysis of the OpenPGP Web of Trust.

keys with 816,785 valid signatures between them. Consequently, the majority of keys in the data set is not verifiable (no signature chains lead to them) and does not belong to the Web of Trust.

Note that our method implies an important caveat. Users are not required to upload their keys. Yet there is no centralised or structured way to search for and download keys other than key server networks. Consequently, our data set has the inherent bias that it contains only keys from users who took the extra step to upload their keys. As the data set is relatively large, statistical analysis will still yield meaningful results, but it is important to keep this limitation in mind. The number of unpublished keys is unknown, unfortunately.

When representing the Web of Trust as a graph, we represented keys as nodes and signatures as directed edges. This was a deliberate choice. An alternative would have been to map keys to individual persons. However, such a mapping is not easy to define: different users may have the same names, or the same users may spell their names differently every time they create and upload a new key, or they may simply use pseudonyms. Although the user ID in an uploaded key contains an email address, this is not particularly reliable, either, as these can change, too. Ultimately, it is keys that sign other keys, and we thus chose to analyse a key-based graph.

5.2.2. Terms and graph metrics

Based on the common notions of graph theory, we define some terms, following [83] herein. In the following, let V be the set of nodes of the graph G, with $|V| = n$. u and v indicate nodes.

Strongly Connected Components (SCCs) An SCC is a maximal connected subgraph of a directed graph where there is at least one directed path between every node pair u, v. Note that the paths from u to v and v to u may incorporate different nodes.

Distances between nodes The distance d between two nodes is the length of the shortest path between them.

Distances in the graph The characteristic distance of a graph, $\bar{d}$, is the average over all distances in the graph, i.e., the average path length:

$$\bar{d} = \frac{1}{n^2 - n} \sum_{u \neq v \in V} d(u, v) \tag{5.1}$$

Eccentricity The eccentricity of a node u, $\epsilon(u)$, is defined as the maximum distance to another node, i.e.

$$\epsilon(u) = \max\{d(u,v) | v \in V\} \tag{5.2}$$

Graph radius and diameter The diameter of a graph is defined as the maximum over all eccentricities:

$$dia(G) = \max\{e(u) | u \in G\} \tag{5.3}$$

The radius is defined as the minimum over all eccentricities:

$$rad(G) = \min\{e(u) | u \in G\} \tag{5.4}$$

Node neighbourhoods We define the h-neighbourhood of a node v as the set of all nodes from which the distance to v is at most h:

$$N_h(v) = \{u \in V | d(v,u) \leq h\} \tag{5.5}$$

Clustering coefficient The clustering coeffcient indicates the probability that two neighbours of a node have an edge between them, i.e., are neighbours themselves.

Let $G = (V, E)$ be the undirected graph. A triangle $\triangle = \{V_\triangle, E_\triangle\}$ is a complete subgraph of G of size 3, i.e., $|\triangle| = 3$. The number of triangles of a node v is given by $\lambda(v) = |\{\triangle : v \in V_\triangle\}|$. A *triplet* of a node v is a subgraph of G that consists of v, two edges, plus two more nodes such that both edges contain v. The number of triplets of a node v can be given as $\tau(v) = \binom{d(v)}{2}$. The *local clustering coefficient* of v is defined as

$$c(v) = \frac{\lambda(v)}{\tau(v)} \tag{5.6}$$

$c(v)$ indicates how many triplets of v are triangles. The *global clustering coefficient* of G can then be defined as:

$$C(G) = \frac{1}{|V'|} \sum_{v \in V'} c(v) \tag{5.7}$$

with $V' = \{v \in V : d(v) \geq 2\}$ to disallow non-defined values for $\tau(v)$.

Correlation of node degrees as defined by Pastor-Satorras et al. Following Pastor-Satorras *et al.* [59], we define a measure for the correlation of node degrees:

$$knn = \sum_{k'} k' P_c(k'|k) \tag{5.8}$$

gives the average node degree of neighbours of nodes with degree k. $P_c(k'|k)$ indicates the probability that an edge that starts at a node with degree k ends at a node with degree k'.

Assortativity coefficient The *assortativity coefficient* [55] is a measure whose purpose is similar to the function defined in Definition 5.8. It measures the degree of *assortative mixing* in a graph: nodes with high degree that are connected mainly to other nodes with high degree. The *assortativity coefficient* takes values between -1 and 1. Positive values indicate assortative mixing, negative ones do not. According to Newman [55], assortative mixing is a property that distinguishes social networks from other real-world networks (e.g. technical or biological ones). It can thus be used to differentiate between similar graphs that exhibit the so-called Small World effect. We return to this in Section 5.3.2.

5.3. Results

In this section, we present the results of our analysis. We begin with an analysis of the Web of Trust's macro structure.

5.3.1. Macro structure: SCCs

An SCC defines a subset of the graph where there is at least one signature chain between every key pair. SCCs are important for participants of the Web of Trust: mutual verification of key authenticity is only possible for participants within the same SCC. An optimally meshed Web of Trust should be one giant SCC.

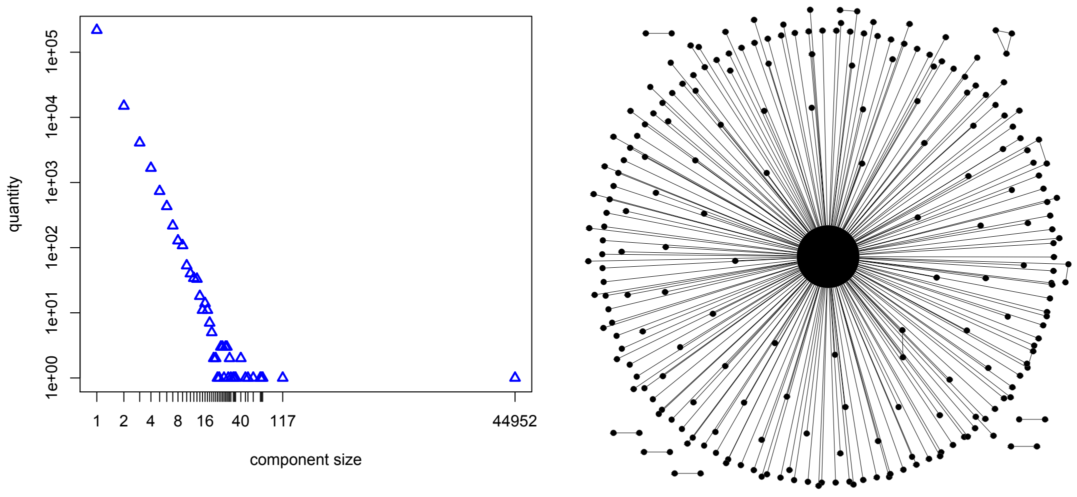
We computed the SCCs of the graph, and found 240,283 SCCs in the Web of Trust. However, more than 100,000 of these consisted of a single node and about 10,000 SCCs consisted of node pairs. The Largest Strongly Connected Component (LSCC) consisted of about 45,000 nodes. The remaining SCCs mostly had a size between 10 and 100 nodes. Figure 5.1(a) shows the distribution. Many SCCs have *unidirectional* edges to the LSCC, but extremely few have edges between each other. The SCCs can be arranged in a star formation around the LSCC in the middle (Figure 5.1(b)).

Out of all smaller SCCs, about 18,000 nodes showed a unidirectional edge into the LSCC, making it (in principle) possible for such a key to verify keys from the LSCC. In the other direction, 92,000 keys outside the LSCC are reachable from a key within the LSCC. We found three interesting hubs in the LSCC and one regional particularity. The German publisher Heise, the non-profit CA CAcert and, until recently, the German association DFN-Verein[2] operate or have operated CAs to sign keys. Together, they signed about 4200 keys in the LSCC in our data set. The Heise CA alone signed, in total, 23,813 keys—yet only 2578 of these were in the LSCC.

This SCC structure gravely impacts the usability of the Web of Trust. First of all, the large number of smaller SCCs means that even among those users who have made the effort to upload their keys to a key server, most do not participate actively in the Web of Trust. Otherwise, their SCCs would already have merged with the LSCC (one mutual signature is enough). This is also emphasised by the following comparison. The ratio of edges to nodes in the LSCC is 9.85; the same ratio for the total Web of Trust is 2.51. Signature activity in the LSCC must thus be much higher than in the rest of the Web of Trust—such strong user activity is very desirable to achieve a better meshing in the Web of Trust.

Second, a high percentage of participants in one of the smaller SCCs is unable to verify most keys in the Web of Trust. The LSCC is really a structure of paramount importance: the keys in the LSCC constitute only 14% of the keys in the Web of Trust, but only the owners of these keys can really profit from the Web of Trust to verify the authenticity of unknown keys. They can build signature chains to all keys in the LSCC plus to twice as many keys outside the LSCC. Thus, a recommendation for new participants would be to obtain a signature from a member of the LSCC as early as possible to make their key verifiable. A good choice is also to get a (mutual) signature of one of the CAs in the LSCC. With such a signature, paths can be built to all keys in the LSCC, plus to a large number of keys outside the LSCC that are only reachable via the CA. This emphasises that a Web of Trust can benefit from CAs.

[2]This entity also acts as a subordinate CA in the X.509 PKI, where it operates a large network of RAs.



(a) (b)

Figure 5.1. – (a) Size distribution of SCCs. (b) Plot of SCCs down to a size of 8.

The remainder of our analysis focuses on the LSCC as the most relevant component for participants.

5.3.2. Usefulness in the LSCC

'Usefulness' is a term that is difficult to express formally. It can be defined in several dimensions. We presented the most relevant ones in Section 5.1. When discussing the implications of distances between nodes, we will generally refer to the default settings of GnuPG as this is a popular implementation of OpenPGP. There is a further important issue to take into account here. In OpenPGP, a signature does not store the so-called 'introducer trust' (see Section 2.5), i.e., information how much a given user trusts another user to verify a participant's identity accurately before issuing a signature. Such information is always stored locally and never released. This has an important consequence for us: when we discuss the usefulness of the Web of Trust, we really assume that the links between keys, which are expressed as signatures, would be considered trustworthy by a verifying entity and thus would be usable to this entity. In other words, we can only assess the 'best case' for the Web of Trust rather than the average or worst case, because we do not have the information how introducer trust is distributed in the Web of Trust.

Distances

We first analysed distances between keys in the graph. The *average distances between nodes* in the LSCC (see Figure 5.2(a)) range between 4–7, which is at best just below GnuPG's limit, which by default allows a maximum path length of 5 in assessing a key's authenticity. At worst, these values exceed it. The characteristic distance of the LSCC is 12.14. Its eccentricity is much higher: it is almost exclusively between 26–31. To determine the implications for usefulness, we identified how many keys are reachable from a given key within a certain distance.

We computed the set of verifiable keys as the nodes in a *h-neighbourhood* for $h = 1, \ldots, 5$. Figure 5.3 shows the cumulative distribution function of h-neighbourhoods. For the 2-neighbourhood, we see a steep incline, from which we can conclude that this

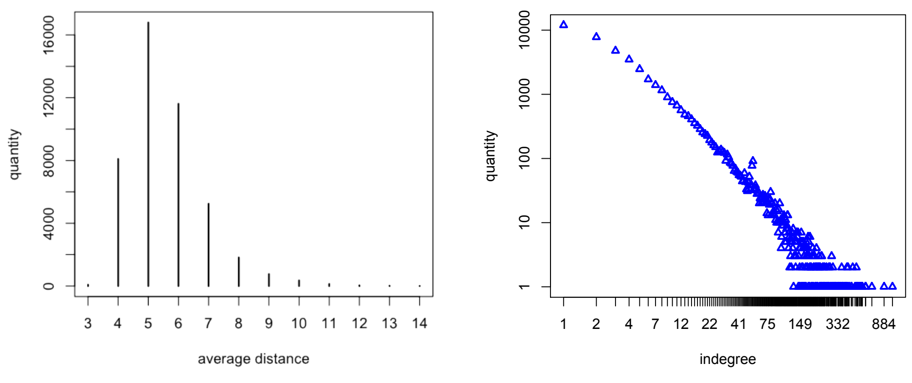


Figure 5.2. – Distribution of (a) average distances, (b) indegree in LSCC.

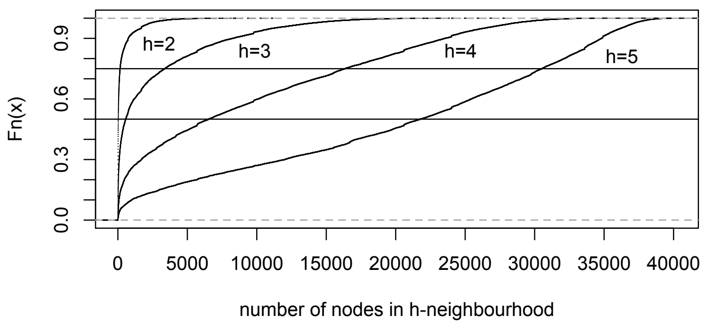


Figure 5.3. – CDF of reachable nodes due to h-neighbourhoods.

neighbourhood must be relatively small for all nodes. The size of the neighbourhoods grows considerably for increasing h. For $h = 3$, the third quartile is about 3300. For $h = 4$ and $h = 5$, it becomes 16,300 and 30,500, respectively.

Our findings indicate that signature chains within GnuPG's restrictions are sufficient to make a very large fraction of the keys in the LSCC verifiable. This is a good result for usefulness and shows that the LSCC is meshed quite well. However, for $h = 5$, the maximum number of reachable keys we found was 40,100. This means that, on average for all keys, there will be almost 5000 keys (a tenth of the LSCC) to which no path at all can be found within GnuPG's restrictions.

Small World effect and social links

The size of 5-neighbourhoods shows that paths are frequently very short. A possible explanation for this is a Small World effect, which—following [56]—can be informally understood to be the phenomenon that the average path length between any two nodes is significantly shorter than could be expected by judging from graph radius and diameter. A high clustering coefficient is often viewed as indicative. We investigated this in the LSCC. As there does not seem to be a universally accepted definition of the clustering coefficient for directed graphs, we reduced the directed graph to an undirected one (omitting the direction of edges and merging duplicates). The clustering coefficient we computed is $C = 0.46$. This indicates that, *on average*, roughly half of all neighbours of a node have edges between them. The value is on the same order as described in [56] for social networks with strong clustering. The characteristic distance in the LSCC is 6.07, while the diameter of the graph is 36 and the radius 16. Our finding is that the LSCC does indeed show a Small World effect. This indicates social clustering. Together with the short paths, this would make trust assessments easier for users. We further explore the social nature of the Web of Trust in Section 5.3.4.

Node degrees

GnuPG's trust metrics view redundant and distinct signature chains as beneficial in assessing a key's authenticity. A high node indegree thus means that the corresponding key is more likely to be verifiable by other keys. A high outdegree increases the likeliness to find redundant signature chains to other keys.

We computed the average indegree (and outdegree) in the LSCC as 9.29. However, as can be seen in Figure 5.2(b), the distribution of indegrees in the LSCC is skewed. The vast majority of nodes have a low indegree (1 or 2). The result for the outdegrees is very similar: as can be seen in Figure 5.4(a), there is a positive correlation between indegree and outdegree of a node. The plot for outdegrees is indeed so similar to the one for indegrees that we omitted it here. About a third of the nodes in the LSCC have an outdegree of less than 3. Together, these results mean that the Web of Trust's usefulness has an important restriction: many nodes need to rely on single certification paths with 'full' introducer trust and cannot make use of redundant paths.

Mutual signatures (reciprocity of edges)

If many Web of Trust participants cross-sign each other, this would be a great improvement in the overall verifiability of keys. We computed the reciprocity of edges, i. e., the fraction of unidirectional edges for which there is also a unidirectional edge in the other direction. The LSCC has a reciprocity value of 0.51. This shows that there is room for improvement: the LSCC would profit much if more mutual signatures were given.

5.3.3. Robustness of the LSCC

The robustness of the LSCC is also an interesting topic: how is the LSCC connected internally, and hence how sensitive is it to removal of keys? In the context of OpenPGP, the random removal of a node can be the result of an event like key expiration or revocation, which invalidates paths leading over the key in question. These events can and do occur in practice. Targeted removal of a key, however, is very hard to accomplish as SKS never deletes keys and stays synchronised. An attacker would need an unlikely high amount of control over the SKS network to make a key disappear.

Scale-free property

Scale-freeness in a graph means that the node degrees follow a power law, i. e., the distribution of degrees can be expressed as a function $f(x) = ax^k$. Connectivity-wise, scale-free graphs are said to be robust against random removal of nodes, and vulnerable against the targeted removal of hubs (which leads to partitioning). This is usually explained by hubs being the nodes that are primarily responsible for maintaining overall connectivity [2]. We thus investigated first to which extent the Web of Trust shows this property.

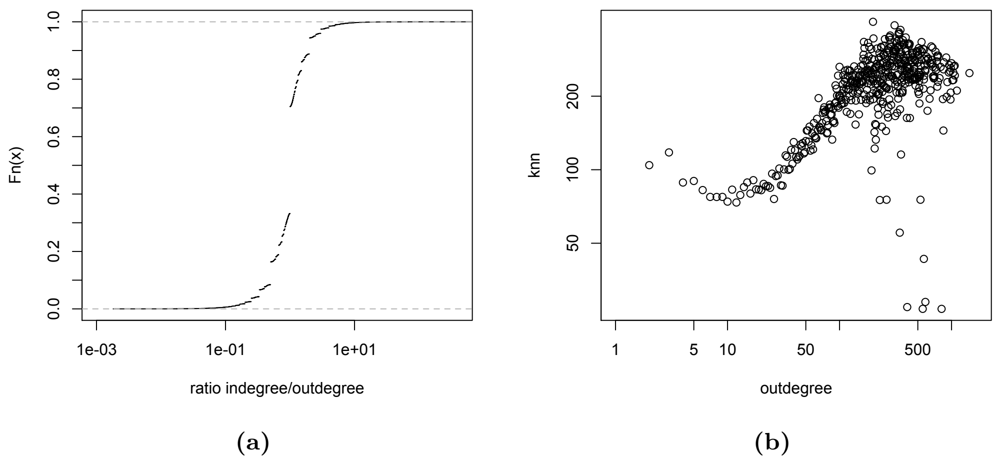


Figure 5.4. – (a) CDF of ratio indegree to outdegree in LSCC. (b) Correlation of node degrees according to Definition 5.8: average outdegree (knn) of neighbours of nodes with degree k.

The double-log scale in Figure 5.2(b) could lead one to the conclusion that the distribution of node degrees follows a power law. However, Clauset *et al.* argued in [15] that this is not indicative and methods like linear regression can easily be inaccurate in determining a power law distribution. We followed the authors' suggestion instead and used the Maximum Likelihood method to derive power law coefficients and verified the quality of our fitting with a Kolmogorov-Smirnov test. The authors of [15] give a threshold of 0.1 to safely conclude a power law distribution. Our values for indegrees and outdegrees were 0.012 and 0.011, respectively. As this is off by a factor of ten, our conclusion is that a power law distribution is not plausible. Consequently, the graph cannot be scale-free in the strict sense of the definition. This finding is contradictory to earlier works by Boguñá *et al.* [9] and Čapkun *et al.* [74].

The question is yet whether the graph is still similar to a scale-free one. Apart from high variability of node degrees, a set of high-degree nodes that act as interconnected hubs are characteristic for scale-free graphs [2, 49]. The positive correlation between the degree of nodes and the average degree of their neighbours (Figure 5.4(b)) suggests that nodes with high outdegrees do indeed connect to other such nodes with high probability. To bolster our finding, we computed the assortativity coefficient and obtained a value of 0.113. This is similar to what has been computed for other social networks with a hub structure [56]. Our conclusion is that the graph is similar to a scale-free one and exhibits a hub structure, but is not scale-free in the strict sense.

Random removal of nodes

Based on this finding, we investigated how the LSCC reacts to random removal of nodes. We removed nodes and recomputed the size of the remaining LSCC as an indication of loss in overall connectivity. For random removal, we picked the nodes from a uniform distribution. Figure 5.5 shows our results. The graph is very robust against the random removal of nodes: we have to remove 14,000 nodes to cut the LSCC's size by half. To reduce it to a quarter, we have to remove more than half the nodes (25,000).

The conclusion here is that events like key expiry or revocation do not greatly influence the robustness, and consequently the usability, of the Web of Trust.

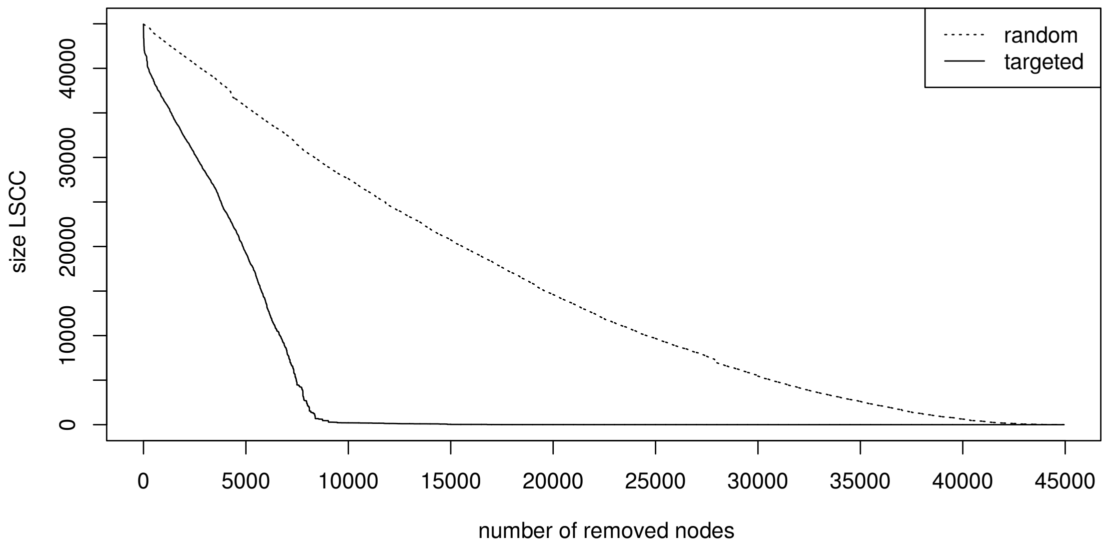


Figure 5.5. – Removing nodes at random and in a targeted fashion and recomputing the size of the LSCC.

Targeted removal of nodes and CAs

For targeted removal, we chose nodes with highest degrees first. The graph was more robust than expected. When we removed all nodes with a degree of more than 160 (240 nodes), the size of the LSCC was still 40,000. Only when we proceeded to remove all nodes with a degree of more than 18 (about 5000 nodes), the LSCC was half its size. Removing 2500 more nodes, we finally cut the LSCC down to about one ninth of its original size. This means that nodes with lower degrees (less than 18) play a significant role in overall connectivity (although the decay of the LSCC is quite pronounced once they are also removed). The rather slow decay stands in contrast to the rapid decay upon removal of the best-connected nodes that is commonly observed in scale-free networks. Targeted removal of keys does not greatly affect the Web of Trust, and is not an efficient attack. The hub structure is not the single reason for highly meshed connectivity in the Web of Trust.

We decided to strengthen the attack by removing the keys of the three CAs. Our finding was similar: the LSCC split into one LSCC of size 42,455 and 1058 very small SCCs. This means that the CAs, although beneficial in making keys verifiable, are not responsible for holding the LSCC together.

5.3.4. Community structure of the Web of Trust

Social clustering is an important property in a Web of Trust: it is more likely that members of a cluster know each other at least to some extent and can thus assess the trustworthiness of particular keys better. The Small World property was a first hint at social clustering. Newman and Park noted that a high degree of clustering is typical for social networks [58]. Fortunato [25] calls such subsets of nodes ‘communities’ if the nodes have high intra-connectivity in their subset, but the subset as such shows a much lower connectivity to nodes outside.

Community detection

We analysed the Web of Trust with state-of-the-art algorithms for community detection to determine whether a pronounced community structure exists and can be mapped to

Method	Modularity	Communities found (size > 3)
BL ($l = 2$)	0.70	936
BL ($l = 5$)	0.71	186
COPRA ($v = 1$)	0.78	1421
COPRA ($v = 3$)	0.79	1354

Table 5.2. – Dissection of the LSCC into communities: algorithms BL and COPRA. Note that the definition of modularity for overlapping communities is different. The values for BL and COPRA are thus not directly comparable.

'real-world' relationships. We also attempted to determine whether signing events like Key Signing Parties can be identified in the graph. These are organised events where a large number of participants come together in person and cross-sign each other's keys.

Unfortunately, algorithms for community detection are often defined for undirected graphs. Signatures also store little information that could help identify social links and events in time. We decided to use DNS domains in user IDs and timestamps of signature creation as a basis. As an algorithm for a directed graph, we chose the one by Rosval *et al.* [66]. For undirected graphs, we chose the algorithms by Blondel *et al.* [8] (BL) and COPRA [31], based on suggestions in [44]. COPRA allows overlapping communities, but is non-deterministic. We ran it ten times and computed the differences. As a measure for the quality of a dissection, we used *modularity* [57], which relates the amount of intra-cluster edges of a graph with communities to the expected value for a graph without communities. Note that the definition for overlapping communities is different, so the values for COPRA and BL cannot be compared directly.

Only the algorithms by Blondel *et al.* and COPRA yielded useful results. The algorithm by Rosval *et al.* computed a dissection into 2869 communities, almost all of them without any intra-cluster edges. The lack of intra-cluster edges means that the algorithm did not detect any true communities at all. We thus had to consider its results unreliable. We ignored them in our subsequent evaluation.

BL and COPRA

Table 5.2 shows the results of dissections with BL and COPRA for communities of size less than 3. Both BL and COPRA are configurable: BL can be repeated in iterative phases and COPRA requires a (user-chosen) parameter v to reflect the degree of overlapping. For BL, phase 2 ($l = 2$) yielded the best results (plausible number of communities, high modularity). For COPRA, values of v up to 3 were found best. We know from [14] that modularity values larger than 0.3 indicate a significant community structure. Depending on the algorithm and chosen parameters, between 94% (COPRA) and 99% (BL) of nodes in the LSCC belonged to such a community.

BL and COPRA agree on the same orders of magnitude with respect to the number of communities and nodes therein. The high modularity values and the general shape of community distributions by size (see Figures 5.6(a) and 5.6(b)) are also similar. Most communities are very small, but a significant number of large or very large communities exist. Similarities, however, end here. COPRA indicates one extremely large community of 19,000–21,000 members. BL finds more communities of medium size (100–500) and mid-large size (500–5000). To further investigate this, we analysed how communities are connected. COPRA found that most small communities are clustered around the largest community and mostly link only to this community. BL found several large communities to which the smaller communities connect.

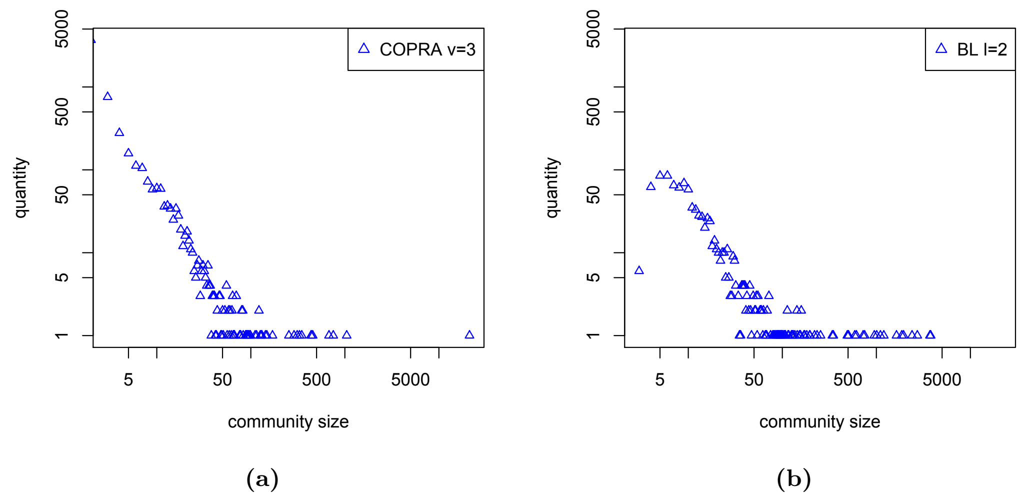


Figure 5.6. – Distribution of communities by size.

Method	dominated by TLD	assignable to TLD	dominated by SLD	assignable to SLD	signatures within 30d
BL ($l = 2$)	53%	45%	4%	27%	12%
BL ($l = 5$)	47%	47%	8%	21%	14%
COPRA-1	58%	40%	13%	30%	40%
COPRA-3	58%	38%	14%	31%	41%

Table 5.3. – Community structure with respect to membership in top-level domains (TLD) and second-level domains (SLD).

Mapping to domain names and keysigning parties

We analysed how the community dissections mapped to top-level and second-level domains (TLDs and SLDs) in the user IDs. If the same TLD or SLD occur very frequently in a cluster, this would be an indication that the cluster represents a true community of users. We say a community is *dominated* by a domain if at least 80% of its nodes belong to that domain. We say a community can be *assigned* to a domain if at least 40% of its nodes belong to it.

Table 5.3 shows the results. For both BL and COPRA, we found that a large percentage of communities are dominated by a top-level domain: between 47% and 58%. If a community was not dominated, we checked if it could at least be assigned. A further 38%–47% could be said to be assignable to a TLD. This result did not change much when we disregarded generic TLDs (`.com`, etc.): with COPRA, 38% of communities were dominated by a country's TLD and a further 23% were still assignable. Results for BL were similar. Together, assigned and dominated communities make up by far the largest part of communities found (98% for BL-2 and COPRA, $v = 1$). However, the picture changes for second-level domains. With COPRA, only about 13% of communities are dominated by an SLD and only a further 30% of communities can be assigned to an SLD.

Keysigning Parties are events where one can expect signatures to be uploaded to key servers within a short time frame. Table 5.3 shows the percentage of nodes in the communities where signatures were created within a month. We find poor results for BL, but much better ones for COPRA. In about 40% of communities, the signatures were created within 30 days of each other.

Conclusion with respect to community detection

Concerning community detection, it is difficult to reach compelling conclusions. We provide ours as a basis of discussion. Both algorithms agreed that a large number of smaller communities exist. Given the huge number of TLDs and SLDs and given that the Web of Trust graph spans more than a decade, the results seem statistically significant enough to conclude that the community structure does indeed capture some 'social' properties of the Web of Trust. However, grouping by TLD is a blunt measure, and the mappings to SLDs were by far not as compelling. Our tentative conclusion is that the signing process in the Web of Trust is indeed supported, to a traceable extent, by real-world social links. The social nature of the Web of Trust is not a myth. At least where certification paths are short, the community structure should make it easier for users to assess the trustworthiness of a key. Beyond this result, however, community detection is yet too imprecise to offer more succinct conclusions.

5.3.5. Cryptographic algorithms

We present results which cryptographic algorithms were used in our data set. Tables 5.4(a) and 5.4(b) present results for hash and public key algorithms, respectively. Keys with a length of less than 1024 bits were rare. The key lengths would have been secure for 2009. Today, it is fair to say that a move towards longer key lengths should be made rather sooner than later. Concerning hash algorithms, the stronger SHA1 algorithm is by far the most frequent. The weak MD5 algorithm is still present at about 10%. SHA2 variants are a small minority. Concerning public key algorithms, ElGamal/DSA constitutes the vast majority (four times more than RSA).

The OpenPGP standard allows to set certain values when signing a key to indicate how thoroughly a key owner's identity has been checked before signing. Precise semantics are not defined, however. The standard value is 'generic', which is also the most common value found (67.1% of signatures in the LSCC), but 26.8% of signatures are issued with the highest value 'positive' and 6.4% with 'casual'. As setting a value other than 'generic' requires user-interaction, it shows that at least a part of the users in the Web of Trust takes the signing process seriously. The values that these users set can be helpful for others when they have to assess the authenticity of a key.

5.3.6. History of the Web of Trust

We also examined the Web of Trust's history. Recall that SKS servers never delete keys; thus our snapshot of the Web of Trust contains its entire history. Our investigation starts in 1991, when PGP was released. Figures 5.7(a) and 5.7(b) show the development of the Web of Trust and the LSCC, respectively. A significant growth occurred only after 1997. This correlates with the founding of PGP Inc., the release of version 5 of the PGP software, and the beginning of the German Heise CA's work. The growth of the total Web of Trust slowed down after 2001, and the growth of the LSCC after 2005. We do not surmise any specific reason for this. Possible explanations are saturation or the advent of S/MIME, a rivalling standard. Figures 5.8(a) and 5.8(b) show the rate at which new PGP keys were added to the Web of Trust and the LSCC, respectively.

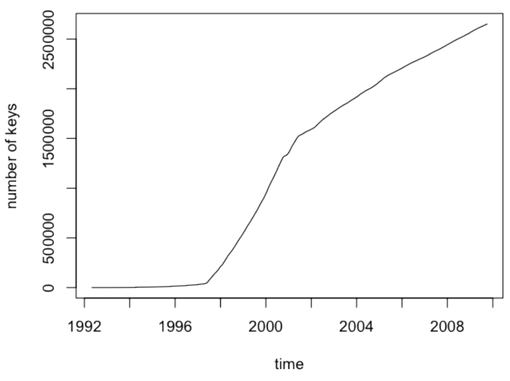
Algorithm	Occurrences
SHA1	398,849
MD5	41,700
SHA256	5031
SHA512	2472
SHA224	532
RIPE-MD/160	122
Signatures total	446,325

(a)

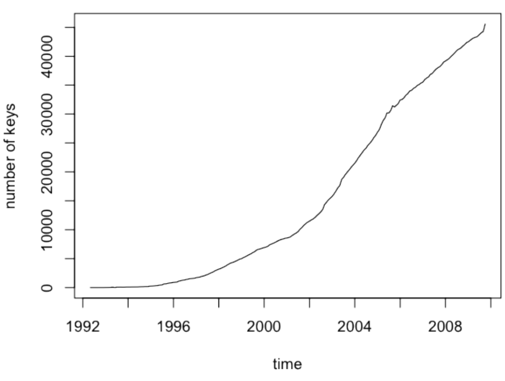
Algorithm	Occurrences
ElGamal/DSA-1024	36,555
RSA-1024	3903
RSA-2048	2408
RSA-4096	1198
RSA-768	257
RSA-512	203
RSA-3072	96
Keys total	44,952

(b)

Table 5.4. – Occurrences of (a) hash algorithms, (b) public key algorithms.



(a)



(b)

Figure 5.7. – Development of key population of Web of Trust (a) and LSCC (b) over time.

ElGamal/DSA keys make up the largest part, probably due to RSA's once-strict licence requirements. In 2009, RSA became the default—this is reflected in the plot of the total Web of Trust. Interestingly, the switch is almost untraceable in the LSCC. The LSCC seems to show more resilience to changes. Our findings do not have direct implications for security. It is known that ElGamal/DSA signatures need strong pseudo-randomness during execution in order to be secure, and this has been found to be problematic in the case of embedded devices used in conjunction with protocols like SSH [35]. However, we find it plausible that the majority of OpenPGP keys are used on desktop machines or even stronger hosts, where entropy is not a problem. Concerning earlier results for the scale-free property, we find that the Web of Trust was much smaller at the time of these previous analyses—thus, previous work analysed a very different network.

5.4. Related work

The OpenPGP Web of Trust has been the subject of investigation before, albeit at other stages of its development and with a focus that was less on security-relevant properties.

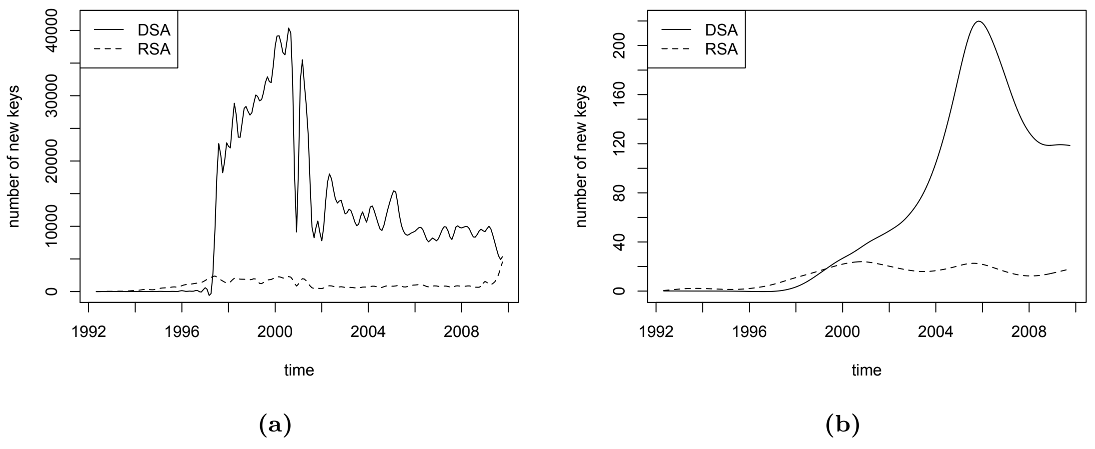


Figure 5.8. – Rate of new PGP keys added to the whole Web of Trust (a) and the LSCC (b).

Čapkun *et al.* [74] analysed several structural aspects of the Web of Trust of 2001. They did not investigate aspects like communities but presented a model to create similar graphs. They found a small characteristic distance and a high clustering coefficient. The authors claimed to have found a power law distribution for node degrees. Our own findings are that a power law distribution is not plausible. However, the graph is similar to a scale-free one, although its hub structure is not solely responsible for robustness. Note however that the rigid methods in [15] were generally not as widely in use then. Boguñá *et al.* [9] also analysed a PGP graph from 2001. They converted the graph to an undirected one and analysed node degrees and clustering coefficient. They also claimed a power law for node degrees and determined a clustering coefficient on the same order as the one we found. The authors also applied an (older) algorithm for community detection. They claimed the community distribution follows a power law, too. All of the above have in common that they used significantly older data sets, and the focus was less on security issues like usefulness and robustness. Furthermore, our community dissection was conducted with more recent algorithms, with the aim of mapping communities to real-world groups.

The OpenPGP community has also contributed some effort in analysing the Web of Trust's structure. The first study that we know of was by Burnett in 1996 [217], which was followed by another study in 1997 [218]. The Web of Trust was small at that time, with the LSCC at about 2000 keys (1996) and 3100 keys (1997), although some of the difference may be attributable to the different sources (Burnett used two different publicly available key sets). The average distance between nodes was about six in both years. The wotsap project [153] creates snapshots of the signatures in the Web of Trust. However, it only considers the LSCC and does not store other key properties. We also found the data set to be incomplete (10% of keys missing), due to a bug. Penning [246] used the wotsap data set to determine aspects like distances, node distribution and robustness based on node removal.

5.5. Summarising view

In the following, we bring the different findings of the last section together to provide a summarising view.

We found that only keys in the LSCC can really profit from the Web of Trust in the sense that they can verify the authenticity of a large number of other keys and find redundant certification paths to them. Unfortunately, this also means that the reach of the Web of Trust is limited to a fraction of its users: only about 45,000 keys out of two millions can use the Web of Trust without restrictions. A large fraction of keys in the smaller SCCs can make very little use of the Web of Trust (or none at all). However, for users with keys in the LSCC, the situation is much better. We found their certificate chains to be relatively short. There is also a pronounced Small World effect. We followed this up with an investigation of the community structure of the Web of Trust. While algorithms for community detection can capture the social groups of the Web of Trust only on a very coarse level, the graph does exhibit a strong community structure.

Another positive aspect is that about 40,000 of 45,000 keys are reachable within GnuPG's restrictions, and several thousand even via just three hops or less. This is positive for the Web of Trust as it can aid users in making better trust assessments regarding other keys that are close and in the optimal case also in the same community. The CAs we found greatly help make keys verifiable, although they are not relevant for holding the graph together. Random removal of keys (e.g., due to expiration or revocation) is not a problem for the robustness of the Web of Trust. The Web of Trust is also very robust against targeted attacks.

However, we found that low indegrees and outdegrees are far too common. This reduces the number of redundant paths between keys, which means that many users would need to have 'full' introducer trust in known entities. More mutual cross-signing would help here.

In essence, our conclusion is that the Web of Trust is likely to be quite an effective PKI structure *within smaller node neighbourhoods, and particularly for those users that frequently sign other keys and are active in the Web of Trust.* The cryptographic algorithms that are in use can generally be considered to be still secure. However, keys that have issued MD5-based signatures should be replaced and signatures renewed. Also, a stronger move towards key lengths of more than 1024 bits is desirable.

5.6. Key contributions of this chapter

This chapter addressed Research Objective O2.2: we analysed the PKI established by the OpenPGP Web of Trust. The purpose of this PKI is different to the X.509 PKI for HTTPS: instead of authenticating WWW servers, the Web of Trust serves the purpose of authenticating bindings of keys to user identities. This PKI is not accessible to scans; but a snapshot is easy to create due to the availability of key servers. Analysing the Web of Trust requires a different methodology, however, namely graph analysis. Our key contributions are as follows:

Fundamental statistics We extracted the Web of Trust as of December 2009 and determined the fundamental statistics. We found that 2.7 million keys exist, but there are only 1.1 million signatures. About 400,000 keys are expired; 100,000 are revoked. We determined the actual size of the Web of Trust to be much lower than the 2.7 million keys we found: only 325,000 keys had valid signatures. This grave mismatch means that the vast majority of keys that were uploaded to key servers are not verifiable within OpenPGP's trust model. Mutual verification of keys is only possible within the SCCs of the Web of Trust. More than 240,000 SCCs exist, but the vast majority is very small, with sizes mostly one or two and some larger ones of sizes in the range of 100 keys. The largest SCC (LSCC) contains just 42,000 keys. The Web of Trust is really only useful for the owners of these keys.

Usefulness We computed to which degree the LSCC is useful for its users to verify the authenticity of a large number of other keys. To this end, we computed the number of keys reachable via 1–5 hops. We found that, on average, only a relatively small number of keys was reachable via two hops, but the number increased drastically for three hops and reached about 40,000 keys for five hops. This is a large number and shows the usefulness of the Web of Trust; however, it also means that, from any chosen key, about one eighth of keys in the LSCC could not be verified. With keys represented as nodes, we computed the node degrees within the LSCC. The node degrees correspond to signatures given or received. We found that a third of keys have received less than three signatures. Within OpenPGP's trust model, these keys can only be used by the signing parties, but not by other entities. More mutual signatures would be beneficial.

Robustness We investigated the robustness of the LSCC by determining its size after random and targeted removal of keys. The LSCC is not scale-free in the strict sense—it contains more inter-connected keys acting as hubs than the scale-free property would suggest. This helps make the network very robust against removal of keys. We found that random removal had little effect, and the LSCC was more robust against targeted removal than would be typical for a scale-free network.

Social relations We investigated the social structure of the Web of Trust. We computed the clustering in the LSCC and found that it exhibits a strong Small World effect. This hints at a strong social structure. We used state-of-the-art community algorithms to determine whether communities exist in terms of common top-level domains and common second-level domains. While we could show that the Web of Trust does exhibit a community structure, the amount of key-related information is generally not sufficient to draw strong conclusions. We also investigated whether keys in a community have signing dates that are within short time spans (one month), which would hint at social events during which signatures were created. One algorithm, COPRA, indicated that this was the case for about 40% of communities.

Cryptography With respect to cryptography, we did not find much reason for concern in the LSCC, at least for the time that our data set is from: most hash algorithms were from the SHA family, with MD5 used in only 10% of signatures. Key sizes were sufficient—sizes of less than 1024 bits were extremely rare. However, these findings need to be evaluated again in the light of new findings since 2009. 1024-bit keys are not recommended any more as their security margin has become very small.

Historical development We investigated the growth of the Web of Trust over the years. We found that it experienced a significant growth after the year 2000, and was still growing at the time we created our data set, although at a slower rate. Earlier work analysed a network that was markedly different from the one we investigated.

The conclusion we draw from analysing our data is that the Web of Trust is indeed a very useful construction, although its usefulness is limited primarily to verifying keys in the 'neighbourhoods' that are only a short hop count away (in terms of signatures). The main factor working against usefulness is the low outdegree of keys. Unfortunately, introducer trust is not a property that is stored on key servers, and thus our findings describe the best case of usefulness in the Web of Trust. In any case, users should give and receive enough signatures so they can verify keys over at least three redundant paths. This would relieve them from the requirement of having 'full' introducer trust into every key on a verification path.

5.7. Statement on author's contributions

This chapter is an extended version of the following paper: A. Ulrich, R. Holz, P. Hauck, G. Carle. Investigating the OpenPGP Web of Trust. *Proc. 16th European Symposium on Research in Computer Security (ESORICS)*, Leuven, Belgium, September 2011 (reference [73]). The publication is based on the results of Alexander Ulrich's intermediate thesis that the author initiated and advised: A. Ulrich. Analyse des OpenPGP Web of Trust. Studienarbeit, Universität Tübingen, Fakultät für Informations- und Kognitionswissenschaften, Tübingen, Germany, June 2010 (reference [72]).

The author of this thesis initiated and guided the Web of Trust project. Alexander Ulrich obtained all empirical results. For the chapter, the author added the results on the history of the Web of Trust, which are also from [72], as well as more background on GPG's default trust metric, to which Alexander Ulrich provided significant input. The author also reassessed the cryptographic algorithms in the light of cryptographic developments since 2010. The paper was written entirely by the author; Alexander Ulrich created the plots and edited the paper during the shepherding phase.

The following sections are adapted from the paper. All modifications were made by the author of this thesis. Section 5.1 is an adapted version of sections 1 and 2 in the paper, extended with an introduction of the investigated questions. Section 5.2 combines section 3 and the appendix from the paper, with only minor stylistic changes made by the author. Section 5.3 combines sections 4–6 of the paper. The author added the results concerning the history of the Web of Trust. He added clarifications and made some stylistic changes. Section 5.4 is an extended version of the section on related work in the paper, with very early related work on the Web of Trust added. Section 5.5 is from the paper (section 8), with only stylistic changes made.

Chapter 6. Analysis of the PKI for SSH

This chapter is an extended version of our previous publication [28]. Please see the note at the end of the chapter for further information.

In this chapter, we address Research Objective O2.3: we analyse and assess the deployment of the SSH infrastructure. As explained in Chapter 2, SSH uses a 'False PKI': in its primary mode of operation, there are no TTPs and no CAs. Instead, administrators distribute host keys to their users out-of-band or rely on trust-on-first-use authentication. This is largely due to SSH's widespread use as a protocol for remote (shell) access. In contrast to X.509, where the purpose is world-wide authentication of hosts, an analysis of this 'False PKI' has to focus on the effects of network management practices, which are reflected in key generation and key deployment. Properties of public keys and their deployment characteristics are at the focus of this investigation.

The SSH PKI is accessible to active scanning: when a client carries out the handshake, it receives the server's host key. However, SSH is a particularly security-relevant protocol as it allows access to protected resources. Thus, certain ethical considerations must be taken into account when scanning SSH.

In the context of SSH, the algorithm DSA is commonly referred to by the name of the corresponding NIST standard, DSS. In this chapter, we follow this use.

6.1. Investigated questions

We gave a short introduction to SSH and its PKI model in Section 2.6. In the following, we motivate and outline our research questions. To find answers to these questions, we carried out three Internet-wide scans and combined the results of our scans with data sets from further measurements.

Protocol versions Two versions of SSH exist. The older version 1 is known to contain cryptographic flaws and should not be used. SSH in version 2 is a complete redesign. It added support for Perfect Forward Secrecy (PFS) by introducing a Diffie-Hellman key exchange as part of the connection establishment. Administrators should enable only SSH 2. We investigated which versions of SSH are in use.

SSH servers Several implementations of SSH servers exist. From time to time, new vulnerabilities become known. It is important to keep servers up to date. We investigated which SSH servers are in use.

Host keys SSH uses public keys (the host keys) to identify a host. There are several important properties that such keys must fulfil. In the following, we give a list of properties we chose to investigate.

Strength of host keys Host keys can be RSA or DSS keys (including elliptic curve versions), and they must be of sufficient length to be cryptographically secure. We decided to investigate the distribution of key lengths and key types.

Cryptographically weak keys The security of keys can be compromised despite them being of sufficient length. Examples are the Debian bug [175], which we already discussed in Chapter 4, but also weaknesses that are introduced due to poor entropy on the system generating a key. Heninger *et al.* analysed both phenomena in their study of 2012 [35]. The authors scanned TLS and SSH servers Internet-wide and downloaded a total of about six million SSH host keys. The primary focus of their work was on determining cryptographically weak keys and the reasons they existed. They could show that pseudo-random number generators were often the cause: 0.03% of RSA keys and about 1% of DSS keys were vulnerable due to poor random number generation. The results by Heninger *et al.* pointed at a problem in many embedded devices: they have too little entropy for cryptographic operations. The work by Heninger *et al.* is our point of reference here: we set ourselves the goal to reproduce their results and determine whether the quality of keys has improved in the year since their publication.

Duplicate keys Heninger *et al.* also found a number of keys that were *not* cryptographically weak yet occurred on many IP addresses. They were able to identify some of these keys as default keys deployed as part of the manufacturing process. The authors did not investigate this more deeply, however, and stated that *'the lack of uniquely identifying host information [prevented them] from distinguishing default keys from keys generated with insufficient entropy'* [35]. We decided to investigate these keys in more detail, for two reasons. First, while default keys in devices may be one cause for the phenomenon, it is necessary to determine *where* these devices occur (in which networks, or in which geographic locations) to estimate the potential impact on security. Second, some keys may be the consequence of intentional administrative decisions, like SSH gateways, or setups in certain shared hosting environments. These are relevant network management practices, and it is an interesting question whether the corresponding keys are used safely or not. Of course, some keys might simply be reused out of convenience and deployed on hosts within the same administrative domain. The consequence in this case would be a multiplication of the attack surface.

SSHFP SSH can make use of the SSHFP resource record in DNS. This allows to store the fingerprint (a hash value) of a host key in the DNS. We decided to investigate to which degree this is deployed, whether the information in the DNS is accurate, and whether it is secured by DNSSEC.

Cryptography Finally, SSH deployment should choose secure ciphers, symmetric session keys of sufficient length, and secure hash algorithms. We investigated this.

6.2. Scanning process and data sets

In the following, we describe our approach to obtaining the data sets and document our scanner and our scanning runs. We enriched our data sets with data obtained from other active scans and databases, specifically DNS scans using a fast custom scanner, AS and WHOIS lookups, as well as a geo-IP database.

6.2.1. Scanner

We scanned from a single IP address from our own AS, AS56357. Our scanner consists of three components: an IP address generator, a port scanner and the actual SSH scanner. We implement a producer-consumer model. The IP generator is responsible for producing IP addresses that are uniformly distributed over the IPv4 space in order to minimise the impact for networks with contiguous IP ranges. Our method of choice is a Linear Congruential Generator (LCG). Our LCG is constructed in the same way that Leonard and Loguinov [48] described. The LCG executes the function $X_{n+1} = (a \cdot X_n + c) \pmod{m}$. Our values for a, c and m are a=1,664,525, c = 1,013,904,223 and $m = 2^{32}$, following a suggestion by Knuth and Lewis in [88]. The LCG generates pseudo-random numbers that are uniformly distributed over the output domain and are guaranteed to never repeat. We interpret these numbers as IP addresses. Our construction also allows us to pause and restart the scanner at will, by using the last output as a seed for the restarted run.

Before we feed the IP addresses to our port scanner, we first compare them against a blacklist, which consists of two parts. The first, largest part contains IP addresses (prefixes) that are not announced via BGP and thus not routable. We determined these prefixes by collecting all announced prefixes on the border routers of our AS and computing the complement. The second part of our blacklist consists of IP ranges whose owners asked us to exclude them from our scans. At the time of writing, the latter part contains only about one million IP addresses. We elaborate on this in more detail in Section 6.4. IP addresses that are not on the blacklist are passed on to the port scanner. This scanner consists of an array of wrappers around `nmap` instances that probe TCP port 22.

Only IP addresses for which this port is found open are passed on to the actual SSH scanner. This scanner consists of threads of modified instances of OpenSSH. Each instance attempts to carry out a full handshake with the remote party and, if successful, stores the full authentication information of the handshake, including the remote host's host key (this would normally not be part of OpenSSH's output, thus our modification).

6.2.2. Scanning periods and data sets

We carried out three scans: in January, April and July 2013. Each scan took about 5-12 days. The resulting data set is presented in Table 6.1, with the number of hosts with which we could carry out SSH handshakes, and the keys we downloaded. Naturally, the difficulty in giving a precise account of hosts lies in the fact that the equation 'distinct IP address = distinct host' does not hold. Hosts may have several interfaces, and interfaces may also be assigned more than one IP address. For simplicity, we still use the term 'host', but it should be kept in mind that when we give numbers for hosts, these are *upper* bounds.

We found SSH servers in 32,000 ASes in each scan. This is interesting as the current estimate of the Internet's size is around 40,000 ASes, meaning that a quarter of ASes either block public SSH access, do not use SSH, are not announced, or are on our blacklist.

The number of IPs with SSH servers decreased from scan to scan. One confirmed reason, although likely the less dominating factor, is that our blacklist grew over time. One large hosting provider asked for the blacklisting of 830,000 addresses. The other factor, although plausible, is harder to confirm: systems blacklisted our IP address, at least for port 22, despite the measures we describe in Section 6.4.

The number of IP addresses for which we could carry out the SSH handshake, and which occur in both the January and April scans, was 6 millions; and those that occur in

Scan	Jan 2013	Apr 2013	Jul 2013
Scanning period	5–12 January	18–29 April	23–27 July
IPs with SSH	12.0M	10.2M	8.8M
Keys	21.9 M (9.8 M)	18.5M (8.7M)	16.0M (7.4 M)
SSH 2			
...RSA	10.9M (4.6M)	9.2M (4.0M)	8.0M (3.6M)
...DSS	9.3M (4.3M)	7.8M (3.7M)	6.7M (3.3M)
...NIST-P256	734k (450k)	758k (493k)	780k (519k)
...NIST-P384	54 (35)	48 (34)	45 (31)
...NIST-P521	696 (556)	655 (517)	734 (529)
SSH 1			
...RSA	984k (538k)	721k (417k)	528k (310k)

Table 6.1. – Data sets from our scans. Numbers in brackets indicate distinct keys. NIST-x indicates an elliptic curve algorithm. Note that SSH 1 uses only RSA keys.

both the April and July scans was 4 millions. About 2.8 million IP addresses appeared in all three scans, i.e., between 24–32% of the IP addresses in the respective scans. AS reassignments were not responsible for this: only 92,000 IP addresses changed their AS during the scanning period. However, we know from [34] and [78] that a large portion of the IPv4 space is dynamically assigned and IP occupancy often shorter than 24 hours. This suggests that many SSH installations are actually found on dynamic IP addresses. This is in line with our findings concerning broadband access devices (see Section 6.3).

6.2.3. Enriching data sets

We used data from other sources to enrich our data sets where needed. These were DNS lookups, AS lookups, WHOIS lookups, and finally geographic location.

DNS querying Our tool of choice was `dns-scraper` by Ondrej Mikle [225]. This tool is capable of querying several million records in a relatively short time (several millions per day). It uses its own resolver and is able to perform DNSSEC validation. We used it to query the PTR records for IP addresses and also in our investigation of domains with SSHFP records. We elaborate on this in Section 6.3.

AS lookup We used `pyasn` [138] to determine the number of the AS to which an IP address belonged. This tool can carry out historical lookups using the Route Views archive [131]. We used one archive per scan to lookup the IPs in that scan, namely the archive that was the most recent one just before the scan started.

WHOIS We used Team Cymru's WHOIS service [264] to carry out WHOIS lookups for ASes, based on their numbers.

Geolocation Finally, we mapped IP addresses to country and city using the Maxmind geo-IP databases [216].

6.3. Results

We present the results we obtained from our scans to answer the research questions we formulated in Section 6.1.

6.3.1. Protocol versions

SSH servers may support SSH 2, SSH 1, or both. They indicate this in the first message they send. Support for older versions is either indicated by sending a string of the form 1.x, which means support for SSH 1 only, with x between 3 and 51, or of the form 1.99 (to indicate support for both SSH 1 and SSH 2). As mentioned, it is now agreed that SSH 1 should not be used due to several fatal weaknesses that, for example, allow an attacker to inject text into the cipher text without a receiver being able to detect this. Given that all major SSH clients today support SSH 2, there is really no reason to leave SSH 1 enabled. To the best of our knowledge, major Linux distributions also ship with SSH 1 disabled by default. We were thus interested which protocol versions servers offered.

We found all three groups in our scans. In all three scans, more than 90% of all SSH servers supported only SSH 2, which is a good finding. The share of hosts that support both SSH 1 and SSH 2 is decreasing: between January and July 2013, it fell from 7.9% to 6.2%, while the share of SSH 2-only hosts increased by 2%. The use of SSH 1 (0.5% in July) could be said to be almost negligible—but this percentage corresponds to an absolute value of 42,000 hosts. We believe these are probably older systems. Overall, however, this distribution can be viewed as very good. If any recommendation can be given at all, then it should be to disable SSH 1 for good on clients and servers.

6.3.2. Server versions

The server string sent to the client reveals which SSH implementations from which vendors are used on server side. To some degree, this allows to identify older, non-updated SSH servers. Updates are an important process to remove vulnerabilities, and proper network management should take care of this. There is an important caveat to take into account here: since our scanner executes the SSH protocol faithfully, we can only report the server string as sent to us. However, SSH servers may have been (manually) patched without this being reflected in their server strings, or they may intentionally mislead the client (e. g., to obscure the attack surface).

Figure 6.1 shows the most frequently used SSH servers and the development of their share over time. For better visualisation, we grouped OpenSSH versions with the suffix *p* together with the original version. The *p* identifies ports to non-OpenBSD systems. We find that OpenSSH is by far the most popular SSH server, running on about 65% of hosts. Dropbear is used on 20% of the scanned hosts. Other servers occurred much less frequently. Worth mentioning are Cisco implementations (4%) and ROSSSH, a MikroTik implementation on about 2% of hosts. All other servers occurred on less than 1% of hosts.

Some SSH servers use custom vendor identifiers. The variance is high: in our April scan, we found more than 40,000 vendor strings that occurred at most five times. However, almost all of these were either random-looking or Base64-like strings. In some cases, we also found what looked like hex-encoded characters—possibly signs that these strings were chosen intentionally. We also found strings that very definitely described the patch level[1]. The random-seeming strings point at some kind of creation

[1]E.g., `OpenSSH_4.3p2-1.9_test4.cern-hpn-CERN-4.3p2-1.9_test4.cern`.

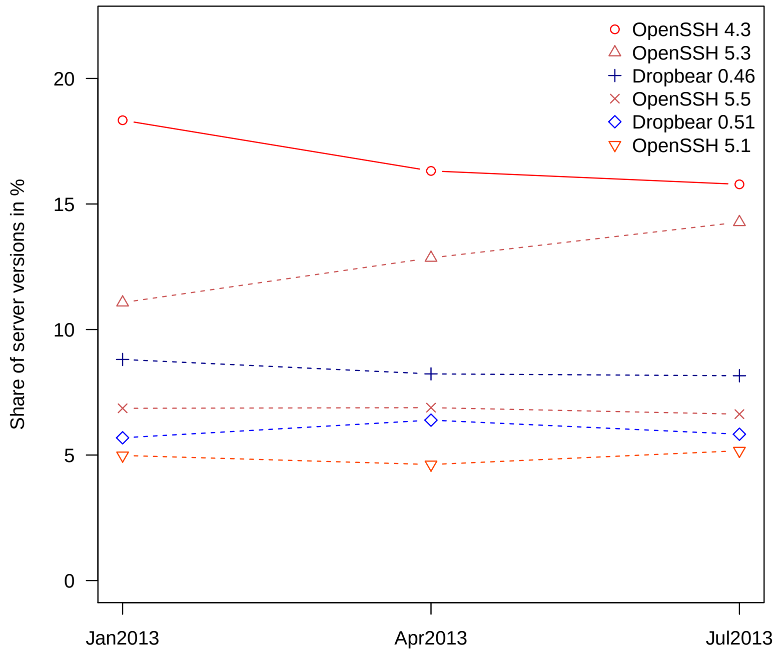


Figure 6.1. – Most frequently encountered SSH server versions over all scans.

Server version	Published in	Known CVEs
OpenSSH 4.3	2/2006	4
Dropbear 0.46	7/2005	3
OpenSSH 5.3	10/2009	2
Dropbear 0.52	11/2008	1

Table 6.2. – Selection of server versions that occurred in our scans for which CVEs have been published.

mechanism during install or compilation time, although their exact purpose remains unknown to us.

Concerning the age of servers, we found that OpenSSH 4.3 dominates the SSH landscape, even though it is from early 2006. There are several known vulnerabilities for this version. Fortunately, we can also see that the use of OpenSSH 4.3 is declining (but more than 1.5 million servers remain). Only 6.6% of OpenSSH servers in our July 2013 scan have a version number of 6.0 or above. OpenSSH 6.0 was released in April 2012. The latest OpenSSH version at the time of writing, 6.2, was published in March 2013. It appears in only 1% of version strings in our scan of July. Similarly, only 6.5% of Dropbear servers are more recent than February 2012 (version 2012.55). Whatever one's view on the rate of patched vs. unpatched systems may be, we find it intriguing that the adoption of new SSH server versions seems to be extremely slow.

Although we cannot establish directly whether a server implementation was patched or not, we do wish to state that vulnerabilities are known for several of the version strings that we encountered. Using [173] as a vulnerability database, we compiled Table 6.2. It shows release dates and number of CVEs[2] for some server versions we encountered in our scans. Determining whether a server reporting a vulnerable version is patched or not would require, at the very least, very intrusive scans. We consider such

[2]CVE is short for *Common Vulnerabilities and Exposures*. The CVE system constitutes a unified framework to report vulnerabilities.

Heninger *et al.*	**Jan 2013**	**Apr 2013**	**Jul 2013**
0.03%	0.016%	0.013%	0.014%

Table 6.3. – Co-prime weak RSA keys across all hosts.

scans unethical. We thus cannot provide further evidence here, and restrict ourselves to the statement that it is at least possible that a fraction of these servers was not patched. We view our judgement as supported by the overall slow rate of change that we generally observe for server-side software—both for SSH as well as previously for TLS (also see Chapter 4).

6.3.3. Weak keys

We investigated how many SSH servers presented RSA keys that must be considered cryptographically weak due to irregularities during key generation. In doing so, we reproduced the results by Heninger *et al.* and determined to which degree the situation has improved since the authors' disclosure.

Co-prime weakness

Heninger *et al.* showed in [35] that a number of RSA public keys exist whose moduli share one prime number with the modulus of a different public key. In such cases, the private keys can be calculated very fast. The authors reported that about 0.03% of RSA keys had this problem. They initiated a responsible disclosure process, too. We used the original authors' implementation [198] to determine how many co-prime vulnerable keys still exist. Table 6.3 summarises our results. While the numbers seem to have fallen in the year that passed, they also do not seem to change any more—the changes in the last two scans are likely due to measurement inaccuracy and rounding effects. With numbers so low, it is hard to tell if this is a result of the responsible disclosure by Heninger *et al.*—normal routine updates may also be involved. Furthermore, the numbers are so low now in any case that it is becoming difficult to attribute changes to updates of any kind rather than measurement inaccuracies.

We were surprised by one effect: when we filtered for SSH 1 keys, we found that about 2.4% of them had a co-prime weakness, and the number is not decreasing. This hints that entropy problems were present from the beginning of deployment. Furthermore, it may be an indication that the very old systems do not receive attention any more. As SSH 1 should not be used anyway, it does not constitute a major weakness, however.

Debian-weak keys

A second well-known vulnerability is the use of SSH keys created on Debian-based Linux systems with a version of OpenSSH that contained a serious bug in the algorithm for key generation [175]. These flawed versions were distributed from 2006 to 2008. Their public-private key pairs can be precomputed with reasonable effort. H. D. Moore published a list [227] of known-weak keys which we used, too. The good news is that, five years after the bug was found, Debian-weak keys are rare and becoming even less common, thus continuing the trends documented in [37, 35]. In January, 0.017% of SSH 1 keys were vulnerable, and 0.077% of SSH 2 keys. In July, the numbers had decreased to 0.011% and 0.063%, respectively. The fact that more SSH 2 keys seem to be affected may hint that the majority of SSH 1 keys were generated before 2006 and these servers do not receive updates.

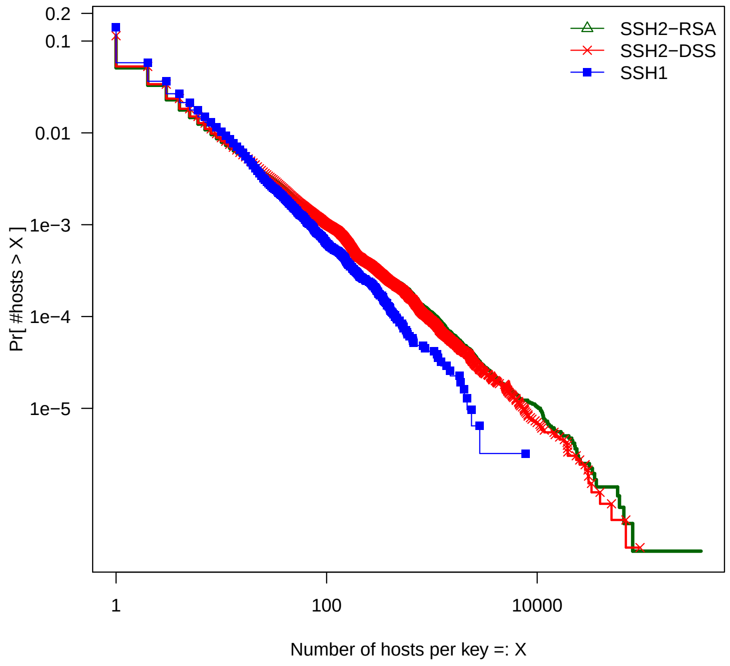


Figure 6.2. – CCDF: frequency of case 'key occurs on more than X hosts' (for July 2013).

	Jan 2013	**Apr 2013**	**Jul 2013**
SSH-1.x	530 k (54.16%)	364 k (50.70%)	261 k (49.51%)
SSH-2.0	12.7 M (61.03%)	10.5 M (59.02%)	8.9 M (57.46%)

Table 6.4. – Duplicate host keys fetched during the scans.

6.3.4. Duplicate non-weak keys

More than half of all fetched host keys were used on multiple IP addresses. This phenomenon was also reported by Heninger *et al.*, who distinguished weak keys on devices with poor entropy (see above), devices with default keys, and simple reuse of the same key on different hosts. We were interested in the latter two phenomena as they relate to network management practices. We investigated which of the two phenomena occurs in which networks, and to which degree it constitutes a security weakness. Our analysis was mostly carried out on the data set from July 2013, with the exception of one AS.

The first thing to note is that duplicate *yet non-weak* keys occur much more frequently than cryptographically weak keys. Ranking their occurrences together, the first weak key appeared on rank 1337 in our data set. Figure 6.2 visualises how commonly the case 'a key is found on more than X hosts' occurs over the entire IPv4 space. Table 6.4 shows the number of non-unique keys for all three scans.

In our study, we investigated the 10 most frequent duplicated keys, which together account for about 6% of all keys. We also added one particularly interesting case that we could identify with the help of an ISP. We first enriched our data set with the DNS PTR resource record for all IPs in our data set. This yielded 6.2 million names, of which 5.8 million were distinct. Second, we determined the operators of ASes and their registered

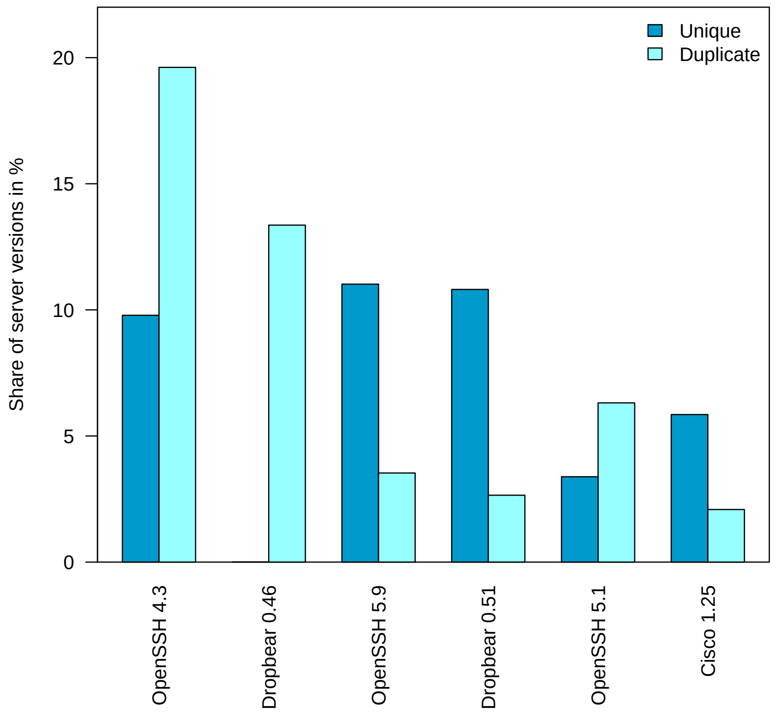


Figure 6.3. – Market share of servers with unique and duplicate keys.

location on the globe. Third, we used the Maxmind geo-IP database to retrieve country and city of an IP address. Fourth, for the cases of mixed hosting/broadband access, we also carried out `nmap` service scans. One fact to take into account here was that most IP addresses we investigated were dynamic, and some changed on a daily basis. As service scans on several ports take longer, it was not possible to match IP addresses unambiguously to SSH keys *and* running services. Hence, we used the service scans primarily to gain a picture how the AS was used.

General findings Figure 6.3 relates duplicate keys to the market share of server versions. Some servers, such as OpenSSH 4.3, are responsible for a large portion of duplicate keys. Dropbear 0.46 distributes almost exclusively duplicate keys. It is known that Dropbear is often used in embedded systems, so this points at devices with default keys. However, it is interesting that Dropbear 0.51 offers unique keys more commonly, which could mean it is primarily used in different environments. For the following analysis, we group by probable cause of key duplication.

Mixed hosting/broadband AS Among the top 10 keys, we found a pattern for key duplicates in ASes where we found IPs to be used for both (Web) hosting and broadband access devices. For example, the most common key in our data set was an RSA key found primarily in just ten ASes; i.e., more than 99% of 360,000 occurrences in July, and 560,000 in January, were in these ten ASes. The only feature that all these hosts seemed to share was that they employed Dropbear 0.46. This particular key had already been found by Heninger *et al.*, although only 240,000 times. The key was of particular interest because 80% of occurrences were found in the two ASes of the same Taiwanese ISP (which we will call *TW1* and *TW2*), particularly in Taipei, and another 16% in that of a mainland Chinese ISP (*CN*). The rest were distributed over Central America, the Caribbean and the US. This geographic pattern was the same across all scans. Almost all (99.5%) of DNS names in Taiwan and China contained the string 'dynamic', which already pointed at use for broadband access devices. Thanks to the database created by the authors of [35], we learned that they had identified the key as a default key in a DSL device, too. It seems plausible that it is a device sold primarily in the above mentioned markets, at least with this key. The AS in question did not only present this key, however—we found a large number of other keys. Figure 6.4 shows the distribution for *TW1*. The security finding here is that the use of a shared key among so many different systems is a weakness as it cannot be safely used for purposes like remote administration. We found two other extremely common keys in *TW1* (ranks 5 and 6; 66,000 and 60,000 times, respectively). We informed the ISPs, but did not receive a reply.

(Mostly) safe broadband access devices We found a similar device as above for the key ranked number 9 in our database. The corresponding IP addresses were almost exclusively in Singapore and belonged to a broadband provider (*SG*). The PTR records were unique strings in that provider's domain. The SSH server was overwhelmingly Dropbear 0.52 (99%). Again, this seemed to be a case of a default key in a device for broadband access. The situation was better here than above, however: all bar 38 devices were within this AS. However, we did find the other 38 occurrences as far away as New Zealand. We can only speculate how the 'escaped keys' got there—possibly by simple labour relocation. Figure 6.4 shows the distribution for AS *SG*—once again, it seems to be used for multiple purposes as there is a large number of other keys, some of which are even unique.

Home entertainment devices We encountered an entirely different case of key reuse for an RSA and a DSS key that occurred 58,000 times and 50,000 times in our scans (ranks 7 and 8), respectively—but distributed across well over 1500 ASes in more than 50 countries. More than 80% of the IP addresses were in the US or a closely related market like the UK, Canada, or Australia, however. The SSH server revealed the solution as it contained a vendor-specific string that belongs to a home entertainment solution by a well-known and globally acting corporation. One of their products is a fully configured 'off-the-shelf' Linux-based entertainment server solution. We verified this by scanning port 80, which revealed itself to use a Red Hat-based Apache version, which this particular vendor also advertised as a feature of their product. Connecting to several of the IP addresses with a browser, we confirmed the brand. Additional credibility was lent to this as in 38% of cases this key occurred in an AS where the WHOIS entry contained the term 'cable'—this type of broadband access is fast. Needless to say, this kind of setup is extremely dangerous in terms of security.

SSH gateways We encountered an entirely different setup for the second most common key in our database, found in the AS of a large German ISP (*DE*, 95,000 occurrences). The key was strictly limited to this one AS. It occurs on 75% of hosts that we found in that AS, although many different keys are used there (see Figure 6.4). The next most frequent key in that AS occurred only 41 times. Resolving the IPs to host names (PTR record), we found that a third of them pointed to just one domain; the rest resolved mostly to different domains. The SSH server was always a modified Dropbear version and the service scans showed that only the Apache Web server was used. This seemed to point at a hosting situation with a very different setup. Two plausible reasons were that we were either facing an SSH gateway or that virtual machine images with the same key had been deployed. The former would be a good solution, the latter a very poor one. Unfortunately, these setups are hard to distinguish without very intrusive scans like OS scans. We contacted the ISP; they confirmed we had identified 'shared hosting' products, and that SSH traffic is directed via a gateway. This allows

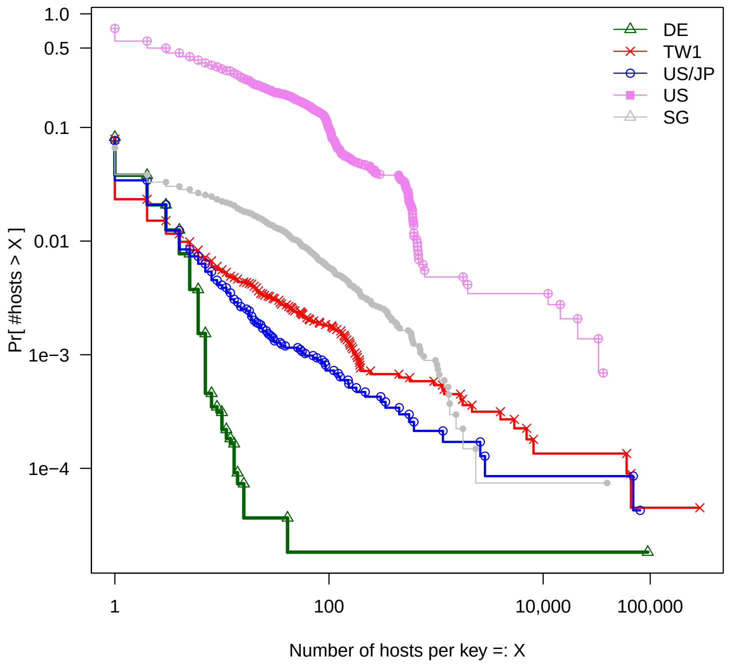


Figure 6.4. – CCDF: 'key occurs on more than *X* hosts', for selected AS.

simpler management and enables them to switch SSH keys on the real machines without customers encountering SSH host key warnings. The private key is never exposed to a customer. When done properly, as it seemed to be the case here, this SSH setup seems quite reasonable.

We found what seemed to be similar cases of 'key clusters' (ranks 3 and 4) in the AS of a Japanese corporation's branch in the US (*US/JP*), where two keys represented 80% of fingerprints. We also found a cluster in a US AS (*US*, rank 10). The variance in these networks concerning SSH servers and Web servers was higher, however. We contacted the ISPs, but did not receive an answer. Their AS setup may be a mix of the ones in Germany and the ones in Taiwan and China.

Per-customer keys An AS of interest was that of another large German company, where we found about 60,000 keys that occurred more than once in their AS, but no single key was used more than 200 times. These keys did not occur in our top 10, but their existence pointed at an entirely different management practice. Our suspicion was that virtual machine images might be distributed to customers without regenerating the host keys. We contacted the ISP. They confirmed that virtual machines were used, but that SSH keys are generated at deployment time. They investigated and offered a plausible explanation: their customers are allowed to bind several IP addresses to an interface. In such cases, the host key would be the same for all interfaces, and we would have mistaken distinct IP addresses for distinct machines (note our caveat from Section 6.2). However, they could also confirm that there were some cases where customers did indeed reuse keys across *different machines*. Note that the type of hosting is entirely different here, with the responsibility of key management on the side of the customer. This latter case of reuse is a potential vulnerability, especially if customers run public-facing services with attack surface.

6.3.5. Use of SSHFP

A way to make SSH connections more secure, especially initial connections, is to use the DNS to store the fingerprint (hash value) of an SSH key in a dedicated DNS resource record, SSHFP. This is defined in RFCs 4255 and 6594 [120, 121]. DNSSEC can be used to sign these records. We determined whether such records are in use and accurate. We looked up the PTR records of IPs with active SSH servers, and then forward-resolved the domain name obtained thereby. We restricted our analysis to such IP/domain combinations where the two matched. Thus, our results constitute a lower bound. We found 2070 distinct domains employing a total of 4374 SSHFP records (a domain may have multiple records, one for every key type it uses). We considered a domain secured with SSHFP if at least one SSHFP resource record matched a key. We found 94% of the 2070 domains were correctly secured with SSHFP records. However, only 660 of these also had secured their SSHFP records correctly with DNSSEC. The rest transmitted the SSHFP record via insecure DNS.

Note that the new TLSA resource record in DANE [105], which we also discuss in Section 8.3, is intended to reproduce the functionality of SSHFP for X.509 certificates meant to be used with TLS. It is an interesting debate whether our result allows to draw conclusions with respect to TLSA. A relatively low number of domains deploy the SSHFP record. It is plausible to assume that those operated by technology aficionados (e. g., open source projects) are more likely to do so. On the one hand, one could argue that 94% correctly deployed records are a good value. On the other hand, 6% wrong records can be taken as an early warning that deploying security-relevant resource records must not be taken lightly. Sites considering such a move should definitely have their DNS and TLS administration tightly integrated.

6.3.6. Cryptographic algorithms

We investigated which cryptographic algorithms were advertised by servers. Note that servers usually advertise an entire list of algorithms, not just one. Concerning symmetric ciphers, we found 111 different ciphers sent by servers. In April 2013, the most frequently supported cipher was 3DES, offered by 99.5% of hosts. AES offers in CTR mode, with a key length of 128 bits or more, summed up to 69.9%; in CBC mode the number was 87.7%. RC4 was also offered frequently (67.1%). Despite recent progress in security analysis on CBC mode [3] and RC4 [144], their usage can currently be regarded as safe. However, very few servers also offered weak ciphers such as DES or RC2 (0.03%). The numbers for July were almost the same.

All servers advertised support for HMAC for integrity protection. We also found the less CPU-intensive yet strong UMAC in 40% of advertised algorithms in the first scan. This rose to 45% in the last scan.

Regarding key exchange mechanisms, the RFCs for SSH mandate support for two forms of Diffie-Hellman key exchange. One is `diffie-hellman-group1-sha1`, the other is `diffie-hellman-group14-sha1`. The former is supported by almost all hosts. However, we found that, among all scans, a non-negligible fraction of servers did not implement the latter. The number decreased from 32% in January 2013 to less than 29% by the end of July 2013. It is known that some Dropbear implementations and very old OpenSSH servers do not provide both methods. The reduction can thus be explained by the changes in market share. This finding shows again that old SSH software is still widespread.

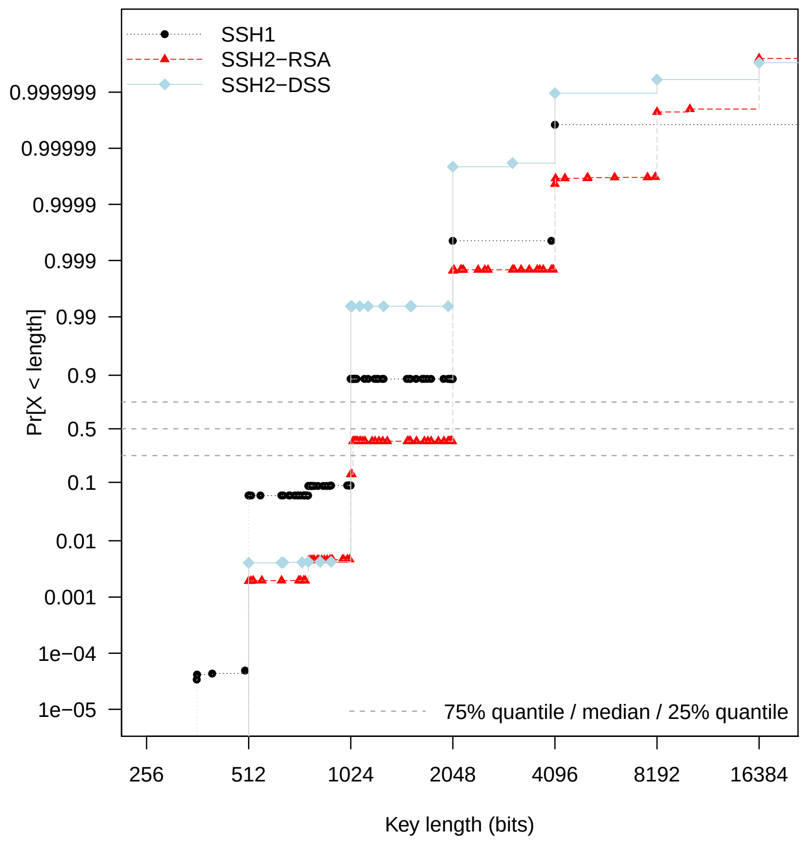


Figure 6.5. – Cumulative distribution of key lengths in July 2013. (Note the unusual atanh scaling (double-ended pseudo-log) of the y axis.)

6.3.7. Key lengths

We investigated the lengths of host keys for SSH 1 and SSH 2. Figure 6.5 shows the distribution for the different key types in July 2013. Concerning SSH 1, we find that about 9% of all keys are shorter than 1024 bits. These key lengths are considered too short by today's standards—RSA at 768 bits was factored in 2009, and NIST recommends at least 2048-bit keys [142]. However, our finding must be viewed against the background of the relatively rare occurrence of SSH 1 and its other flaws. Furthermore, as the plot shows, most remaining keys have 1024 bits or more. Concerning the more important SSH 2, we see an entirely different distribution. Keys with a length of less than 1024 bits make up for only about 5%. Keys with a length of 1024 bits make up for more than 50%, and those with a length between 1024 bits and 2048 bits add another 44%. Longer keys are much rarer, although there are a few examples of 16,384-bit keys, which can be regarded as extremely conservative from a security perspective. We did not find any major changes in the key length distribution over the course of our scans. We also compared our numbers to the findings by Heninger *et al.* The authors had reported an occurrence of 0.08% of 512-bit SSH 2 keys. In April 2013, we found 0.3% of keys to have this length. This means there are about 50,000 hosts with 512-bit keys.

6.4. Ethical considerations

Our scans for SSH are different from our TLS scans in three respects. First, we scanned the entire (routable) IPv4 space—the number of IP addresses we scanned ranged in the billions, compared to just millions for TLS. Second, to achieve this, we needed to send our probes at a much faster rate. Third, SSH is a protocol that is generally used to access *protected* resources—TLS for HTTPS is more often used to authenticate and encrypt otherwise *public* content. Thus, scanning SSH is a much more sensitive activity. In the following, we discuss the two issues of carrying out scans in a responsible way and publishing the resulting data sets.

6.4.1. Responsible scanning

Scanning SSH on Internet scale means to probe highly sensitive ports so fast that an AS is likely to be hit several times during a day. This means our scans trigger certain Intrusion Detection Systems, which in turn may cause unnecessary work for administrators. Hence, the first step is to minimise negative effects.

To this end, we implemented several cautionary measures during our scans. As a first precaution, we notified about 20 CERTs, watch list services like the Internet Storm Center [205], and several block list operators before our scans. As we had to carry out full handshakes, we sought for a way to label our scans as less threatening. First, we supplied contact information in the SSH user name[3]. Second, we changed the authentication method to a non-existent one[4] and added additional information with a URL in the comment section of the SSH server string.

As a consequence of our scans, we received a number of complaints by email. We replied to *every* such email in person, stating our intentions, offering to blacklist the affected system and providing links to further information. Over the course of all three scans, we received a total of 191 abuse emails. About 80% of these were automated complaints sent by intrusion detection systems, firewalls or filtering software such as `fail2ban`. On ten occasions, friendly conversations with administrators followed our reply—mostly, these were requests for further information. We received only one email mentioning possible legal consequences for us. We replied to this with a second, longer explanatory email and blacklisted the affected system. This seemed to resolve the issue as we did not receive any further replies.

It is vital to maintain a blacklist of parties that do not wish to be scanned. Our blacklist now contains entries from 17 parties. Only one hosting provider requested blacklisting for their range. The total number of IP addresses requested to be blacklisted is roughly one million—about 830,000 of these requested by the mentioned provider.

With the above precautions implemented, we ultimately decided to carry on with our scans on Internet scale. The position we take is that research can contribute to a safer Internet as a whole. Furthermore, carrying out scans over a period of time has given us valuable pointers. We recommend to always notify watchlist and blocklist operators, CERTs, and previously affected parties as far as they are known.

6.4.2. Sharing data sets

We believe it is important for the scientific community to be able to reproduce our results. We encourage further analysis of our data sets, which we are going to release on our site [249]. However, to prevent abuse, certain precautions are necessary. In part, we derive the measures we take from the practices described by Allman and Paxson in [5]. Our goal is to avoid that an attacker can use sensitive information like weak keys or server version to stage attacks without having to carry out the same kind of large-scale scan that we did. Thus, we must avoid that vulnerable servers can

[3] `university-security-research-see-https-pki-net-in-tum-de`

[4] `security-research@bozen.net.in.tum.de`

be identified without interacting with them. Approaches like k-anonymity [70] and l-diversity [53] have limitations when the attacker is in possession of other data sets that he can link to ours. A better way to protect against abuse is thus to remove indications of vulnerability from the data set and to suppress IP addresses entirely, replacing them with random numbers. This still allows researchers to reproduce most of our statistical results, but prohibits potential attackers from learning more from our data set than they could learn by scanning themselves. We will also be happy to share our data sets in their entirety with other research groups if they are willing to enter an agreement with us that they must protect the data to the same standards and will not engage in misuse.

6.5. Related work

A number of previous publications have addressed large-scale scans, including scans of security-relevant protocols.

To the best of our knowledge, the first larger SSH scan was carried out by Provos and Honeyman in 2001 [62]. They presented results of an SSH scan of 2 months' duration. Their scanner generated IP addresses randomly by encrypting a counter with a block-cipher. It then proceeded to download the SSH identification string of 2 million public IP addresses and from hosts on their local university network. They found that the adoption of SSH 2 had risen to almost 50%, which was a favourable value at the time. They also matched the server identifier string to a list of vulnerable versions and found that the use of vulnerable servers in their university network had declined during the investigation period. Since then, scanners have become much faster. Our scans, for example, exceed theirs by a huge margin.

Scans of other security protocols have been carried out before. Yilek *et al.* [79] carried out scans of TLS to determine the number of weak public keys that were caused by the now-infamous Debian bug [175]. They scanned 50,000 hosts and found about 750 weak keys. The authors also found a slow rate of fixing. They did not scan SSH. In 2011, our own scans led to an analysis of X.509 certificates [37], which we documented in Chapter 4. Results in another publication by Vratonjic *et al.* [75] supported our statistics. In 2010, the Electronic Frontier Foundation presented data from a scan of TLS at non-academic venues [184, 245].

In 2012, Heninger *et al.* presented results of an Internet-wide sweep of TLS and SSH servers [35]. Their publication is an excellent example of combining measurement methods and cryptographic insights. They downloaded both X.509 certificates and SSH host keys and recovered 6.2 million unique SSH host keys. The authors' focus was on detecting weak keys. They showed that poor entropy in pseudo-random number generators may cause RSA and DSS keys to be recoverable and demonstrated this for 0.03% of all RSA keys and more than 1% of DSS keys. DSS keys are described to be more vulnerable due to the importance of good random number generators for every message. Lenstra *et al.* had conducted a similar study a few months earlier [46] with different conclusions. They studied available X.509 data and found more vulnerable RSA than DSS keys and thus concluded that the dangers of using RSA are greater than for using DSS. Heninger *et al.* argued for the opposite. They complemented their cryptographic work with active measurements of devices. One of their findings was the existence of embedded devices with poor entropy, particularly at start-up. This led to the creation of keys whose moduli share a prime number. A further finding was the existence of default keys on numerous devices. This aspect of the work by Heninger *et al.* was one of our points of departure—based on their findings, we investigated the deployment properties of SSH as a result of network management practices.

Concerning large-scale scans in general, Leonard and Loguinov reported on service discovery scans on Internet scale [48]. We use their principle of IP generation, too. In 2013, Durumeric *et al.* presented another fast scanner that computes target IP addresses using modular group arithmetic [21].

6.6. Summarising view

We carried out three Internet-wide scans of the SSH protocol. These constitute the longest observations of this protocol at this scale. Using previous work by Heninger *et al.* as our point of departure, we analysed the deployment of host keys, protocol and server versions, and the cryptography employed. We put a special focus on the phenomenon of duplicate and frequently used keys, *which cannot be explained by cryptographic weaknesses*, and related these occurrences to specific network setups.

We were able to reproduce the results by Heninger *et al.* concerning cryptographically weak keys. We found that while the number of weak keys is low and has somewhat decreased since the disclosure, improvements are now on the border of measurement inaccuracies. Among our key contributions is an analysis in which networks and ASes strong yet duplicate keys occur, and what reasons this might have. We found setups that offered varying degrees of security: we found well-kept and centralised SSH gateways, geographically restricted devices with default keys; but also such ones that are geographically wide-spread or even sold globally.

We generally found that relatively old software is still common on the Internet, and the rate of change is slow. This finding is consistent with our earlier findings for the TLS protocol (see Chapter 4). It is not possible to determine a server's patch level with our scanning method, and probably not with any scanning method that would not be considered too intrusive by certain ethical standards. However, many vulnerabilities have been known for years, and there is little excuse for not carrying out the necessary updates. Our analysis might thus serve as a warning that a number of unpatched systems exist, although it is not possible to give a definite number. On the bright side, the SSH 2 protocol seems to have mostly displaced the flawed SSH 1. Some pockets of SSH 1 use remain, however, and a sizeable fraction of servers offers SSH 1 as a second protocol, too.

We also analysed the use of the SSHFP DNS resource records and found that these were generally accurate, which gives SSH users useful feedback on the host key they are encountering. However, many records were not secured with DNSSEC, which is a weakness in this context. It is hard to draw conclusions with respect to other records holding cryptographic information.

Concerning the actual ciphers and key lengths, our results show that key lengths are sufficient in the case of SSH 2 and ciphers give practically no reason for concern. However, the situation should be closely monitored: new findings for AES-CBC [244] and RC4 [4] have already affected the security of the TLS protocol. While it is unclear whether they may be applicable to SSH as well, it is an old wisdom that attacks become only better, never worse. It is probably wise to move away from at least RC4, even in the short term.

One summarising conclusion that we offer is, as before, that cryptography is a very subordinate concern: some of the most important issues are caused by deployment practices.

6.7. Key contributions of this chapter

This chapter addressed Research Objective O2.3. We analysed the deployment of the SSH infrastructure as an example of a 'False PKI', i.e., a PKI that does not rely on TTPs. SSH keys are deployed to the clients where they are needed—a method that does not scale, but is often used for administrative purposes. Our goal in analysing the SSH 'PKI' was to determine the deployment properties of SSH as the result of network management practices. In the following, we recapitulate the key findings.

Long-term scans We carried out three Internet-wide scans over the duration of seven months. At Internet scale, these constitute the longest available observation of this protocol to date.

Protocol versions We investigated which versions of the SSH protocol are in use. We found that the vast majority of SSH servers offers SSH 2, although less than 10% allow fallback to SSH 1. SSH 1 should definitely not be used any more.

Server versions We investigated which implementations of SSH servers are in use. We found that older versions are very common, although it is not possible to tell from active scans whether these have been patched manually to remove vulnerabilities. An older version of OpenSSH from 2006 is the dominant SSH server in use. Overall, SSH servers seem to be updated rather slowly.

Strength of host keys We investigated the lengths of SSH host keys. On the whole, most keys were of sufficient length. We found that only about 5% of SSH 2 keys are not of sufficient length. A significant fraction of SSH 1 keys is of a length less than 1024 bits.

Confirmation of earlier results on weak keys We could reproduce the results of a previous publication by Heninger *et al.*, both for the co-prime weakness of RSA keys and weaknesses due to the Debian bug. We found that the numbers have fallen slightly since the authors' disclosure. This may be a sign that their disclosure process is working. However, there are so few cryptographically weak keys that future measurements will now be on the border of measurement inaccuracy.

Duplicate keys A special focus of our investigation was on reuse of SSH host keys that are *not* cryptographically weak. We found that these constitute the vast majority of duplicate keys. We investigated the use of the ten most common such keys in the context of the ASes where they occurred. We found several typical patterns. One pattern that we consider secure is the use on SSH gateways, which we could confirm in one case and have strong reason to suspect in two more cases. The remaining patterns were all insecure. In particular, we found key reuse due to SSH default keys in devices that are used in mixed-purpose networks (hosting and broadband access). In some cases, these keys occurred world-wide; in others, they were strongly restricted to one geographical area. The latter is better, but still not good for security. We found one particular key to be the default key in a globally popular home entertainment device. Another reason for duplicate keys were hosts with several IP addresses, which naturally share the same host keys. However, the German provider where this phenomenon occurred and which we contacted could confirm to us that several of their customers also reused keys across physical machines, which increases the attack surface.

SSHFP We investigated the use of SSHFP, which allows to store the fingerprint of a host key in a DNS resource record. We found this record is not used very often;

but when it is, it is mostly accurate. Unfortunately, it is mostly not secured with DNSSEC.

Cryptography Our findings concerning cryptographic algorithms are in line with our findings for TLS (see Chapter 4): although the situation should be closely monitored, the vast majority of choices is currently safe.

Ethical considerations Finally, we described our guidelines concerning ethical issues revolving around scanning for such a security-sensitive protocol. Our position is that such scans, when carried out carefully with a view to minimising impact, help improve the security of the SSH ecosystem as a whole.

Our overall conclusion is that, while the deployment of SSH may result in secure use in many cases, sound network management practices are central to improving the security of the SSH infrastructure. We found various setups: some were secure, some rather dubious, and others even dangerous. The general rate of updates of SSH servers seems to be slow—it seems advisable for server administrators to carry out updates more frequently.

6.8. Statement on author's contributions

This chapter is an extended version of the following paper: O. Gasser, R. Holz, G. Carle. A deeper understanding of SSH: results from Internet-wide scans, *Proc. 14th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Krakow, Poland, May 2014 (reference [28]). The text is adapted from the submitted version, not the camera-ready version.

The publication is a continuation of the work Oliver Gasser carried out in his Master's thesis, which the author of this thesis initiated and advised: Oliver Gasser. Understanding SSH: large-scale measurements and notary-based authentication. Master's thesis, Technische Universität München, Fakultät für Informatik, February 2013 (reference [27]). Oliver Gasser wrote the SSH scanner. The first scan described in this chapter was still carried out and partially analysed during the Master's thesis.

The author made the following contributions to the results in the paper and in the chapter. The author contributed to carrying out the scans and made significant contributions to the results from data analysis, in particular the analysis which host keys occur in certain ASes as a result of network management and the analysis of SSHFP. The author made significant textual contributions to the paper [28].

The following sections are adapted from the paper. Section 6.2 is an extended version of section IV. The author added details on the scanner and on enriching the data sets. Section 6.3.1 is a version of section V.A where the author added some background on vulnerable versions. Section 6.3.2 is an extended version of section V.B. The author added the analysis of server versions versus published CVEs and elaborated more on the Base64-like strings in server versions. Section 6.3.3–Section 6.3.7 are from the paper, with only minor stylistic changes and some clarifications made by the author. Section 6.4 is a revised version of section VI, with minor changes made by the author to explain why scanning security-sensitive protocols is problematic. Section 6.5 is a version of section III, with some stylistic changes made by the author to better embed it in the context of the thesis. Section 6.6 is an extended version of section VII. The author added a discussion of the rate of change of setups on Internet servers. The author also added a new discussion of cryptographic algorithms in the light of new developments in this area, and put both into the context of the previous findings for TLS (see Chapter 4).

Part III.

Strengthening the X.509 Public Key Infrastructure

Chapter 7. Unified notation for X.509 reinforcements

Several schemes have been proposed over roughly the last two years to reinforce the X.509 PKI. We are going to analyse these in Chapter 8 with respect to our recommendations from Chapter 3. However, these schemes differ in both goals and methods they propose, and often consist of subprotocols that have certain dependencies. In order to be able to assess their properties better and compare them to each other, we first lay some groundwork in this chapter: we define a notation to capture the essential properties of each scheme in a succinct and concise way.

7.1. Developing a notation for PKI reinforcements

In the following, we motivate our notation and document the principal design decisions.

7.1.1. Motivation and design goals

The schemes we analyse are documented and specified in very different ways. Some schemes are described in RFCs, others in (white) papers, some in both. Neither form is very accessible to analysis, however. For instance, RFCs are written for implementers. They rarely give more than a rough outline of the key ideas and focus on precise ('wire-level') descriptions of protocols. Publications like academic papers, on the other hand, serve the purpose of transporting only the key ideas. Certificate Transparency (CT) is a good example of both methods: the key ideas are described on the homepage [154], the actual protocols are defined in an RFC [109]. To obtain an overview of a scheme, a reader has to collect pieces of information from different documents.

Both forms of presentation, RFCs and papers, also share one property that is a weakness in our context: they are mostly written in prose, and where they do employ other means of visualisation, their notations are usually very different to accommodate the different purposes. This is particularly unfortunate in our context as we wish to analyse schemes in a common context. A unified representation would be very helpful here. It would certainly be possible to describe all schemes in prose, too, and structure a discussion and analysis around that. However, there are good reasons against this:

Conciseness A description in prose that includes both the necessary level of detail and is yet comprehensive would easily become very long, and a reader might find it difficult once again to collect all pieces of information that are necessary to assess a scheme.

Multiple protocols The schemes that we describe often employ more than one protocol or procedure to achieve their goals. A good representation should make it easy for a reader to identify these. It should also denote encapsulated blocks of functionality in an accessible way.

Complex interactions We will see that protocols in our schemes often show complex interactions and dependencies, which are important at different stages of a scheme. It is harder to track such dependencies through written text than following them through a notation where they are explicitly denoted.

Varying roles The participants in the schemes we describe are often the same. Yet they act in different roles not only between schemes but sometimes also within the same scheme. Consequently, they often have different responsibilities. This should be convenient for a reader to identify.

Similarities Some schemes share common structures and concepts (like data structures, internal procedures, or even other, embedded schemes). Describing the schemes in prose makes it difficult to identify these notions and to determine to which degree they are shared.

Changes to TLS Several schemes also make changes to TLS and introduce new extensions. A representation in an appropriate notation makes it easier to compare these changes to normal TLS.

For the above reasons, we chose to design a common notation in which to describe the schemes. The above motivation can be viewed as objectives to meet. However, our notation also has to bridge a certain gap. On the one hand, a scheme usually defines protocols between participants, and sometimes complex actions on internal data structures. Such factors are very relevant to security, and should be accurately reflected in the notation. Consequently, we need to aim for a high degree of formalism here. On the other hand, we need to include appropriate abstractions in our notation to keep it succinct. There are some typical cases where such abstractions are sensible:

Common tokens Many schemes operate on tokens like certificates, public keys, or nonces. Despite their core functionality being the same, these tokens can be implemented in a variety of ways (e.g., certificates with extra attributes, public keys of type RSA vs. ElGamal, etc.). To reduce complexity, we will abstract over such issues.

Common algorithms Some schemes rely on algorithms which are known to be correct and secure, but are too long to be included in a notation. A typical example would be the verification of a certificate—which is also an example of a procedure that is common to most schemes.

Non-technical processes Some schemes assume participants to execute certain processes that are of a legal or business nature. A typical example would be the certification of a domain by a CA: several of the checks that the CA applies may depend on local jurisdictions. The checks are often only documented in legal terms in CPSes.

Complex communication patterns: queries and lookups It is common for entities on the Internet to execute certain operations with the help of other entities, which may remain unknown or even hidden in some cases. DNS is a typical example: when entities query a domain name, they typically communicate with a local resolver, which then proceeds to communicate with an unknown number of other DNS servers (including caches) before it delivers the result back to the querier. It would be infeasible to include this complexity in our notation. Nor do we actually need it in the accurate description of a scheme: we only need to know the result of the lookup, not the entities involved.

All of the above elements have in common that while formal descriptions could be given in most cases (with the probable exception of legal or business processes as their descriptions can be rather vague), doing so would incur a large overhead into the notation and distract from its main purpose, namely a concise and succinct presentation of how a scheme works. We will thus accommodate the above elements with appropriate abstractions. Note that isolating elements like business processes in abstractions also make it possible to assess a scheme's security more readily by considering them to be 'black boxes' that operate either securely or insecurely. Hence, we accept a certain loss in formality in our notation as the prize for such abstractions.

Against the background of the above requirements, we define the following—more precise—design goals for our notation:

Participants The notation shall allow us to denote which, and how many, participants exist in a scheme.

Protocols The notation shall allow us to denote which protocols exist in a scheme and denote the message exchange for each protocol, including cryptographic tokens. We denote clearly which communication channels exist.

Protocol interaction The notation shall allow us to denote how protocols interact and which dependencies exist between them.

Use with TLS The notation shall allow us to denote the necessary changes to the TLS protocol that a scheme requires.

Abstractions The notation shall allow us to define appropriate abstractions for certain tokens, algorithms and procedures (in particular of a legal or business nature).

In the following, we describe the design of our notation.

7.1.2. Design elements

The design of our notation is based on the design goals listed above. In the following, we describe the concepts we employ. We show the concrete notation for each concept in Section 7.2.

Role scripts We use so-called *role scripts* in our notation. These are a concept we borrow from the input languages of protocol model checkers, e.g., HLPSL in AVISPA [140] or SPDL in Scyther [172]. Role scripts denote the activities of participants by defining events to which the participants react in a prescribed way. Our notation shares many constructions with HLPSL. We elaborate on this in Section 7.4.

Records Every token or entity in our notation is represented as a *record*. A record is a 'named container with named fields'. There are two fields we assume every record to have. The first one is *name*. This field provides a globally unique identifier by default. The second field is only used when a record represents a participant in a protocol: *state*. We explain this field in Section 7.1.4.

All records can be extended by assigning arbitrary fields to them, which may hold any value. New kinds of records can be defined as part of a scheme.

Well-known records *Well-known records* are our abstraction for the above mentioned tokens that occur frequently in schemes and protocols. Section 7.1.5 gives a list of well-known records.

Sets Records and well-known records can appear in groups, represented as unordered sets. We use mathematical notation to refer to elements in sets or in Cartesian products of sets. We write elements in Cartesian products as tuples. Note that tuples are not records—records have names and fields.

Control structures and predicate logic Participants make decisions based on tokens they send and receive in protocol message exchanges. We use control structures like *For-loops* and *If-clauses* to express decision processes. We denote operations like the selection of a subset of records or tests of existence with predicate logic.

Procedures We group functionality that is executed several times within a protocol in *procedures*. We write calls to our procedures as we know it from many programming languages, with parameters given in parentheses.

Well-known processes Abstractions for algorithms and processes of legal or business nature are called *well-known processes*. Where well-known processes are used to denote that some algorithm is employed, we give a reference where the precise definition of the algorithm can be found.

Instantiation We sometimes need to express that a record, well-known record, or map is brought into existence. A typical example would be the generation of a certificate. We express this with an operator—see Section 7.1.7.

Maps Maps are used to group records and well-known records in a way that makes them convenient to access. A Map holds an arbitrary number of key-value pairs and allows to access a value by the corresponding key. We use square brackets to describe this, e.g., *m[a]*. Additionally, we define a Map to be associated with two well-known processes, which we denote as two fields in the Map, *keys()* and *values()*. These processes yield the keys and the values stored in the Map as sets.

Note that although we do not give a formal definition here, it is easy to see that maps are syntactic sugar. In order to define a map formally, one would first define a key-value pair as a record with three fields, 'key', 'value', and 'next'. One would then define lists as nested records by having the field 'next' contain the next record in the list (one would also define a special record to indicate the end of the list). The final element would be to define a procedure to search for a record with a given key, and to define the well-known processes *keys()* and *values()*.

We also introduce a shorthand notation to indicate that a map is going to hold only certain records or well-known records. Again, we use notation from popular programming languages and indicate this with angular brackets, i.e., $\mathit{Map}\langle \ldots, \ldots \rangle$.

Channels Entities communicate over so-called *channels*. We define a channel as a record with two associated well-known processes for sending and receiving (called *send()* and *recv()*). The well-known record also contains the fields *srcname* and *dstname* to store the names of the entities it connects and the fields *local_ip* and *remote_ip* for the respective IP addresses (we abstract over IP version). The well-known processes are denoted like normal fields in the record and used like normal procedures.

Channels are always understood to be asynchronous, bidirectional, and have the properties of Dolev-Yao channels [18]. This means an attacker is allowed any action: the attacker may eavesdrop, delay, modify and also drop messages[1]. However, if entities

[1] In our analysis, we will later assume all participants can detect delayed messages. See Section 8.1 for details.

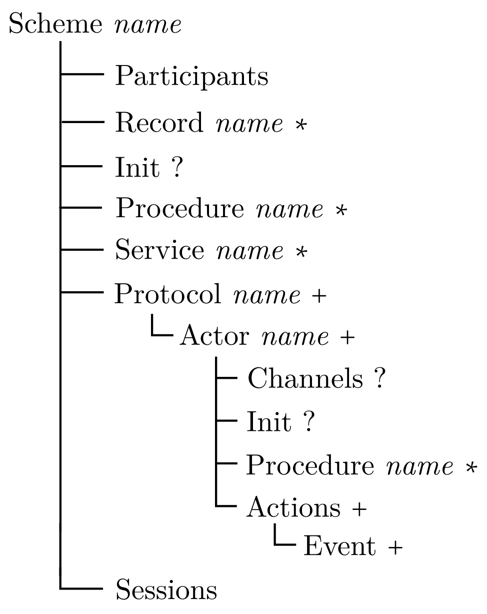


Figure 7.1. – Structure of a scheme in our notation.

send each other cryptographically protected messages, the attacker cannot break the cryptographic primitives.

Secure channels Secure channels are syntactic sugar for channels where the communicating parties are authenticated and use encryption and integrity protection. Secure channels occur in schemes where participating entities are, e.g., preconfigured with cryptographic keys.

Services We address the case of queries and lookups with so-called *services*. Services are participants in a scheme that represent an entire group of entities and abstract over the interactions between them. To other participants, services appear like normal participants, except that services have channels to every possible participant in a scheme. We give an example in Section 7.2.

Error treatment The execution of protocols and processes often leads to error states. In our notation, we commonly omit an explicit description how a participant reacts to an error. Instead, we simply state the fact and assume that the participant aborts the protocol entirely.

7.1.3. Structure of a scheme

With these design elements in mind, we are now going to explain how our notation is structured. We employ a block structure to denote a scheme. Blocks group statements that logically belong together. We use indentation to indicate which elements are contained in a block. A block ends when a new block begins or the scheme ends. Note that blocks may be nested.

Figure 7.1 shows the block structure. Some blocks may occur an arbitrary number of times or be omitted. We use the quantifier '∗' to express this. Others may occur once or not at all (quantifier '?'), or must occur at least once but may otherwise occur any number of times (quantifier '+'). A block that occurs exactly once is denoted without quantifier.

In the following, we give a description of each block.

Scheme The block that contains all other blocks is *Scheme*.

Participants This first block denotes which kinds of entities participate in a scheme. At the outset, a participant is a record that contains only the field *name*. The purpose of this block is to group participants into different sets with different names, e.g., ‘clients’ or ‘domains’, to denote a certain category of participant.

Record This block holds the definition of a new record. Records can be referred to by their name. If a name appears in a nested block (and possibly also outside the block), we assume a local scope as we know it from programming languages.

Init Entities in a scheme may be preconfigured to have certain information. For example, all CAs need cryptographic key material. We define such initialisations in an *Init* block. The *Init* block may also occur within an *Actor* block (see below). In this case, it holds initialisations that are protocol-specific.

Procedure This block holds procedures. The block may be used either after *Init*—in that case, it holds procedures that may be used by any participant in any protocol. Or it may be used inside an *Actor* block—in that case, the procedure is only available to the respective actor (i.e., participant acting in this protocol, see below).

Service This block holds a service. Since services are a kind of participant, the block may hold the same blocks as they also occur inside the *Actor* block (see below).

Protocol One *Protocol* block is used for each protocol in a scheme.

Actor The *Actor* block holds the descriptions how a participant behaves in a protocol. We use the term *Actor* to indicate that this is a participant executing the protocol, not just an element in the sets of participants.

Channels We denote channels between participants explicitly, and group all channels in this block.

Actions This is the primary block in an *Actor* block. The actions that an *Actor* carries out are described here.

Event We use events to define under which circumstances participants execute certain steps. This is described in more detail in Section 7.1.4.

Sessions This block holds the description which protocols in a scheme run with which participants, and when.

Several blocks, namely *Record*, *Service*, and *Procedure*, have a name to allow to refer to them from within the scheme. The block *Scheme* also has a name—we use this to refer to the execution of a scheme from within another scheme. We elaborate on this in Section 7.2.

7.1.4. Sessions and event-driven descriptions

A concept that is commonly found in role scripts is to define steps to execute as a reaction to events that occur during the execution of a protocol. A concept that is related to this are *sessions*: the protocols of a scheme may run at different times and also have dependencies on other protocols. We introduce both concepts together here.

Sessions The notation to express that a protocol is to be run takes the form of a call as we know it from programming languages: the protocol is called like a procedure, with parameters passed in round brackets. The parameters of a protocol are always elements from the sets of participants. In the context of a protocol, we refer to them as *actors*. We allow protocols to be run multiple times, and with different actors taken from the sets of participants.

The need to describe dependencies between protocols is best explained by example. Consider the case of TLS with X.509: a domain must have received a certificate from a CA before it can authenticate itself to a client. Both the certification as well as TLS are protocols, but one must run before the other. Section 7.2 shows the concrete notation.

Events With sessions now defined, we turn to the event-driven description of actions of actors. Recall that actors are records, and thus have an implicit field *state*. In protocols, this field is used to denote at which point in the protocol an actor currently is (e.g., it might be waiting for a certain message). The values may be arbitrarily chosen—we usually assign it a String (a well-known record, see Section 7.1.5). The value of the field is initially always 'Start'. As mentioned above, we sometimes need to state that an error has occurred. In such cases, we simply assign *state* the value 'Error'. The state field of an actor and the events that occur determine together what steps an actor executes next. We define two different kinds of events.

The first kind is used to start a protocol. This links the concept of sessions to the concept of events. When we denote in a session that a protocol is run, an implicit event *Start* is passed to the protocol. There must be exactly one actor defined that will react to the event. Note that sessions allow protocols to run multiple times. Furthermore, once the state of an actor has been set back to 'Start', the actor will react to the event *Start* again (unless additional conditions have to be met, too).

The second kind of event is the arrival of a message on a channel. This must generally coincide with a participant being in a certain state (for example, awaiting a reply to a previous message). If the message matches the description given in the event, the actor will execute the steps defined for this particular event.

7.1.5. List of well-known records

The following is a list of the well-known records in our notation.

String This represents a string of characters. We abstract over implementations in computer systems and alphabets. When we write a String, we generally surround it with ticks, e.g., 'message'.

Integer This represents an integer value. We abstract over implementations in computer systems.

Boolean This represents a boolean value, i.e., either *true* or *false*. We abstract over implementations in computer systems.

Timestamp This indicates a point in time. We abstract over implementations in computer systems, and also over possible calendars and time zones. Timestamps are instead universally comparable within our schemes. We define the arithmetic operation 'difference' on them. The difference between two *Timestamp* records is an *Integer*, which gives the difference in seconds.

Nonce This represents a nonce, i.e., a value that is only used once across all runs of a protocol. A nonce will have a different value every time a protocol is executed.

Sym_Key This represent a symmetric key as used in symmetric ciphers. We abstract over concrete algorithms like AES, etc.

Pub_Key This is our representation of the public part of an asymmetric public/private key pair. We abstract over concrete algorithms like RSA or ElGamal. A field that holds a *PubKey* is often denoted by k.

Priv_Key This represents the private (secret) part of an asymmetric key pair. We abstract over concrete algorithms in the same way we do for *Pub_Key*. A field that holds a *Priv_Key* is often denoted by k^{-1}.

Key_Pair This well-known record represents a tuple of *Pub_Key* and *Priv_Key*. We introduce it here as a record of its own since we often use it when denoting the creation of a key pair.

Cert This denotes a certificate as an entity's name and public key, with a signature on both. We allow two ways to denote the generation of a certificate. One is by writing *new Cert(domain)*. We typically use this when we only want to indicate that a certificate for a given subject has been brought into existence, but we do not need to indicate either specific key or signing party. The second way is to denote these factors explicitly. We write, e.g., *new Cert(d,k)* to indicate the public key k explicitly. If we also want to indicate the signing key, we write *new* $Cert_{k^{-1}}(d, k_d)$. We always assume that the public key can be accessed by the field k.

DHParm This is our generic representation of Diffie-Hellman parameters as they are needed in a Diffie-Hellman key exchange [17]. A *DHParm* is a tuple of a secret value and public value (that is to be sent to the other party involved in a key exchange). We abstract over multiplicative groups and primitive roots as used in [17].

7.1.6. List of well-known processes

The following is a list of well-known processes we use in our notation. We will extend this list later to include scheme-specific processes.

time() This well-known process yields the current time as a *Timestamp*. It takes a String as an argument. *now* means the current moment; otherwise the argument indicates how far in the future the timestamp will be. For example, time('48h') means '48 hours from now'.

hash(x) This represents the application of an ideal cryptographic hash function to the argument x. The output of *hash(x)* is denoted as a String.

mac(x) This represents the application of a function to create a Message Authentication Code (MAC) to the argument x. In cryptography, this generally involves a series of steps like repeated hashing of input and key, and often padding. We generally denote the key in the index and write mac_k. The output of *mac(x)* is denoted as a String.

sig(x) This represents a digital signature on the argument x. In cryptography, this generally involves several steps like hashing an input, adding padding, and encrypting the thus obtained value with a private key. We generally write the key in the index, e.g., $sig_{k^{-1}}$. The output of *sig(x)* is denoted as a String.

$valid\_sig_k(x, s)$ This denotes the process of verifying that a signature is correct. Argument x is the token on which the signature was created. Argument s is the signature. The key is denoted in the index.

derive_dh(s,p) This represents the computation of a symmetric key as the result of a Diffie-Hellman key exchange [17]. The input values are the secret value of the local actor, s, and the public part of the *DHParm*, p.

valid_cert(c), valid_cert(c, n) This denotes combined verification and validation of a certificate. We understand the terms verification and validation as before: verification means correct signatures and a certificate chain that chains up to a recognised root. We abstract over the concrete root store here. Validation means validation in the style of RFC 5280 [94], in particular checks of validity periods and correctness of subject. We do not include revocation checks as part of the process. This well-known process may be called with the certificate to validate or with the certificate and the name that is expected to be in the subject of the end-host certificate.

meets_cps(d) This describes the actions that a CA carries out when determining whether it may issue a certificate for a domain. These actions are generally defined in the CA's CPS, and involve business processes and possibly legal checks. d is the name of the domain.

7.1.7. Operators and comments

The final elements that we need in our notation are basic operators. The following is a list of operators we use in our notation:

$x \leftarrow y$ expresses assignment of y to x.

$x = y$ expresses a test of equality of x and y.

$x|y$ expresses concatenation of x and y. We allow concatenation of arbitrary records.

new X() This expresses the instantiation of a record X.

'X' This is shorthand notation for the instantiation of a String X. Note this is just syntactic sugar to avoid using the *new* operator for Strings, a concession we make for succinctness.

$\triangleright$ We also allow comments in our notation and precede them with this symbol.

7.2. Representation of design elements in the notation

In the following, we show how we represent the different design elements in our notation.

Blocks A block is always denoted by its type and the name we give it. We use a colon to indicate that a block starts and indent all parts of a block. Example:

```
1: Scheme TLS:
2:     ▷ Rest of scheme
3:     ...
```

Participants Participants are denoted in a block of their own. We denote them as a set and by the name under which we refer to them. Note that the set may be infinite or finite. Example:

```
1: Participants:
2:     CAs : $\{Ca_0, \ldots, Ca_9\}$                ▷ Set of CAs, finite
3:     Domains : $\{D_0, D_1, \ldots\}$              ▷ Set of domains, infinite
```

Records We define new records by giving the record a name and specifying the fields and the well-known records they are going to hold (separated by a colon). Note that the field *name* is implicit. A record can be instantiated with *new* and specifying the values of the fields. Example:

```
1: Record Key_Info:
2:     owner : String                          ▷ Name of owner as String
3:     k : Pub_Key                             ▷ Public key
4: ki ← new Key_Info(‘Alice’, $k_{Alice}$)
```

Maps Maps are instantiated just like records. We can also indicate which records they are going to hold. Access is possible with ‘array notation’ and via the well-known processes *keys()* and *values()*. Example:

```
1: store ← new Map⟨String, Pub_Key⟩()
2: store[‘Alice’] ← k                          ▷ Store k under String ‘Alice’
3: $k_{tmp}$ ← store[‘Alice’]                  ▷ Retrieve from Map
4: store.keys()                                ▷ Yields {‘Alice’}
5: store.values()                              ▷ Yields {k}
```

For-loops We denote *For-loops* as blocks and denote the records over which the loop is going to be applied in set notation. Example:

```
1: for k ∈ {pubkeys}:                          ▷ Loop over a set of public keys
2:     ▷ Carry out some actions:
3:     ...
```

If-clauses We denote *If-clauses* as blocks. An *If-clause* may test one condition or several ones. In the latter case, further conditions can be added after the words *and* or *or*. *Else-clauses* and *Else-if* clauses are possible. Example:

```
1: if $b > 5$ and $c < 10$:
2:     ...
3: else if $b < 3$:
4:     ...
5: else:
6:     ...
```

Initialisation Global initialisation and local initialisation blocks follow both the same structure. We often use *For-loops* to initialise a number of records at once. Example:

```
1: Init:
2:     for Ca ∈ CAs:
3:         $(Ca.k, Ca.k^{-1})$ ← new Key_Pair()     ▷ Each CA has one key pair.
```

Protocols Protocols are grouped in a container block, and each protocol has its own block. An example follows.

```
1: Protocols:
2:     Protocol TLS ((Cl,D)):
3:         ▷ Definition of protocol follows...
4:         ...
5:     Protocol Certification((D,Ca)):
6:         ▷ Definition of protocol follows...
7:         ...
```

Sessions Protocols are called from the sessions block, where they are called in the form of a procedure, with the participants passed as a tuple. This allows us to pass participants from different sets. We denote dependencies as *Dependency X not before Y*, with X and Y being calls to protocols. Since we usually cannot explicitly name a particular participant as a parameter—due to protocols being allowed to run multiple times and with different actors—we indicate variable actors with an $*$. We understand this to mean that for any protocol with a certain combination of actors to run, the dependency must be fulfilled for exactly these actors.

Once dependencies are defined, we write *At random* to indicate that there are otherwise no constraints that govern when a protocol is allowed to run. The protocol calls carry the names of actors as parameters. The following shows an example session:

```
1: Sessions:
2:     Dependency TLS_DHE(*, *) not before Certification(*,*)
3:     At random Certification with $(\mathrm{D}, \mathrm{Ca}) \in \mathrm{Domains} \times \mathrm{CAs}$
4:     At random TLS_DHE with $(\mathrm{Cl}, \mathrm{D}) \in \mathrm{Clients} \times \mathrm{Domains}$
```

Actors Actors are the specific participants in a protocol—passed to a protocol by a call in a session. Their activities are defined in a corresponding block. Example:

```
1: Actor Cl:                                        ▷ Actor Cl is a client
2:     ▷ Definitions for actor follow...
3:     ...
```

Channels Channels are defined for each actor, in a block of their own. We denote the existence of a channel by indicating which actors in the protocol it connects. Secure channels are denoted analogously. Actors may use the channels to send and receive records. When we denote that an actor sends records over a channel, we add the records to be sent in parentheses. When we denote that it receives records, we denote these by a name under which they can be referred to and the kind of record or well-known record that is expected (after a colon ‘:’). Example:

```
1: Actor A:
2:     Channels:
3:         Ch: Channel(A, B)
4:         Sec_Ch: Sec_Channel(A, C)
5:     ...
6:     Ch.send(‘hello’)
7:     ...
8:     Ch.recv(reply: String)
```

Actions, events and state Participants react to events. Their activities are defined in the *Actions* block, which contains definitions for one or more events. The definition of an event is usually a combination of the arrival of a message and the state an actor is currently in. An example follows.

```
Actor A:
    Actions:
        Event state = 'Wait' and Ch.recv(cert: Cert):
            state ← 'Start'
            ...
```

Events may also be passed from within a session. The most important form of this is the implicit passing of the *Start* event when a protocol is called. But events can also be explicitly sent from within the session by adding them in parentheses. We denote this as shown below:

```
Protocol Verification((A,B)):
    Actor A:
        Actions:
            Event Start:
                ...                                   ▷ React to Start event

    Actor B:
        Actions:
            ▷ Normal event via message
            Event state = 'Wait' and Ch.recv('hello'):
                ...
            ▷ React to directly passed event
            Event Check:
                ...

Sessions:
    At random Verification with (A,B) ∈ Clients × Clients
    At random Verification(Check) with (A,B) ∈ Clients × Clients
```

Well-known processes Well-known processes are denoted and used just like procedures. Example:

```
Actor A:
    Actions:
        Event state = 'Wait' and Ch.recv(cert: Cert):
            if valid_cert(cert):
                ...
```

Services We denote services in a way that is similar to actors. There are two differences. First, services may contain *Init* blocks. Second, the definition of channels is different to denote that services can communicate with all participants in a scheme. Channels are thus denoted as sets, and when we denote the arrival of a message, we explicitly indicate on which channel it arrived.

In the following, we define the DNS as a service. The service DNS represents the set of all DNS servers. Domains are stored in a *Map* where the domain name is the key, and the value another *Map*. Keys in this second map are the types of resource records; the values are the actual resource records. There is one simplification we make: for each type of resource record, we allow only one entry. Although the real DNS allows more, this will be sufficient for our purposes. Note, however, that extending the description to store sets of entries instead would be straight-forward.

```
Participants:
    Clients : $\{Cl_0, Cl_1, \dots\}$                    ▷ Set of clients
    Domains : $\{D_0, D_1, \ldots\}$                                   ▷ Set of domains
    DNSServers : $\{DNS_0, DNS_1, \ldots\}$                         ▷ Set of all DNS servers

**Record** RR:                                                  ▷ Generic resource record
    ▷ We store the type...
    type : String
    ▷ ... and the value.
    value : String

**Service** DNS **represents** DNSServers:
    **Channels:**
        Chs : {Channel(q, d) : $q \in$ (Clients $\cup$ Domains), $d \in$ DNSServers}
    **Init:**
        domains ← new Map⟨String, Map⟨String, RR⟩⟩()
    **Actions:**
        **Event** Ch ∈ Chs: Ch.recv('store', d: String, rr_type: String, val: String):
            ▷ Create resource record:
            rr ← new RR(rr_type, val)
            ▷ Store resource record:
            domains[d][type] ← rr
        **Event** Ch ∈ Chs: Ch.recv('load', d: String, type: String):
            ▷ Retrieve resource record:
            rr ← domains[d][type]
            Ch.send(rr)
```

Running a protocol from without a scheme In general, every block that carries a name can be referenced by this name. We already described one use of this when we introduced the call to protocols in sessions and when we described the notation for services. There is one further use we introduce now: we allow to call a protocol defined in another scheme directly by calling the scheme and protocol by their names, concatenated with a dot. This allows us to let an actor run a protocol like TLS. When we do this, we understand that the called protocol may set fields of the actor, even if these have been set previously. The following gives an example:

```
**Actor** A:
    **Actions:**
        **Event** state = 'Start':
            ▷ Actor A runs TLS_DHE from TLS, with actor B.
            TLS.TLS_DHE( (A,B) )
            ▷ TLS_DHE stores $cert_S$ in A, which can be used later.
            $cert_{new} \leftarrow cert_S$
```

7.3. Example: certification in the current X.509 PKI and TLS

We give an example how our notation can be used to describe TLS together with a certification process in which a CA is involved. The protocol flow of TLS is the same as described in Section 2.4, except that we omit client authentication. Listings 2 and 3 at the end of this chapter show the complete definition. In the following, we discuss the example by going through each block.

The participants (lines 2–5) are defined as sets of CAs, domains and clients. There is one global initialisation (lines 7–11) to take into account that every CA must have a public and private key, as does every domain.

Looking at the *Session* block (lines 80–83), we see that a dependency is defined: the protocol *TLS_DHE* must not be run without a previous run of *Certification* first. Otherwise, the protocol may be run at random. *TLS_DHE* may run with an arbitrary domain and client, denoted as a tuple from the Cartesian product of the respective sets. Certification may be run with an arbitrary combination of domain and CA.

The protocol *Certification* (lines 14–36) describes how a domain may obtain a certificate for its public key. Note that there must be a secure channel between CA and domain—this is because the domain must be authenticated to the CA in order to receive a certificate. The protocol is defined as follows. The actor to react to the *Start* event is the domain (line 20). It causes it to send its name and public key to the CA (line 21), and then wait for a reply. We model the latter by setting *state* to 'Wait_Cert'.

The message sent by the domain arrives on the channel that links the CA to the domain. The CA must now determine whether the domain with this name meets its requirements that it has defined in a CPS. We model this with the well-known process *meets_cps* (line 34). Note that this well-known process operates on the domain, not just the domain name (line 33). We thus obtain the domain by its name from the set of domains. If the CA determines it can issue a certificate, it creates the new certificate (line 35) and sends it to the domain (line 36).

The domain receives the certificate on the channel that links it to the CA (line 23). Note that it is in state 'Wait_Cert'. It stores the certificate in a field and returns to state 'Start'.

The TLS protocol is initiated by the client, which reacts to the *Start* event (line 42), which is passed from within the session (line 83). The client generates a nonce, sends it to the domain, and enters a waiting state (lines 43–45).

The domain receives this (line 63). It generates a nonce itself as well as the necessary Diffie-Hellman parameters and sends both back to the client, together with a signature. It then enters a waiting state itself (lines 64–68).

Upon receiving the domain's reply (while in waiting state, lines 46–47), the client first validates the domain's certificate, with the well-known process *valid_cert*, and verifies the signature (line 48). It then generates its own Diffie-Hellman parameters and uses the domain's Diffie-Hellman parameters and its own secret Diffie-Hellman value to compute a symmetric key with the well-known process *derive__dh()* (shown in lines 49–50). This key is then used to compute the MAC over all messages so far (with the well-known process *mac()*), which is sent to the domain as a String, together with the client's Diffie-Hellman parameters (lines 51–53). The domain receives this message (lines 69–70). It derives the symmetric key, using the well-known process *derive_dh()* (line 71). It verifies if the MAC it computes itself is the same String that it received (line 72). If this is the case, it computes a final MAC over the fields that have been used so far (line 73). This is sent to the client. The server is finished with the TLS connection setup (lines 76–78). The client receives the server's final message (line 54). It verifies if the MAC is the same it can compute itself (line 55). If this is the case, the handshake is finished. Note that we do not model in our notation what happens after the TLS handshake has finished.

7.4. Related work

To the best of our knowledge, there is no related work that would address the exact question how to describe PKI schemes in a unified notation. The Universally Composable Security framework [13] allows to specify protocols in a highly formal way and then use these descriptions to reason about their security. For our purposes, however, the framework is too fine-grained: it lacks abstractions for typical PKI processes, and would thus result in very long descriptions. Furthermore, some of the PKI processes that we need to include in our notation cannot be easily defined in a formal way (e. g., verification against practices documented in a CPS).

Our notation has drawn inspiration from two well-known projects. The first such project is AVISPA [140]. AVISPA is a model checker for the automatic verification of cryptographic protocols. It supports a variety of definitions of authentication, among them Lowe's definition of 'authentication as injective agreement' [51] that we introduced in Chapter 2. AVISPA defines the High Level Protocol Specification Language (HLPSL) [139]. HLPSL definitions are role scripts: sessions determine which possible participants may act in which protocol and execute certain steps. The arrival of a message is modeled as the occurrence of events while an actor is in a certain state. HLPSL served as a blueprint in the design of our notation: the building blocks of our notation are very similar, in particular initialisation, actors, protocols and sessions, as well as the event-based notation. In comparison with AVISPA, our notation is considerably more high-level. There are several important differences. First, we allow to define records and we replace AVISPA's data types (of which there are very few) with our well-known records. Second, the concept of services does not appear in AVISPA—in HLPSL, all communication is from peer entity to peer entity. Third, we allow to define dependencies between protocols. Finally, we use predicate logic to express many steps that actors execute, which gives our notation a very high degree of flexibility. The differences that our notation shows to AVISPA are, naturally, a consequence of the different purpose it serves. HLPSL is intended to be a language that can be automatically translated into an intermediate format, to which different model checking back ends can be applied. HLPSL also allows to define protocol goals and violations, which the model checker can detect. This is not a concern for our notation. It should also be noted that we allow to abstract over very complex processes with the help of well-known services—HLPSL cannot allow this lesser degree of formality due to the need to be a compilable language. Instead, AVISPA and HLPSL must focus on the implementation of selected primitives that allow it capture protocol flows (rather than entire schemes, or even their interaction, as is the case in our notation).

The second project that uses a related notation for role scripts is Scyther [172]. Like AVISPA, Scyther is a tool for the automatic verification of protocols. Its input language, SPDL, shows similarity to AVISPA but is less verbose. Scyther has some advantages over AVISPA when it comes to verifying protocols for an unbounded number of parallel protocol sessions, and it also supports multiple definitions for authentication.

7.5. Key contributions of this chapter

This chapter addressed the first part of Research Objective O3.1. We introduced a formalised notation to capture schemes that describe how to enhance or reinforce the X.509 PKI. Our contributions in this chapter were as follows:

Concise presentation Our notation makes it easy for a reader to determine the key elements in a scheme: participants, channels, and protocols. It allows several protocols per scheme, allows to define how they interact and which actions participants in each protocol execute.

Design decisions We made several design decisions to keep our notation concise yet powerful. Event-based role scripts define the behaviour of participants. Mathematical set notation, predicate logic, For-loops and If-clauses allow to model decision processes in protocols in a succinct way. Records are our basic representation for entities and tokens, enhanced with Maps. Well-known processes, well-known records and services are abstractions over descriptions that would be lengthy or impossible to formalise.

Example: TLS We gave an example how our notation can be used to describe the current CA-based certification process in the context of TLS.

Listing 2 Scheme of TLS with CA certification, Part 1.

```
Scheme TLS:
    Participants:
        CAs : {Ca_0, Ca_1, ...}                                  ▷ Set of all CAs
        Domains : {D_0, D_1, ...}                                ▷ Set of all domains
        Clients : {Cl_0, Cl_1, ...}                              ▷ Set of all clients

    Init:
        for Ca ∈ CAs:
            (Ca.k, Ca.k^-1) ← new Key_Pair()                     ▷ Each CA has one key pair.
        for D ∈ Domains:
            (D.k, D.k^-1) ← new Key_Pair()                       ▷ Each domain has one key pair.

    Protocols:
        Protocol Certification((D,Ca)):
            Actor D:                                             ▷ Actor D is a domain
                Channels:
                    ▷ Certification requires a priori secure channel
                    Ch: Sec_Channel(D, Ca)                       ▷ domain ↔ CA
                Actions:
                    Event Start:
                        Ch.send(name, k)                         ▷ name is implicit field in every record
                        state ← 'Wait_Cert'
                    Event state = 'Wait_Cert' and Ch.recv(cert_new: Cert):
                        cert ← cert_new
                        state ← 'Start'

            Actor Ca:
                Channels:
                    Ch: Sec_Channel(Ca, D)                       ▷ domain ↔ CA
                Actions:
                    Event Ch.recv(domain_name: String, k_rcv: Pub_Key):
                        ▷ Get domain with this name:
                        d ← (d_i ∈ Domains : d_i.name = domain_name)
                        if meets_cps(d):
                            cert ← new Cert_{k^-1}(domain_name, k_rcv)
                            Ch.send(cert)
```

Listing 3 Scheme of TLS with CA certification, Part 2.

```
Protocol TLS_DHE((Cl,D)):
    Actor Cl:                                              ▷ Actor Cl is a client
        Channels:
            Ch: Channel(Cl, D)                             ▷ Client ↔ domain
        Actions:
            Event Start:
                $n_{Cl}$ ← new Nonce()
                Ch.send($n_{Cl}$)
                state ← ‘Wait_Kex’
            Event state = ‘Wait_Kex’ and Ch.recv($n_D$: Nonce, $cert_D$: Cert,
              $dh_D$: DHParm, sgn: String):
                if valid($cert_D$, D.name) and $valid\_sig_{cert_D.k}$($n_C|n_D|dh_D$, sgn):
                    a, $dh_{Cl}$ ← new DHParm()            ▷ Generate DH parameters
                    k ← derive_dh(a, $dh_D$)               ▷ Compute symmetric key
                    msg_mac ← $mac_k(n_{Cl}|n_D|cert_D|dh_D|sgn|dh_{Cl})$
                    Ch.send($dh_{Cl}$, msg_mac)
                    state ← ‘Wait_Accept’
            Event state = ‘Wait_Accept’ and Ch.recv(fin_mac: String):
                if fin_mac = $mac_k(n_{Cl}|n_D|cert_D|dh_D|sgn|dh_{Cl}|msg\_mac)$:
                    . . . ▷ TLS handshake done
                    state ← ‘Start’

    Actor D:                                               ▷ Actor D is a domain
        Channels:
            Ch: Channel(D, Cl)                             ▷ Domain ↔ client
        Actions:
            Event state = ‘Start’ and Ch.recv($n_{Cl}$: Nonce):
                $n_D$ ← new Nonce()
                (b, $dh_D$) ← new DHParm()                 ▷ Generate DH parameters
                sgn ← $sig_{k^{-1}}(n_C|n_D|dh_D)$          ▷ Create a signature
                Ch.send($n_D$, cert, $dh_D$, sgn)
                state ← ‘Wait_Kex’
            Event state = ‘Wait_Kex’ and
              Ch.recv($dh_{Cl}$: DHParm, rcv_mac: String):
                k ← derive_dh(b, $dh_{Cl}$)
                if rcv_mac = $mac_k(n_{Cl}|n_D|cert|dh_D|sgn|dh_{Cl})$:
                    fin_mac ← $mac_k(n_{Cl}|n_D|cert|dh_D|sgn|dh_{Cl}|rcv\_mac)$
                    ▷ The last MAC is necessary for confirmation:
                    ▷ an attacker cannot fake it
                    Ch.send(fin_mac)
                    ▷ TLS handshake done
                    state ← ‘Start’

Sessions:
    Dependency TLS_DHE(∗, ∗) not before Certification(∗,∗)
    At random Certification with $(D, Ca) \in Domains \times CAs$
    At random TLS_DHE with $(Cl, D) \in Clients \times Domains$
```

Chapter 8. Proposals to replace or strengthen X.509

In this chapter, we present schemes to reinforce the X.509 PKI for TLS. The approaches they employ can be categorised by the technologies they use: DNS, pinning, notaries, and public logs. For our analyses, we choose one representative of each technology. We describe each scheme with the notation we developed in the last chapter and give an assessment with regard to three aspects.

The first aspect is *evaluation of a scheme with respect to the criteria we developed in Chapter 3 to improve the status quo. We also take conclusions from Chapter 4 into account.* We thus structure this part of the analysis as follows:

Out-of-band protection The first question we ask is whether a scheme increases the security for clients by introducing out-of-band communication channels.

Incident detection The second question is whether a scheme allows to identify rogue certificates or rogue CAs fast and react to the incident.

Transparency by observation The third question is split into three sub-questions:

Transparency of certification We showed in Chapter 3 that it is helpful to increase the level of transparency in the certificate issuance process. Our question is thus: does a scheme facilitate an (ideally continuous) monitoring of the X.509 PKI, in particular certifications?

Transparency of deployment We showed in Chapter 4 that deployment of certificates is too often quite poor. We thus investigate whether a scheme makes contributions towards monitoring this factor.

Transparency of scheme itself Finally, we ask whether a scheme lends itself well to monitoring in the sense that external parties can determine its correct operation.

The *second aspect* we address is the *security of a scheme in the face of attackers, i. e., its robustness.* To this end, we define three threat models to describe attackers of different strength. These are inspired by the attackers we described in Chapter 3.

The *third aspect*, finally, is an *assessment of the changes that would be necessary to deploy a scheme on the Internet*, in particular which entities need to make changes to their systems. This is a key factor for a scheme's success.

8.1. Threat models

We present three threat models for different kinds of attackers, in order of rising strength. Our threat models form a hierarchy: the stronger attacker has all capabilities of the weaker attackers plus several additional ones.

Our attackers have in common that we allow them near-complete control over communication channels in certain parts of the network. In the parts of the network an attacker controls, he may eavesdrop on, alter, or delete any message. This is almost the Dolev-Yao model [18], except that we disallow the attacker to delay messages. Instead, we assume all our clients to set timeouts. Messages that are delayed but still arrive before the timeout are not rejected and not considered suspicious. Messages that arrive after the timeout are treated as if they had never arrived. The rationale here is twofold. First, Internet communication can show considerable delays, which are hard to predict, and developing a policy to treat delayed messages is error-prone. Second, none of the schemes we describe in this chapter discusses the notion of delayed messages, either.

As in the Dolev-Yao model, our attackers cannot break cryptographic primitives. Where entities use cryptography to provide authentication, encryption, and integrity protection, the attacker cannot tamper with the traffic in any way that would not be detected. We thus refer to these channels as *secure*.

Finally, none of our attackers is allowed to break the integrity of *end-systems*, i.e., clients and servers. The rationale here is that such an attack would trivially break the entire protection that TLS and X.509 can offer anyway.

Model A: local and resource-restricted attacker

Our first threat model represents an attacker with relatively few resources at his disposal who has nevertheless managed to compromise a local network by gaining access to a gateway device. This corresponds to the idea of an attacker working out of mischief or for financial gain and without the resources of an organisation at his disposal. We will generally discuss two different kinds of victim systems in this threat model. The victim system may be a local stub network, e.g., a private wireless network, which is attached to an Internet broadband provider via a gateway device (e.g., a wireless access point). In this case, the attacker is able to tamper with all traffic to and from the home network going through the gateway, but his control ends when the traffic is sent further upstream to the Internet provider.

Or the victim system may be the local network of an organisation that uses it for hosting purposes—e.g., a commercial Web server. This network is again linked to the global Internet via a router, which we assume the attacker to be in control of. Again, the attacker's control ends at the gateway to the public Internet. Note that, in both cases, the attacker is not in control over any end-host systems (client devices or servers).

Concerning further resources, this attacker cannot compromise *any* entities in a scheme. When this attacker stages a man-in-the-middle attack against TLS, his hope is that a portion of clients accepts a rogue (invalid) certificate despite a browser warning. Studies like [36] have shown that this number may be relatively high, so this is not an unreasonable assumption.

Model B: regional attacker

Our second threat model is meant to capture the notion of attacks running on a regional level, as was alleged in the case of DigiNotar. The motivation here is to model the possible actions of a rogue authoritarian state.

We extend the control of this attacker to routers that are geographically located inside his country. This means the attacker is in control of all routers in ASes that are located entirely within this region. Where an AS extends beyond the region, we assume that the attacker still only controls routers within his country. The rationale here is to model an attacker who, by means of state authority, can force access to these

Attacker	**local, weak (A)**	**regional (B)**	**global (C)**
in control of			
… network	localised to victim	AS in country	AS in country
… routing	no	no	BGP, or control over selected routers
… entities	no	limited number	limited number
… DNS	full control over local traffic	full control over local and regional DNS traffic	full control over regional and some global DNS traffic
… DNSSEC	no	zone of his country, permanent	additionally temporarily over selected zones

Table 8.1. – Summary of threat models representing the weak attacker (A), the regional attacker (B), and the globally active attacker (C).

devices. Consequently, this attacker has full control over all traffic entering and leaving the country, and all traffic within it.

We allow this attacker to be more powerful in terms of his capacity to compromise other systems, however. When we use the model, we allow the attacker to control a limited number of (chosen) entities in a scheme as long as they are not end-systems. For example, when discussing a scheme in the context of this threat model, we will allow the attacker to compromise a CA, which means the attacker is always able to obtain a rogue CA-signed certificate.

Furthermore, we give our attacker partial control over the DNS, even when it is protected with DNSSEC[1]. The rationale here is to model an attacker who, by legal or authoritarian means, can modify DNS entries in the zone that belongs to his region. Commonly, this would be the zone that represents his country's top-level domain. The attacker can tamper with any unprotected DNS traffic entering and leaving the region (and any DNS traffic within the region). However, where entries are protected with DNSSEC, we limit his influence to queries and replies concerning the DNS zones that the attacker has control over.

Model C: globally active attacker

This attacker is the most powerful attacker we consider. In addition to the powers of the regional attacker, we give this attacker some possibilities as they have not yet been documented in attacks. The intention is to model a 'lawless' attacker with the resources of a country at his disposal, who is willing to engage even in such activities as they might be condemned by other states. We acknowledge this attacker model is more extreme. However, with the Internet being a backbone of many societies now, it is not impossible that such attackers will exist, especially if we take into account that several countries have created departments for 'cyber-warfare' [193]. Compared to the regional attacker, we allow this attacker additional technical capacity.

The globally active attacker has partial control over selected routing paths on the Internet. Such control can be achieved by (at least) two means. The attacker may advertise BGP routing announcements (e.g., in the style of Hepner [199]) to route traffic through his own systems. Or he may be able to compromise chosen routers on the Internet and tamper with the traffic that flows through them. This is possible: in 2012, an unknown person used poorly protected devices, routers among them, to create a large botnet and carry out network scans [136].

[1]See Section 8.3.1 for an introduction to DNSSEC.

We also allow this attacker the political, legal, and technical means to exercise partial control over the DNS, even with DNSSEC. This means this attacker is able to compromise the signing keys for selected zones that are outside his own legal authority. However, we do not allow unlimited control over all zones, and we assume the attacker cannot sustain the attack on DNSSEC for more than a few days.

Table 8.1 summarises our three threat models. With these defined, we will now present the different proposals.

8.2. Pinning

The term *pinning* refers to the concept of keeping a local database of known mappings from entities to their respective public keys. Entries in the database can be created the first time an entity contacts another entity and learns about its public key. The combination of entity and public key is stored so it can be compared with the public keys presented during subsequent contacts. If the keys are the same, this is a strong indication that the authentication is secure. If there is a mismatch, there *might* be an ongoing man-in-the-middle attack. Unfortunately, pinning generally suffers from a problem, namely that there are legitimate reasons why a key might change. In SSH, for example, a host might simply have updated its host key, without any activity from an attacker. In TLS, there are many more legitimate reasons for changes of public keys or certificates, which increases the potential for false-positive warnings.

Several proposals to improve the resistance of X.509 against attacks rely on pinning. In the following, we present a particularly promising approach: *Trust Assertions for Certificate Keys (TACK).*

8.2.1. Choice of TACK as subject to study

Until a few years ago, pinning for TLS was not a feature of Web clients, despite an (abandoned) attempt at IETF standardisation [103]. The benefits were recognised when Google's method to ship the public keys of Google services with its Chrome browser helped detect the DigiNotar compromise. At about the same time, several add-ons were developed to implement pinning for the Firefox browser, e.g., [271, 275]. These did not just track a host's public key but also often stored information about the identity of the issuing CA as this was also seen as a way to detect possible compromise.

These approaches faced a serious problem, however: there are too many legitimate cases where certificates and public keys may change. Some Web sites may use several certificates, even from different CAs. Potential reasons can be load balancing mechanisms or CDNs[2]. Hence, pinning may produce false-positive warnings. This can be quite problematic. For example, we know from studies like [36] that it is actually economically rational for users to develop a lax attitude towards warnings as as a result of encountering false positives too often.

TACK proposes to address this problem by integrating additional information from server-side. It should be noted that a similar approach has also been proposed recently as a way to extend HTTP. The corresponding Internet-Draft [101] defines a way for a domain to indicate the currently valid certificate to a browser. This is similar to how TACK operates. However, TACK's approach is more flexible and adds explicit life-cycle management for pins. It is thus the concept we chose to study.

[2]During our early scans of X.509, we found anecdotical evidence of domains like `microsoft.com` serving different certificates from the same IP address.

8.2.2. TACK operation and representation in our notation

TACK makes two key observations. First, server operators have an incentive to help their clients (and users) connect to the right server. Second, they can help them by signalling clients information about the current certification status. The idea behind TACK is that server operators do not just hold a key pair for their certificate, but also create a second key pair that they use to sign assertions about their currently valid certificate. TACK also adds methods to address life-cycle-management, such as TLS key roll-over, key compromise, and replacement of the signing keys for the assertions.

TACK proposes a TLS extension that is currently defined in an Internet-Draft [112]. It has been implemented in several popular libraries and for some Web servers [215].

TACK does not require a certificate to be issued by a CA. Although normal X.509 certification may be used, TACK works just as well with self-signed certificates or certificates signed by a CA that is not included in a browser's root store. The only participants in the scheme are domains and clients. Note that while several domains may run on the same hardware, TACK treats every domain as a separate participant.

In the following, we will describe how TACK operates. Listings 4–6 denote TACK in our notation.

Initialisation (deployment)

The participants in TACK are denoted as two sets, clients and domains (lines 2–4). TACK requires every domain to create a key pair, which we show in the global initialisation block (line 20). The private key is called the *TACK Signing Key (TSK)*. It is used in the creation of a *tack*. A tack, shown in lines 6–13, is a signed data structure that holds the public part of the TSK, the hash value of the domain's X.509 public key, and validity information. The latter is expressed in terms of an expiry timestamp plus two so-called *generations*. A generation is essentially a version number that rises monotonously during the lifetime of a domain. Together, a minimum and current generation in the tack define a window of currently valid generations. A tack also holds a boolean flag that determines whether the tack is in so-called active state[3]. In our listings, we initialise all tacks for a validity period of one year. This value is just an example and may be chosen arbitrarily.

Tack life-cycle

The life-cycle management of TACK is represented as a protocol with just one participant, the domain. We show it in lines 34–72. The events correspond to the life-cycle of a tack. The major events that may occur are expiry of a tack, roll-over of a TLS key, compromise of the TLS key, and roll-over of the TSK itself. Domains store their tacks in a tack store, and update them to reflect changes.

Tacks expire and must be updated when their expiry date is exceeded (line 37). All expired tacks are replaced with new tacks. The minimum generation of the new tack is set to a new value that is higher than the highest minimum generation of any tack in the store. The current generation is also updated. The new tack stores exactly the old tack's information about TSK and TLS key and is set to active.

When a domain updates its TLS key (line 47), it creates a new tack (set to active) and stores it in the tack store (the old tacks that refer to the old key are not needed any more). The generation of the tack is the old minimum generation.

[3]In the RFC, this flag is sent as a value outside the tack, but inside the same TLS extension. In our notation, we add the flag to the tack itself, but do not sign it either. This is semantically equivalent.

The situation is different if a domain needs to replace its TLS key due to compromise (line 55). It needs to signal that earlier tacks are to be considered revoked. Just as in the expiry case, it increments both the minimum and current generations when creating a new tack. The new tack replaces all other tacks in the store and is set to active.

The only case left in the life-cycle of tacks is when a domain's TSK is compromised or must be replaced (line 64). In this case, the domain will generate a new TSK. It creates a new tack and increments both the minimum and the current generation to values that invalidate all tacks that refer to the old TSK. The new tack will be sent together with the old tacks; however, all old tacks are first set to inactive. This serves as a signal to clients that they must not be used any more.

Modified TLS protocol and pins

Tacks are used to express which public key a domain currently considers a valid choice for TLS. Domains send their tacks to clients, which use the information to create *pins*. Instead of referring to a domain's TLS key, a pin refers to the TSK. The data structure of a pin is shown in lines 15–18.

Pins are used within TLS. The modified TLS protocol is denoted in lines 74–135. There are two participants, domains and clients. Note that we only denote the parts of TLS where there are changes to the protocol; all other parts remain the same as previously shown in Listings 2 and 3. The most important changes occur when the client receives the tacks at the beginning of the handshake (lines 94–95) and when it processes them at the end of the TLS handshake (line 102 ff.). We define one procedure to determine the validity of a tack (lines 79–86). Within this procedure, we make use of the well-known process *valid_sig()*. The well-known process *valid_cert()* is a part of a client's evaluation of a server certificate.

A pin has a life time. When a client connects for the first time, the life time is set to 0 to indicate the pin is not yet active (line 120). It will be activated (and its life time extended) when the client reconnects and encounters valid tacks and TLS keys (line 117). The client also stores the minimum generation it learns from a tack. This serves as an invalidation and revocation feature: any (subsequent) tack encountered by the client that has a current generation less than the minimum generation stored by the client is considered invalid.

The life-cycle of pins is managed by clients as part of the TLS protocol. A client signals its support for TACK via a special message in the extension (line 91). The server will react to this message by adding its tacks in its reply. Upon receiving the tacks in the domain's reply, the client executes the usual steps to determine the validity of a certificate.

The client must determine revocations of TSKs. It verifies if any active pins refer to TSKs where the tack's minimum generation is less than the minimum generation stored by the client. If no revocation is found, the client updates the locally stored minimum generation (lines 107–110). Next, the client determines whether any active pins exist that refer to TSKs for which no tacks have been sent (lines 111–113). This can only occur if the client had previously connected to the domain on several occasions and created an active pin. If this is the case, and the client finds a pin that is not matched by any TSK in a tack, it must not consider the connection secure.

The final step a client executes is to update its pins. Pins that are not active yet and for which no tack has been found are removed (lines 115–116). This can happen if the client has connected to the domain only once before and the domain has changed its TSK in the meantime. Pins with a matching tack are activated or their life time extended. The life time that TACK proposes is 30 days or the time span since pin creation, whichever is lower. The client also adds new, inactive pins for those tacks

Listing 4 Scheme of TACK with TLS, Part 1.

```
Scheme TACK:
    Participants:
        Domains : {D_0, D_1, ...}                                   ▷ Set of all domains
        Clients : {Cl_0, Cl_1, ...}                                 ▷ Set of all clients

    Record Tack:
        tsk : Pub_Key                                               ▷ TACK Signing Key
        $g_{min}$ : Integer                                         ▷ minimum generation
        $g_{cur}$ : Integer                                         ▷ current generation
        exp : Timestamp                                             ▷ expiry date
        hash : h(Pub_Key)                                           ▷ Hash of TLS key
        sig : sig(Pub_Key|Integer|Integer|Timestamp|h(Pub_Key))     ▷ Signature
        active : Boolean                                            ▷ Tack activation flag

    Record Pin:
        init : Timestamp                                            ▷ Initialisation date
        end : Timestamp                                             ▷ Expiry date
        tsk : Pub_Key                                               ▷ TACK Signing Key

    Init:
        for D ∈ Domains:
            (D.tsk, $D.tsk^{-1}$) ← new Key_Pair()                  ▷ TACK Signing Key
            (D.k, $D.k^{-1}$) ← new Key_Pair()                      ▷ TLS key
            D.cert ← new Cert(D.name, D.k)                          ▷ Certificate
            ▷ New tack, valid for 1 year, set to active
            D.tack ← new Tack(D.tsk, 0, 0, time('1y'), h(D.k),
                        $sig_{D.tsk^{-1}}$(D.tsk|0|0|time('1y')|h(D.k)),true)
            ▷ Initiate tack store
            D.tacks ← {tack}
        for Cl ∈ Clients:
            Cl.pins ← new Map()                 ▷ Initialise pin store, per domain
            Cl.gens ← new Map()     ▷ minimum acceptable generations, per domain

    Protocol TACKLife(D):
        Actor D:
            Actions:
                Event expiry_check:
                    ▷ Get expired tacks and compute generations:
                    $tacks_{exp}$ ← {t ∈ tacks : t.exp < time('now')}
                    for $t_{exp}$ ∈ $tacks_{exp}$:
                        $\bar{g}_{min}$ ← max(t.$g_{min}$ : t ∈ $tacks_{exp}$) + 1
                        $\bar{g}_{cur}$ ← max(t.$g_{cur}$ : t ∈ tacks) + 1
                        $tack_{new}$ ← new Tack(tsk, $\bar{g}_{min}$, $\bar{g}_{cur}$, time('1y'), $t_{exp}$.hash,
                                    $sig_{tsk^{-1}}$(tsk|$\bar{g}_{min}$|$\bar{g}_{cur}$|time('1y')|$t_{exp}$.hash),
                                    true)
                        tacks ← (tacks∖ {$t_{exp}$}) ∪ {$tack_{new}$}
```

Listing 5 Scheme of TACK with TLS, Part 2.

```
            **Event** tls_key_roll_over:
                $(k, k^{-1}) \leftarrow$ new Key_Pair()
                cert $\leftarrow$ new Cert(name, k)
                $\bar{g}_{min} \leftarrow \max(t.g_{min} : t \in tacks)$
                $\bar{g}_{cur} \leftarrow \max(t.g_{cur} : t \in tacks) + 1$
                $t_{new} \leftarrow$ new Tack(tsk, $\bar{g}_{min}$, $\bar{g}_{cur}$, time('1y'), h(k),
                                        $sig_{tsk^{-1}}(tsk|\bar{g}_{min}|\bar{g}_{cur}|time('1y')|h(k))$, true)
                tacks $\leftarrow \{t_{new}\}$
            **Event** tls_key_compromise:                    ▷ TLS key compromised
                $(k, k^{-1}) \leftarrow$ new Key_Pair()
                cert $\leftarrow$ new Cert(name, k)
                ▷ Revoke by incrementing $g_{min}$
                $\bar{g}_{min} \leftarrow \max(t.g_{min} : t \in tacks) + 1$
                $\bar{g}_{cur} \leftarrow \max(t.g_{cur} : t \in tacks) + 1$
                $t_{new} \leftarrow$ new Tack(tsk, $\bar{g}_{min}$, $\bar{g}_{cur}$, time('1y'), h(k),
                                        $sig_{tsk^{-1}}(tsk|\bar{g}_{min}|\bar{g}_{cur}|time('1y')|h(k))$, true)
                tacks $\leftarrow \{t_{new}\}$
            **Event** tsk_roll_over:                               ▷ Renewal of TSK
                **for** $\{t \in tacks : t.active = true\}$:
                    t.active $\leftarrow$ false                       ▷ De-activate old tacks
                $(tsk, tsk^{-1}) \leftarrow$ new Key_Pair()            ▷ New TACK Signing Key
                $\bar{g}_{min} \leftarrow \max(t.g_{min} : t \in tacks) + 1$
                $\bar{g}_{cur} \leftarrow \max(t.g_{cur} : t \in tacks) + 1$
                $t_{new} \leftarrow$ new Tack(tsk, $\bar{g}_{min}$, $\bar{g}_{cur}$, time('1y'), h(k),
                                        $sig_{tsk^{-1}}(tsk|\bar{g}_{min}|\bar{g}_{cur}|time('1y')|h(k))$, true)
                tacks $\leftarrow$ tacks $\cup \{t_{new}\}$

**Protocol** TLS_DHE_TACK((Cl,D)):
    **Actor** Cl:
        **Channels:**
            Ch: Channel(Cl, D)                                   ▷ Client $\leftrightarrow$ Domain

        **Procedure** valid_tack(t: Tack, cert: Cert, $g_{min}$: Integer):
            ▷ Check generations, especially against locally stored $g_{min}$
            **if** $t.g_{min} \geq g_{min}$ **and** $t.g_{cur} \geq t.g_{min}$ **and** time('now')< t.exp
              **and** h(cert.k) = t.hash
              **and** $valid\_sig_{t.tsk}(t.tsk|t.g_{min}|t.g_{cur}|t.exp|t.hash, t.sig)$:
                **return** true
            **else:**
                **return** false

        **Actions:**
            **Event** Start:
                $n_C \leftarrow$ new Nonce()
                Ch.send($n_C$, 'tack_req')                             ▷ request tack
                state $\leftarrow$ 'Wait_Kex'
```

Listing 6 Scheme of TACK with TLS, Part 3.

```
            **Event** state = ‘Wait_Kex’ **and**
             Ch.recv($n_D$: Nonce, $cert_D$: Cert, $dh_D$: DHParm, sgn: String,
              tacks: {Tack}):
                ...
                ▷ Tacks received, continue normally...
            **Event** state = ‘Wait_Accept’ **and** Ch.recv(fin_mac: String):
                ...
                ▷ Finish handshake normally, then check: all tacks valid?
                ▷ Also, only one tack per key is allowed.
                $g_{min} \leftarrow gens[D]$
                **if** $(\forall t \in tacks : valid\_tack(t, cert_D, g_{min}))$
                  **and** $(\forall t_i, t_j \in tacks : t_i.hash \neq t_j.hash)$:
                    ▷ Check tack generations
                    **for** (p, t) ∈ pins[D] × tacks:
                        **if** $p.tsk = t.tsk$ **and** $pin.end > 0$ **and** $t.g_{min} \leq g_{min}$:
                            state ← ‘Error’            ▷ pin active, but tack revoked
                        **else if** $p.tsk = t.tsk$ **and** $pin.end > 0$ **and** $t.g_{min} > g_{min}$:
                            $gens[D] \leftarrow t.g_{min}$                    ▷ update $g_{min}$
                    ▷ Determine store status
                    **if** $(\exists p \in pins[D] : (p.end > 0 \wedge (\nexists t \in tacks : t.tsk = p.psk)))$:
                        state ← ‘Error’                   ▷ No tack for active pin
                    ▷ Pin activation
                    **for** $p \in \{\bar{p} \in pins[D] : \bar{p}.end = 0 \wedge (\nexists t \in tacks : t.tsk = \bar{p}.psk)\}$:
                        $pins[D] \leftarrow pins[D] \setminus \{p\}$            ▷ remove inactive pins
                    ▷ Activate all pins with matching tacks:
                    **for** $(p, t) \in \left\{(\bar{p}, \bar{t}) \in pins[D] \times tacks : \bar{t}.active \wedge (\bar{t}.tsk = \bar{p}.tsk)\right\}$:
                        $p.end \leftarrow time(‘now’) + \min(30d, time(‘now’) - p.init)$
                    ▷ Add inactive pins for unmatched tacks:
                    **for** $t \in \left\{\bar{t} \in tacks : \bar{t}.active \wedge (\nexists p \in pins[d] : p.tsk = \bar{t}.tsk)\right\}$:
                        $p_{new} \leftarrow$ new Pin(time(‘now’), 0, t.tsk)
                        $pins[D] \leftarrow p_{new}$
                ... ▷ TLS connection setup finished
                **else:**                                   ▷ invalid tacks found
                    state ← ‘Error’

        **Actor** D:
            **Channels:**
                Ch: Channel(D, C)
            **Actions:**
                **Event** state = ‘Start’ **and** Ch.recv($n_C$: Nonce, ‘tack_req’):
                    ... ▷ normal TLS
                    Ch.send($n_D$, cert, $dh_D$, sgn, tacks)
                ... ▷ Continue with normal TLS

    **Sessions:**
        **At random** TACKLife(expiry_check) **with** D ∈ Domains
        **At random** TACKLife(tls_key_roll_over) **with** D ∈ Domains
        **At random** TACKLife(tls_key_compromise) **with** D ∈ Domains
        **At random** TACKLife(tsk_roll_over) **with** D ∈ Domains
        **At random** TLS_DHE_TACK **with** (Cl, D) ∈ Clients × Domains
```

that are active but for which no pins exist yet (see above). This is important when a domain is in the transition phase from one TSK to the next: the current TSK would still be accepted due to the previous step, but the client is prepared for a new TSK for the next connection. After this final step, the TLS session setup is finished.

We define five calls to protocols in the session block (lines 137–142). The first four refer to the events in a tack's life-cycle; we call these at random. The TLS protocol is also executed at random—note that the initialisation takes care that all necessary tacks are available.

8.2.3. Assessment of TACK

We assess the contributions that TACK makes towards a more secure X.509 PKI and discuss its robustness against attackers of varying strength. We also discuss possible difficulties in deploying TACK.

Reinforcement of X.509

TACK makes the following contributions to the security of X.509.

Out-of-band mechanism TACK aims to strengthen the authentication in TLS. Servers use a second key pair to signal a client whether an encountered key is genuine and correct. Although this information is transmitted in a TLS extension, the second key pair makes it a true out-of-band mechanism. TACK aims directly at improving the security for clients, doing so in an entirely transparent way.

Incident detection TACK does not provide any means to determine the cause of key mismatches automatically nor to report them, and the authors of TACK do not mention incident detection as a motivation for TACK. Although TACK is able to detect mismatches of keys, such a detection would only be conclusive evidence of a man-in-the-middle attack if one could assume that all server operators who use TACK execute their duties faithfully. This assumption seems questionable: in our scans (see Chapter 4), we found a certain laxness concerning certification (e. g., expired certificates, self-signed certificates without correct hostname). Given that TACK requires domain owners to renew expired tacks and carry out key roll-overs with care, there seems to be good reason to assume that mismatches will occur in benign situations, too.

Transparency by observation TACK is not concerned with monitoring certification nor with the deployment of X.509 certificates. In fact, the scheme can even work without any CAs at all. Concerning its own transparency, the concept lends itself well to observation by external parties. It is easy to determine whether a given deployment is problematic or not: tacks contain all the information that is necessary to compare the configuration of TACK with the actual certification of a domain. As they are transmitted as part of a TLS extension, tacks can be obtained with active scans.

Robustness against attackers

The security provided by TACK depends entirely on the security of the first contact. If this contact remains uncompromised, the scheme yields an authenticated secure channel for later use. Working under the assumption that the first contact was secure, TACK is a remarkably strong concept. This is due to the fact that TACK avoids TTPs. As none of our threat models allows servers or clients to be compromised, tacks and pins can thus be created safely. Consider our strongest attacker from Model C. Even if we allow this attacker to compromise all CAs *and* control all network paths on the Internet, he can only suppress TACK communication, but not alter it in a way that would have impact. Our strongest attacker has all the possibilities of the weaker attackers of Models A and B. We can conclude that TACK can withstand attacks even by globally active attackers with the resources of countries.

Note however that TACK's omission of TTPs is also its primary weakness. In extremely security-critical scenarios, e.g., political dissidents using Web mail from a foreign provider, it seems unwise to put trust into the security of a first contact. The authors of TACK are aware of this problem and hint at a solution in their RFC. Although not yet specified, it may be suitable on a smaller scale: the authors propose that a TACK client is initialised with pins from a trusted source. These could be propagated out-of-band, e.g., on an external medium. Note that the tack values could be determined semi-automatically with the help of active scans—ideally, from several different vantage points to avoid attackers in certain networks.

Deployment

TACK is proposed as an extension to TLS. This means it does not require any upgrades on client-side: a client that is not aware of TACK will simply not use the extension. This makes it possible for TACK to be deployed gradually.

Unfortunately, TACK puts a significant burden on server operators. First of all, they need to upgrade their TLS implementations to support the TACK extension. Second, they must execute their duties very carefully, in particular with respect to tack life-cycle management. We know from Chapter 4 that certification misconfigurations are quite common and it seems not implausible to assume that TACK configurations will often suffer the same fate. Web servers number in the high millions—it is quite likely that a certain fraction would exhibit TACK misconfigurations.

However, one factor may come to TACK's aid here: the RFC requires a client to drop a connection for which it cannot confirm an existing pin. This gives a site operator—in particular the operator of an important site—a strong incentive to carry out proper life-cycle management. Larger sites generally also have the necessary funds to finance such activities. Hence, TACK's chances of deployment are greatest for popular sites, which is in-line with its primary purpose of protecting clients. The fact that it can be rolled out gradually and does not require an opt-in from all WWW servers helps, too.

8.3. Storing certification information in the DNS

Two recent proposals use the Domain Name System (DNS) to store cryptographically secured information about the certification status of a domain: *Certification Authority Authorization (CAA)* and *DNS-based Authentication of Named Entities: TLS Anchor (DANE-TLSA)*.

The DNS can be used to store arbitrary data about a domain, and several RFCs exist that exploit this fact. For example, RFC 6594 [121] defines a way to store the fingerprints of SSH host keys in the DNS. RFC 4398 [107] defined an early way to store certificates and OpenPGP keys in the DNS (although it seems to be mostly forgotten today). CAA and DANE-TLSA are the latest entries, and they explicitly address X.509 and make use of the Domain Name System Security Extensions (DNSSEC). We thus chose them as the two most relevant approaches to study. To better explain their use, we first give a short introduction to DNSSEC.

8.3.1. DNSSEC

DNSSEC is defined in a set of RFCs, of which [98, 90, 91, 110] may be the most prominent ones. The goal of DNSSEC is to add integrity protection to the otherwise insecure transmission of DNS replies. DNSSEC can be applied to all types of resource records.

DNSSEC defines a PKI for its purpose. The PKI structure follows the zone hierarchy of the DNS and starts with the root zone. The root zone is the trust anchor of the PKI—a single TTP. Every DNS zone is associated with two public/private key pairs: a Zone Signing Key (ZSK) and a Key-Signing Key (KSK). The ZSK is used to sign the records in the zone, plus the ZSK itself. The KSK is used to sign the ZSK. This scheme allows easier key roll-over and makes it possible to keep ZSKs at the length that is currently required to be cryptographically secure. This is beneficial for performance. Delegation of authority, from parent zone to child zone, is enabled in DNSSEC by having the parent zone publish the key that is used in a child zone and sign the corresponding resource records. This scheme propagates down the entire DNS hierarchy.

When a resolver queries the DNS, it has to verify the chain of signed delegations linking up to the root zone. As the public key of the root zone is the trust anchor of the DNSSEC PKI, it must be distributed out-of-band to all verifiers.

DNSSEC is a relatively complex technology as caching and transmission issues must be taken into account. Deployment of DNSSEC has been very slow, in fact, and has introduced new sources of failures. Lian *et al.* showed this in their study of 2013 [50]. They found that DNSSEC-enabled resolvers are rare (they determined an upper bound of 2.7%) and that failure rates were slightly higher compared to normal DNS lookup. One source for the failures could be determined to be the larger size of DNSSEC replies. However, the authors also found a relatively strong geographic dependency, which relativises the results to some degree. The size of DNSSEC records, however, has been criticised previously as it could help attackers stage amplification attacks in a denial-of-service attack [143].

Concerning deployment, it is also of note that many DNS stub resolvers on client computers currently do not validate DNSSEC records themselves. This task is thus left to the resolvers, which may be positioned in an ISP's network. This introduces a certain weakness on the 'last mile' to the client.

The technical complexity of DNSSEC is not the only concern. The DNS is a database that is distributed over many countries and legislations, each with their own interests. To address concerns that one such country may abuse its power, the cryptographic material for the root zone was generated in two key ceremonies in 2010, with a number of so-called Trusted Community Representatives[4] present to oversee the correctness of the process [204].

The root zone, however, may be the lesser concern. Countries and governments exercise legal control over their own top-level domains (and thus the corresponding zones). This allows them to alter DNSSEC-protected records. Since it is quite common for organisations and companies to own domains outside their own legal jurisdictions, such national control over their resource records makes them vulnerable to what one could call an 'attack by legal means'. This attack is particularly troublesome as it can easily be a tool to censor free access to the Internet. Soghoian and Stamm described the same attack for X.509 certificates in [67]: they warned that governments might be willing to compel a CA operating within their jurisdiction to issue rogue certificates for the purpose of interception of encrypted traffic.

[4]The Trusted Community Representatives were chosen by ICANN after a solicitation process among the 'Internet community' [203].

Listing 7 Service abstraction for DNSSEC.

```
Participants:
    DNSServers : {DNS_0, DNS_1, ...}                     ▷ Set of all DNS servers

Record RR:                                               ▷ Signed resource record
    type : String
    value : String
    sig : sig(String|String|String)         ▷ On domain name, RR type, and value

Service DNSSEC represents DNSServers:
    Channels:
        Chs : {Channel(q, d) : q ∈ (Clients ∪ Domains), d ∈ DNSServers}
    Init:
        $(k, k^{-1})$ ← new Key_Pair()
        domains ← new Map⟨String, Map⟨String, RR⟩⟩()
    Actions:
        Event Ch ∈ Chs: Ch.recv('store', d: String, rr_type: String, val: String):
            ▷ Signature on domain name, type and value:
            rr_sig ← $\mathrm{sig}_{k^{-1}}$(d|rr_type|val)
            ▷ Create resource record:
            rr ← new RR(rr_type, val, rr_sig)
            ▷ Store resource record:
            domains[d][type] ← rr
        Event Ch ∈ Chs: Ch.recv('load', d: String, type: String):
            ▷ Retrieve resource record:
            rr ← domains[d][type]
            Ch.send(rr)

Protocol ... (...):
    Actor ... :
        Init:
            $k_{dns}$ ← DNSSEC.k          ▷ Actors need to know DNSSEC's public key
```

8.3.2. Representing DNSSEC in our notation

We showed in the last chapter how the DNS can be represented as a *service* in our notation. Adding DNSSEC does not require many changes. The important thing to note is that, for the purpose of our notation, we do not actually need to represent the complex resolution and validation steps that a resolver carries out. The only functionality we need to model is a service that signs and stores resource records, and allows to retrieve them on request. In assessing a scheme, we can then discuss its security by allowing this service to be compromised or not.

In our notation, we thus treat DNSSEC just like DNS, except that we assign it a key pair to sign resource records. The public part of this key pair needs to be distributed to all entities participating in the scheme. We show this in Listing 7.

8.3.3. Certification Authority Authorization (CAA)

The first DNS-based scheme we describe is CAA. CAA defines a mechanism whose primary purpose is to support the verification processes of CAs. It is specified in

RFC 6844 [104]. The specification recommends to protect the CAA records with DNSSEC signatures, but does not actually mandate it.

CAA requires a domain owner, who is assumed to be in control of the corresponding DNS resource records, to add one or more CAA resource records to the existing resource records. A CAA record stores three so-called *properties*:

issue This property allows to specify who may issue certificates for the domain in question. The value is meant to be a unique identifier for a CA.

issuewild This property allows to define who may issue wild card certificates for the domain, i.e., certificates with wild cards denoting possible subdomains.

iodef This property allows to specify a URL where CAs violating the rules specified in the previous two records can be reported (i.e., CAs issuing a certificate for the domain despite the CAA record pointing to another CA).

The information that CAA provides can be quite useful for CAs. Recall the case of StartSSL in 2008 (see Section 3.3, p. 37): the CA was able to thwart an attack because it kept a list of high-value domains and treated certificates for domains from this list differently. CAA can be seen as a generalisation of this approach.

8.3.4. CAA operation and representation in our notation

Listings 8 and 9 show the CAA scheme. We only show the property *issue* in our listing. The property *issuewild* is really only an extension for subdomains and does not add any new insight on a semantic level[5]. Concerning the property *iodef*, CAA does not specify under which circumstances, or how, a CA should report a violation.

The protocol flow of CAA is simple: the only changes are in the certification process. The domain owner publishes the CAA records in the DNS (line 46). The remaining process remains unchanged from his point of view: he requests a certificate from the CA and waits for the reply.

The CA, on the other hand, must retrieve the CAA record when it receives a certification request for the domain. We show the process in lines 60–69. It must verify that the signature in the CAA record is correct and that its own name is stored as a value in the record. The remaining steps remain the same. Note that, in particular, the entire definition of TLS remains the same.

8.3.5. Assessment of CAA

In the following, we assess CAA's contributions to the security of X.509 according to the criteria we defined. We determine its robustness in our attacker models and assess potential deployment issues.

Reinforcement of X.509

CAA makes the following contributions to X.509.

Out-of-band mechanism CAA gives CAs an additional mechanism to include in their operational practices when determining whether to issue a certificate for a given domain. Security for clients is only indirectly improved. However, if all CAs followed the procedures of CAA, the bar for attacks would be raised.

[5]The property can be modelled by introducing string matching to the notation, e.g., by introducing a well-known process as an abstraction of pattern matching with regular expressions.

Listing 8 Scheme of CAA, Part 1.

```
Scheme CAA:
    Participants:
        CAs : {Ca_0, Ca_1, ...}                                    ▷ Set of all CAs
        Domains : {D_0, D_1, ...}                                  ▷ Set of all domains
        Clients : {Cl_0, Cl_1, ...}                                ▷ Set of all clients
        DNSServers : {DNS_0, DNS_1, ...}                           ▷ Set of all DNS servers

    Record RR:                                                     ▷ Signed resource record
        type : String
        value : String
        sig : sig(String|String|String)          ▷ On domain name, RR type, and value

    Init:
        for Ca ∈ CA:
            (Ca.k, Ca.C.k^-1) ← new Key_Pair()
        for D ∈ Domain:
            (D.k, D.k^-1) ← new Key_Pair()
        for Cl ∈ Clients:
            ... ▷ Normal TLS initialisation

    Service DNSSEC represents DNSServers:
        Channels:
            Chs : {Channel(q, d) : q ∈ (CAs ∪ Domains), d ∈ DNSServers}
        Init:
            (k, k^-1) ← new Key_Pair()
            domains ← new Map⟨String, Map⟨String, RR⟩⟩()
        Actions:
            Event Ch ∈ Chs: Ch.recv('store', d: String, rr_type: String, val: String):
                ▷ Signature on domain name, type and value:
                rr_sig ← sig_{k^-1}(d|rr_type|value)
                ▷ Create resource record:
                rr ← new RR(rr_type, value, rr_sig)
                ▷ Store resource record:
                domains[d][type] ← rr
            Event Ch ∈ Chs: Ch.recv('load', d: String, type: String):
                ▷ Retrieve resource record:
                rr ← domains[d][type]
                Ch.send(rr)
```

Incident detection CAA provides a clearly designated point of reference where third parties (domain owners, clients, browser vendors) can report violations of the rules specified in the CAA record: the *iodef* property. This makes it easier to take appropriate steps if a violation is detected. However, CAA does not define a process nor a mechanism for the actual detection mechanism. This is unfortunate as the detection could be easily automated. It would be worthwhile to scan both DNS records from domain lists (e.g., from zone files) and then certificates from TLS connections.

Listing 9 Scheme of CAA, Part 2.

```
Protocol Certification((D,Ca)):
    Actor D:
        Channels:
            DNSCh: Channel(D, DNSSEC)                          ▷ Domain ↔ DNS
            SecCh: Sec_Channel(D, Ca)                          ▷ Domain ↔ CA
        Actions:
            Event Start:
                DNSCh.send('store', name, 'CAA', Ca.name)
                SecCh.send(name, k)
                state ← 'Wait_Cert'
            Event state = 'Wait_Cert' and SecCh.recv($cert_{new}$: Cert):
                cert ← $cert_{new}$
                state ← 'Start'

    Actor Ca:
        Channels:
            DNSCh: Channel(Ca, DNSSEC)                         ▷ Ca ↔ DNS
            SecCh: Sec_Channel(Ca, D)                          ▷ CA ↔ domain
        Init:
            $k_{dns}$ ← DNSSEC.k
        Actions:
            Event state = 'Start' and SecCh.recv($name_d$: String, $k_d$: Pub_Key):
                DNSCh.send('load', $name_d$, 'CAA')
                state ← 'Wait_CAA'
            Event state = 'Wait_CAA' and DNSCh.recv($rr_d$: RR):
                if $rr_d$.type = 'CAA'  and $rr_d$.value = name
                  and $valid\_sig_{k_{dns}}$($name_d$|$rr_d$.type|$rr_d$.value, $rr_d$.sig)
                  and meets_cps($name_d$):
                    cert ← new $Cert_{k^{-1}}$($name_d$, $k_d$)
                    SecCh.send(cert)
                    state ← 'Start'

Protocol TLS_DHE_CAA(Cl, D):
    . . .
    ▷ There are no changes to the TLS protocol.

Sessions:
    Dependency TLS_DHE_CAA(*,*) not before Certification(*,*)
    At random Certification with $(D, Ca) \in Domains \times CAs$
    At random TLS_DHE_CAA with $(Cl, D) \in Clients \times Domains$
```

Transparency by observation CAA is not concerned with monitoring X.509 deployment nor with certification. However, it is a relatively transparent scheme: misconfigurations of CAA can be easily detected with active scans by comparing a site's X.509 certificate with the CAA record from the DNS.

Robustness against attackers

The protection that CAA can offer depends on the strength of the attacker. A particular weakness of CAA is that it does not mandate the use of DNSSEC.

Consider the weak attacker from our Model A. There are two ways how this attacker might succeed. Either he is in control of the channel over which a domain owner sets the CAA record, or he controls the channel over which a CA communicates with the respective DNS servers to query the CAA record. In the first case, he would have to be a man-in-the-middle between the DNS server and the administrator. This may be possible in the case of hosting providers operating nameservers for their customers and allowing them to change records via a Web interface. In such a setting, the attacker would have to intercept the communication with the Web interface, which is only possible if it is not secured (or assuming an administrator would ignore a warning in his browser about an invalid certificate). This attack seems somewhat unlikely to succeed, although it cannot be ruled out entirely. The second option for the attacker would be to attack the communication between a CA and the responsible nameserver. One way to achieve this would be to compromise the gateway host of a CA. This seems not very likely as CAs have particularly good reasons to invest in the security of their infrastructure. The more likely option may be that the attacker is able to compromise the host where the CAA record is stored, i.e., the nameserver, or poison the cache of a non-authoritative nameserver that is queried first. Both attacks could be thwarted with DNSSEC. The conclusion to draw here is that CAA works well against the weak attacker from our Model A. The attacker has very little to no chance of success, in particular if DNSSEC is used.

CAA loses much of its security in our other two threat models, however. Consider the attacker in our Model B: this attacker may compromise a CA of his choosing. This means he can simply bypass the CAA checks and issue a rogue certificate, which trivially breaks CAA. But even if we disallow the attacker to compromise a CA to this degree, there are still ways for him to defeat CAA. One attack vector is directed against CAs inside the attacker's own country: the attacker controls the necessary paths to tamper with their traffic. If DNSSEC is used, he is limited to such domains that are in the top-level domain he operates. CAs outside the attacker's country are at risk when they need to issue a certificate for a domain whose DNS entries are served by a nameserver to which the attacker controls the network path. If DNSSEC is used, the attacker is again limited to his control over certain zones.

CAA starts to fall apart entirely when confronted with our globally acting attacker of Model C. We allow this attacker to reroute DNS traffic. He is thus able to change the value of the CAA record to a CA that he also controls. DNSSEC is not a good barrier, either, as we allow the attacker to compromise several zones that are outside his direct organisational control.

The conclusion here is that CAA is only a protection against the weaker kinds of attackers. It is insightful to relate this to actual events. In the more devastating incidents like Comodo (2011) or DigiNotar (2011), the CAs' issuance process was compromised. CAA would not have provided any security.

Deployment

CAA has the advantage of being a very simple concept with low implementation costs. It can also be rolled out gradually, and DNS operators could opt in once their software supports the new record. CAA does not require any changes to the TLS protocol, either. This eases deployment and makes it possible for early adopters to create CAA records. Indeed, at the time of writing, Google has begun to deploy CAA records for their domains[6] [209]. Although the number of nameservers that need updates may be high, the overall conclusion here is that CAA encounters very few barriers to deployment. At the same time, it raises the barriers for attackers and, more importantly, makes it possible to establish a better incident detection.

8.3.6. DNS-based Authentication of Named Entities: TLS Anchor (DANE-TLSA)

DANE is a working group within the IETF. According to its charter [174], its objective is to define a method to store bindings between names and public keys in the DNS, secured with DNSSEC. The first proposal the working group published is called DANE-TLSA and is specified in [105].

The corresponding resource record—TLS Anchor (TLSA)—defines an alternative way to verify a certificate. The record can store either a certificate or a public key, and several options to carry out the verification process are defined. TLSA does neither mandate nor prohibit the usual CA certification processes. Domains can have their certificates issued normally from a CA and use TLSA as an additional trust anchor or they may simply use certificates that are either self-signed or from a CA that is not part of any root store.

The resource record has fairly simple semantics. A record consists of the following four fields.

Certificate Usage This field indicates what kind of certificate is defined in the trust anchor. There are four options.

- The first is to specify the certificate or public key of a CA. When clients verify the certificate chain, this certificate or public key must appear in it. The record is useful to limit which CAs may issue certificates for a domain and can thus be used as a stand-in for CAA.
- The second option is to specify the certificate or public key of an end-entity, i.e., commonly a server. Such a certificate is assumed to have been issued by a CA. The option allows to define which certificate is considered valid for a domain.
- The third option is to specify the certificate or public key of the root certificate that is to be used when verifying the certificate chain. This option is meant to be used when a domain uses certificates from a CA that is, e.g., not generally recognised by root stores.
- The last option is to specify an end-entity's certificate without requiring issuance by any CA. This option can be used for self-signed certificates.

Selector This field indicates whether a complete certificate is stored or just the public key. The advantage of storing a public key instead of the certificate is that the TLSA record does not have to be updated when a certificate is renewed (with the same public key).

Matching Type This field indicates whether the full certificate or public key is specified in the resource record or whether a hash value is used.

Certificate Association This field contains the certificate or public key to which the domain name is bound.

[6]The command `dig +short -t TYPE257 google.com`, available on GNU/Linux systems, shows this.

8.3.7. DANE-TLSA operation and representation in our notation

The RFC for TLSA leaves the choice when to send the query for the TLSA record to the client. In our listings 10–12, we show the case of querying the record during the TLS handshake. We do not model all of the different options described above but model only the case where the hash value of the certificate (chain) is stored in the TLSA record. As a simplification, we do not distinguish between certificates and certificate chains. We also omit a potential fallback to normal TLS in case no usable TLSA information is received by the client.

Our listings show the creation of the TLSA record as part of the certification process. Certification is the normal process as we know it from TLS (line 40 ff.)—also see listings 2 and 3. A domain sets the TLSA record once it has received its certificate (line 50). We model this by sending the domain name and the hash value of the certificate to the DNSSEC service.

Lines 64–81 show which changes occur on client side when using TLSA. The flow of the TLS protocol remains unchanged up to the point when the client receives the server certificate. The client verifies the correctness of the certificate by retrieving the TLSA record from the DNSSEC service. Once the record is received, the client proceeds with the verification process of the server certificate. Note that we do not verify the server certificate before the client receives a reply from the DNSSEC service. The motivation is to avoid any computation until the record is received. The verification process consists of the normal server certificate verification and the verification of the TLSA record, i.e., whether it is issued for the correct domain name, carries the correct hash value of the server certificate, and is correctly signed.

If the verification process succeeds, the TLS connection continues as in normal TLS. Note that there are no changes at all on server side.

8.3.8. Assessment of DANE-TLSA

In the following, we present our assessment of DANE-TLSA.

Reinforcement of X.509

DANE-TLSA makes the following contributions to the security of X.509.

Out-of-band mechanism The purpose of DANE-TLSA is to provide confirmation of a server certificate over a second, supposedly secure channel. This is exactly the out-of-band mechanism that we identified as a necessary step to improve the security of X.509 for clients (see Chapter 3). In contrast to CAA, the protection is a direct one: domains set the record, clients use it. DANE-TLSA also mandates DNSSEC; thus an attacker's options are limited to attacking DNSSEC itself.

Incident detection DANE is not designed to report detected mismatches of TLSA records and server certificates. However, incidents could be uncovered more easily with a monitoring infrastructure that scans both DNS entries as well as server certificates. DANE-TLSA lacks an important property of CAA, namely a way to define a point of reference to report incidents.

Transparency by observation DANE-TLSA addresses neither X.509 certification nor deployment. But it is a very transparent scheme, just like CAA. DANE-TLSA requires the certification status of a domain to be closely synchronised with up-to-date DNS records. Monitoring would be beneficial in uncovering unintentional mismatches.

Listing 10 Scheme for DANE-TLSA, Part 1.

```
Scheme TLSA:
    Participants:
        CAs : $\{\mathrm{Ca}_0, \mathrm{Ca}_1, \ldots\}$                         ▷ Set of all CAs
        Domains : $\{\mathrm{D}_0, \mathrm{D}_1, \ldots\}$                       ▷ Set of all domains
        Clients : $\{\mathrm{Cl}_0, \mathrm{Cl}_1, \ldots\}$                     ▷ Set of all clients
        DNSServers : $\{\mathrm{DNS}_0, \mathrm{DNS}_1, \ldots\}$                ▷ Set of all DNS servers

    Record RR:                                              ▷ Signed resource record
        type : String
        value : String
        sig : sig(String|String|String)          ▷ On domain name, RR type, and value

    Init:
        for Ca ∈ CA:
            (Ca.k, Ca.$k^{-1}$) ← new Key_Pair()
        for D ∈ Domain:
            (D.k, D.$k^{-1}$) ← new Key_Pair()
        for Cl ∈ Clients:
            . . . ▷ Normal TLS initialisation

    Service DNSSEC represents DNSServers:
        Channels:
            Chs : {Channel(q, d) : q ∈ (CAs ∪ Domains), d ∈ DNSServers}
        Init:
            (k, $k^{-1}$) ← new Key_Pair()
            domains ← new Map⟨String, Map⟨String, RR⟩⟩()
```

Robustness against attackers

We assess the security of DANE-TLSA in our three threat models. The first thing to note is that the security of DANE-TLSA does not just depend on the strength of the attacker, but also on whether a client uses a validating stub resolver (i.e., validates a TLSA record itself) or whether this is done by the DNS resolver via which it queries DNS records. The difference between these two cases is fundamental.

Let us consider the question of the stub resolver first. Consider our weakest attacker, from Model A, in a scenario where he has control over a gateway, and the client computer uses one of today's off-the-shelf operating systems. At least at the time of writing, these operating systems do not yet provide DNSSEC-validating stub resolvers, and thus the DNS resolvers of the client's ISP are queried. Neither query nor reply are protected in any way; and the attacker can rewrite any DNS reply. DANE-TLSA can thus only be secure if validating stub resolvers are used. An alternative would be to use methods like TSIG [123], which establish an integrity-protected channel between stub resolver and recursive resolver. Otherwise, DANE-TLSA fails even in the weakest threat model.

In the following, we assume a client with a validating stub-resolver or an equivalent protection. We also assume that a missing or incorrect DNS reply causes our client to abort the TLS connection. The security of TLSA depends now entirely on whether an attacker is able to change DNS entries. Our attacker from Model A is powerless. The situation in the other two threat models is somewhat different, however.

Listing 11 Scheme for DANE-TLSA, Part 2.

```
        Actions: ▷ Define actions for service
            Event Ch ∈ Chs: Ch.recv('Store', d: String, rr_type: String, val: String):
                ▷ Signature on domain name, type and value:
                rr_sig ← $sig_{k^{-1}}$(d|rr_type|val)
                ▷ Create resource record:
                rr ← new RR(rr_type, val, rr_sig)
                ▷ Store resource record:
                domains[d][type] ← rr
            Event Ch ∈ Chs: Ch.recv('Load', d: String, type: String):
                ▷ Retrieve resource record:
                rr ← domains[d][type]
                Ch.send(rr)

    Protocol Certification((D,Ca)):
        Actor D:
            Channels:
                DNSCh: Channel(D, DNSSEC)                    ▷ D ↔ DNS server $d_D$
                SecCh: Sec_Channel(D, Ca)                     ▷ domain ↔ CA
            Actions:
                Event Start:
                    SecCh.send(name, k)
                    state ← 'Wait_Cert'
                Event state = 'Wait_Cert' and SecCh.recv($cert_{new}$: Cert):
                    DNSCh.send('store', name, 'TLSA', h(cert))
                    cert ← $cert_{new}$
                    state ← 'Start'

        Actor Ca:
            Channels:
                SecCh: Sec_Channel(Ca, D)                     ▷ CA ↔ domain
            Actions:
                Event SecCh.recv($name_d$: String, $k_d$: Pub_Key):
                    if meets_cps($name_d$):                   ▷ Check against CPS
                        cert ← new $Cert_{k^{-1}}$($name_d$, $k_d$)
                        SecCh.send(cert)

```

Our regional attacker from Model B may compromise any CA he chooses. However, his control over DNSSEC-secured entries is restricted to the zones over which he is given control. Thus he can only attack domains within these zones. DANE-TLSA provides protection to all other zones.

The position of our globally active attacker is better. His control over DNSSEC extends to several other zones outside his direct organisational control. This means such an attacker can do damage outside his own region. As we also allow him to reroute traffic, this attacker is powerful enough to carry out man-in-the-middle attacks that cannot be prevented by TLSA.

To summarise, DANE-TLSA provides a high degree of security against all attackers that have less resources than what may be associated with countries. However, DANE-TLSA can be compromised by regionally active attackers (in control of DNS) or the

Listing 12 Scheme for DANE-TLSA, Part 3.

```
Protocol TLS_DHE_TLSA((Cl,D)):
    Actor Cl:
        Channels:
            Ch: Channel(Cl, D)
        Init:
            $k_{dns} \leftarrow$ DNSSEC.k
        Event Start:
            ... ▷ Normal TLS until server cert arrives
        Event state = 'Wait_Kex' and
         Ch.recv($n_D$: Nonce, $cert_D$: Cert, $dh_D$: DHParm, sgn: String):
            DNSCh.send('load', D.name, 'TLSA')
            state ← 'Wait_TLSA'
        Event state = 'Wait_TLSA' and DNSCh.recv($rr_D$: RR):
            if valid_cert($cert_D$)  and $valid\_sig_{cert_D.k}(n_C|n_D|dh_D, sgn)$
              and $rr_D$.type = 'TLSA'  and $rr_D$.value = h($cert_D$)
              and $valid\_sig_{k_{dns}}$(D.name|'TLSA'|h($cert_D$), $rr_D$.sig):
                ... ▷ Normal TLS continues
            else: ▷ Else, we abort:
                state ← 'Error'

    Actor D:
        ... ▷ No changes to server required

Sessions:
    Dependency TLS_DHE_TLSA(*,*) not before Certification(*,*)
    At random Certification with $(D, Ca) \in$ Domains × CAs
    At random TLS_DHE_TLSA with $(Cl, D) \in$ Clients × Domains
```

globally acting attacker. The potentially more interesting question here is whether attacks would be detectable and whether attackers are willing to take the risks that are associated with detection. A monitoring infrastructure that employs several vantage points would be very useful—it would be able to detect whether DANE-TLSA records change suddenly and whether the certificates also change, as determined by all vantage points. If misconfigurations can be excluded as a reason, this would be an indication of compromise.

Deployment

Like CAA, DANE-TLSA has the advantage of relatively low deployment costs. It can be rolled out gradually and does not require changes to the TLS protocol itself. It requires changes to common implementations to support the additional queries and evaluation of DNS records, however. On the whole, it seems unlikely that either of these constitutes a serious barrier.

A more pressing question may be whether DNS servers support the new record soon enough. With DNSSEC deployment still relatively slow (see, e.g., the study by [50]), DANE-TLSA is not a solution that can be useful on a large scale immediately.

Our biggest concern, however, is that TLSA records need to be closely synchronised with a domain's current certification state. This is particularly true when hash values of certificates are stored in the resource record rather than hash values of public keys.

Every time a domain changes its server certificate, it needs to update its DNS entries immediately. If only the public key is changed, this need may occur more rarely. However, forgetting to set the TLSA record or failure to propagate it within the DNS may cause domains to be unreachable for a brief period of time.

Since DNSSEC is not yet widely deployed, one can only speculate if this may become an issue or not. Recall, however, that we have data points from our analysis of the SSHFP resource record, whose functionality is the same as TLSA's: we found about 6% of records were inaccurate. This would be a rather high number in the case of the WWW, especially with clients aborting connections on encountering a mismatch between certificate and TLSA record.

The conclusion we draw here is that deployment of DANE-TLSA shows a strong dependency on DNSSEC and must be done with great care. Organisations wishing to use DANE-TLSA should consider bundling the responsibility for DNS and TLS in one administrative team, with the necessary deployment steps defined precisely to avoid mistakes.

8.4. Notary concepts

In this section, we discuss the so-called notary concept by example of *Perspectives* [77], the first such published approach. The fundamental assumption underlying any notary concept is that an attacker is extremely unlikely to control all or almost all routing paths on the Internet. Notary concepts propose well-known servers, distributed globally, whose public keys have been distributed out-of-band by some mechanism. Clients can query notaries for certain assessments. In the case of TLS and Perspectives, such an assessment could be the public key a notary has stored for a given domain. The assessment has the function of a 'second opinion' that the client can request every time it connects to the domain in question.

8.4.1. Choice of Perspectives as subject to study

To the best of our knowledge, Perspectives was the first published notary concept and the first to deliver source code [247]. It employs the concept described above, with some optimisations to achieve better performance. It also features a rather elaborate and interesting cross-validation mechanism.

Perspectives is currently not the only notary concept. In 2011, Marlinspike presented Convergence [268]. The fundamental concept of Convergence is not different from Perspectives (on which it explicitly builds). Convergence adds some improvements and optimisations. First, Convergence follows a stronger on-demand approach (a notary only connects to a domain when explicitly queried by a client). Second, Convergence notaries are free to provide their service for other properties of a domain, not just certificates (e. g., DNS records). Third, Convergence puts more emphasis on privacy and uses so-called notary bouncing, i. e., clients use one notary as a forwarder to others. This is a simplified form of onion routing [29]—in this particular case, it can be circumvented if forwarder and responding notary collude.

For the purpose of this section, we choose Perspectives as it is the original concept and detailed design documents exist. There is no design document for Convergence.

8.4.2. Operation and representation in our notation

The participants in Perspectives are domains, clients and notaries. This makes Perspectives one of the concepts that can be employed with and without CAs. As described above, notaries are entities that can be queried for information about remote domains.

Listing 13 Scheme of Perspectives, Part 1.

```
 1: Scheme Perspectives:
 2:     Participants:
 3:         Domains : {D_0, D_1, ...}                ▷ Set of all domains
 4:         Clients : {Cl_0, Cl_1, ...}                ▷ Browsers, etc.
 5:         Notaries : {N_0, N_1, ..., N_n}    ▷ Limited number of notaries
 6:
 7:     Record Period:                        ▷ Represents a time interval
 8:         min : Timestamp                               ▷ Start of period
 9:         max : Timestamp                                 ▷ End of period
10:
11:     Record Observation:        ▷ Holds all observation periods for a key
12:         k : Pub_Key                ▷ Public key this observation refers to
13:         pds : {Period}                            ▷ Observation periods
14:         sig : sig(Pub_Key|{Period})                         ▷ Signature
15:
16:     Record Obs_Sig:             ▷ Signed observations—shadows use this
17:         obs : {Observation}
18:         sig : sig({Observation})
19:
20:     Init:
21:         for N ∈ Notaries:
22:             (N.k, N.k^-1) ← new Key_Pair()
23:             history ← new Map⟨String, {Observation}⟩()
24:         for D ∈ Domains:
25:             D.cert ← new Cert()
26:         for Cl ∈ Clients:
27:             ... ▷ Initialisation as in normal TLS
```

Clients choose from the available notaries which ones they want to trust with ‘second opinions’. Listings 13–18 describe Perspectives in our notation.

The role of notaries in Perspectives is to store a history of *observations* for a number of domains. The term observation refers to occurrences of the same public key in server certificates, for a given domain. Observations are stored as a set of periods (lines 7–9) during which the notary observed only the public key in question for the given domain. Every notary has its own key pair (lines 21–23). We model observations as a record which holds periods and a signature (lines 11–14).

Notaries scan a list of domains from time to time and store their observations. This behaviour is intended to reduce their reaction time for client queries—they do not have to scan on-demand when a client queries them. The original paper does not explain which domains a notary is expected to scan or whether a client can request a notary to include a domain in future scans. We thus do not model it here either, but note that it would be easy to add.

We describe this continuous scanning as a *Protocol* in our notation (see Listing 16). The protocol is called at random, but before any run of TLS (lines 252–257 in Listing 18). Notaries do not validate their observations in any way—their role is that of a neutral party that is simply logging what it encounters. Observations for new domains (i.e., those previously not scanned) are stored directly when they are made (lines 136–140).

Listing 14 Scheme of Perspectives, Part 2.

```
Procedure sane_obs(obs: {Obversation}):
    pds ← $\bigcup_{o \in obs}$ o.pds
    if ($\forall p \in pds : p.min \leq p.max$):
        return true
    else:
        return false

Procedure overlap_obs(obs: {Obversation}):
    pds ← $\bigcup_{o \in obs}$ o.pds
    if ($\exists\, p_1, p_2 \in pds, p_1 \neq p_2 : p_1.min \leq p_2.max$):
        return true
    else:
        return false

Procedure consistent_obs(obs$_{old}$: {Obversation}, obs$_{new}$: {Obversation}):
    for $o_{old} \in obs_{old}$:
        $o_{new} \leftarrow (o \in obs_{new} : o_{old}.k = o.k)$          ▷ Get observation for same key
        if $o_{new} = \varnothing$:          ▷ Old observation for k missing in new observations
            return false
        else:                                     ▷ Check consistency of periods:
            $pds_{old} \leftarrow o_{old}.pds$
            $pds_{new} \leftarrow o_{new}.pds$
            for $p \in pds_{old}$:                   ▷ Old periods contained in new ones?
                if ($\nexists\, \bar{p} \in pds_{new} : (p.min = \bar{p}.min) \wedge (\bar{p}.max \geq p.max)$):
                    return false
    return true

Service Notarising represents Notaries:
    Channels:
        Cl_Chs : {Channel(cl, n) : cl ∈ Clients, n ∈ Notaries}        ▷ With clients
        S_Chs : {Channel(n, s) : n ∈ Notaries, s ∈ Notaries}         ▷ With shadows

    Actions:
        ▷ Return shadows of all notaries indicated by name:
        Event Ch ∈ Cl_Chs: Ch.recv('get_shd', names: {String}):
            shd_map ← new Map⟨String, {String}⟩()
            ▷ Get notaries with these names:
            for $N \in \{\bar{N} \in Notaries : \bar{N}.name \in names\}$:
                shd_map[N.name] ← N.shadows
            Ch.send(shd_map)
        ▷ Return histories of all notaries indicated by name, for a domain:
        Event Ch ∈ Cl_Chs:
          Ch.recv('get_hst', name$_D$: String, names: {String}):
            hst_map ← new Map⟨String, {Observation}⟩
            ▷ Get notaries with these names:
            for $N \in \{\bar{N} \in Notaries : \bar{N}.name \in names\}$:
                hst_map[N.name] ← N.history[name$_D$]
            Cl_Chs.send(hst_map)
```

Listing 15 Scheme of Perspectives, Part 3.

```
Service Shadows represents Notaries:
    Channels:
        Cl_Chs : {Channel(cl, s) : cl ∈ Clients, s ∈ Notaries}
        N_Chs : {SecChannel(n, s) : n ∈ Notaries, s ∈ Notaries}

    Init:
        $(k, k^{-1})$ ← new Key_Pair()        ▷ Shadows key; used to sign stored data
        ntr_keys ← ∅                              ▷ Notaries' public keys
        for N ∈ Notaries:
            ntr_keys[N.name] ← N.k
        ▷ Notary observations: Map⟨Notary, Map⟨Domain, Observations⟩⟩
        ntr_obs_sig ← new Map⟨String, Map⟨String, Obs_Sig⟩⟩()

    Actions:
        Event Ch ∈ N_Chs: Ch.recv('store', $name_N$: String, $name_D$: String,
                                  $obs_{new}$: {Observation}):
            ▷ We only act if all signatures are correct:
            $k_N$ ← ntr_keys[$name_N$]
            if ($\forall o \in obs_{new} : valid\_sig_{k_N}(o.k|o.pds, o.sig)$) :
                obs_map ← ntr_obs_sig[$name_N$]
                ▷ No entry for notary—just store:
                if obs_map = ∅:
                    obs_map ← new Map⟨String, Obs_Sig⟩()
                    obs_map[$name_D$] ← new Obs_Sig($obs_{new}$, $sig_{k^{-1}}(obs_{new})$)
                    ntr_obs_sig[$name_N$] ← obs_map
                ▷ No previous observations for domain?
                else if obs_map[$name_D$] = ∅:          ▷ None? Just store.
                    obs_map[$name_D$] ← new Obs_Sig($obs_{new}$, $sig_{k^{-1}}(obs_{new})$)
                    ntr_obs_sig[$name_N$] ← obs_map
                ▷ Else, do consistency checks with old observations:
                else:
                    $obs_{old}$ ← obs_map[$name_D$]
                    if sane($obs_{new}$) and ¬overlap($obs_{new}$)
                          and consistent($obs_{old}$,$obs_{new}$):
                        ▷ Replace old observations:
                        obs_map[$name_D$] ← new Obs_Sig($obs_{new}$, $sig_{k^{-1}}(obs_{new})$)
                    else:                               ▷ Checks failed—store proof
                        $obs_{proof}$ ← $obs_{new} \cup obs_{old}$
                        obs_map[$name_D$] ← new Obs_Sig($obs_{proof}$, $sig_{k^{-1}}(obs_{proof})$)
                    ntr_obs_sig[$name_N$] ← obs_map

        Event Ch ∈ Cl_Chs: Ch.recv('get', $name_D$: String, names: {String}):
            reply_map ← new Map⟨String, Obs_Sig⟩          ▷ Contains reply
            for N ∈ names:
                obssig_map ← ntr_obs_sig[$name_N$]        ▷ Fetch Map for notary
                reply_map[N] ← obssig_map[$name_D$]       ▷ Add observations
            Ch.send(reply_map)
```

Listing 16 Scheme of Perspectives, Part 4.

```
Protocol Monitoring(N, $\{N_{s,1}, \ldots, N_{s,j}\}$):
    Actor N:
        Channels:
            Shd_Ch : SecChannel(Cl, Shadows)

        Init:
            shadows ← $\{N_{s,1}.\text{name}, \ldots, N_{s,j}.\text{name}\}$

        Actions:
            Event scan_domains:
                for d ∈ Domains:
                    TLS.TLS_DHE(N,d)                ▷ N will get server cert: $\text{cert}_D$
                    $k_{seen}$ ← $\text{cert}_D$.k
                    if history[d] = ∅:                  ▷ Domain never observed
                        $p_{now}$ ← new Period(time('now'), time('now'))
                        sig ← $\text{sig}_{k^{-1}}(k_{seen}|\{p_{now}\})$
                        o ← new Observation($k_{seen}$, $\{p_{now}\}$, sig)
                        $\text{obs}_{new}$ ← {o}
                    else:                               ▷ Earlier observations exist
                        $o_{k,seen}$ ← o ∈ history[d] : o.key = $k_{seen}$
                        $\text{obs}_{else}$ ← {o ∈ history[d] : o.key ≠ $k_{seen}$}
                        if $o_{k,seen}$ = ∅:                ▷ Key has never been observed
                            $p_{now}$ ← new Period(time('now'), time('now'))
                            sig ← $\text{sig}_{k^{-1}}(k_{seen}|\{p_{now}\})$
                            o ← new Observation(k, $\{(p_{now})\}$, sig)
                            $\text{obs}_{new}$ ← $\text{obs}_{else} \cup \{o\}$
                        else:                           ▷ Key has been observed before
                            ▷ Is key the current one? Then extend period.
                            if $(\exists p \in o_{k,seen}.\text{pds} :$
                                $(p.\text{max} > \bar{p}.\text{max}\ \forall \bar{p} \in \bigcup_{\bar{o} \in \text{obs}_{k,else}} \bar{o}.\text{pds}))$:
                                ▷ Get last period:
                                $p_{last}$ ← $(p \in o_{k,seen}.\text{pds} :$
                                    $(\forall \bar{p} \in o_{k,seen}.\text{pds}, p \neq \bar{p} : p.\text{max} > \bar{p}.\text{max}))$
                                $p_{new}$ ← new Period($p_{last}$.min, time('now'))
                                $\text{pds}_{new}$ ← $\left(o_{k,seen}.\text{pds} \setminus \{p_{last}\}\right) \cup \{p_{new}\}$
                                $o_{k,seen}.\text{pds}$ ← $\text{pds}_{new}$
                                $o_{k,seen}.\text{sig}$ ← $\text{sig}_{k^{-1}}(k_{seen}|\text{pds}_{new})$
                                $\text{obs}_{new}$ ← $\text{obs}_{else} \cup \left\{\text{obs}_{k,seen}\right\}$
                            ▷ Key was seen, but is not latest one. Append period:
                            else:
                                $p_{new}$ ← new Period(time('now'), time('now'))
                                $\text{pds}_{new}$ ← $o_{k,seen}.\text{pds} \cup \{p_{new}\}$
                                $o_{k,seen}.\text{pds}$ ← $\text{pds}_{new}$
                                sig ← $\text{sig}_{k^{-1}}(k_{seen}|\text{pds}_{new})$
                                $\text{obs}_{new}$ ← $\text{obs}_{else} \cup \left\{o_{k,seen}\right\}$
                    history[d] ← $\text{obs}_{new}$
                    Shd_Ch.send('store', name, d.name, $\text{obs}_{new}$)    ▷ To shadows
```

Listing 17 Scheme of Perspectives, Part 5.

```
Protocol TLS_DHE_Perspectives((Cl, D), $\{N_{Cl,1}, \ldots, N_{Cl,i}\}$):
    Actor Cl:
        Channels:
            Ch: Channel(Cl, D)
            Ntr_Ch : Channel(Cl, Notarising)
            Shd_Ch : Channel(Cl, Shadows)

        Init:
            notaries ← $\{N_{Cl,1}.name, \ldots, N_{Cl,i}.name\}$
            ntr_keys ← ∅
            for N ∈ $\{N_{Cl,1}.name, \ldots, N_{Cl,i}.name\}$:
                ntr_keys[N.name] ← N.k
            k_shd ← Shadows.k
            key_cache ← new Map⟨String, {Pub_Key}⟩

        Actions:
            Event Start:
                ▷ Obtain list of shadows from notaries:
                Ntr_Ch.send('get_shd', notaries)
                state ← 'Wait_Shd_Lst'
            Event state = 'Wait_Shd_Lst' and Ntr_Ch.recv(shd_map: Map) :
                shadows ← shd_map
            ... ▷ Continue with normal TLS

            Event state = 'Wait_Accept' and Ch.recv(fin_mac: String):
                if $cert_D$.k ∈ key_cache[D.name]:                ▷ Key in cache?
                    ▷ TLS handshake can be finished as usual
                    ...
                else: ▷ Else, start queries to notaries
                    Ntr_Ch.send('gst_hst', notaries)
                    state ← 'Wait_Hst'
            Event state = 'Wait_Hst' and
             Ntr_Ch.recv(hst: Map⟨String, {Observation}⟩):
                responders ← ∅
                for N ∈ hst.keys():
                    $obs_N$ ← hst[N]                              ▷ N's observations
                    $k_N$ ← ntr_keys[N]
                    ▷ We only consider observations with valid signatures:
                    $obs_N \leftarrow \{o \in obs_N : valid\_sig_{k_N}(o.obs, o.sig)\}$
                    $o_{latest} \leftarrow (o \in obs_N : (\exists\ p \in o : p.max > \bar{p}.max,$
                                  $\forall \bar{p} \in \bigcup_{\bar{o} \in (obs_N \setminus \{o\})} \bar{o}.pds))$
                    if $o_{latest}.k = cert_D.k$:
                        responders ← responders ∪ {N}
```

Listing 18 Scheme of Perspectives, Part 6.

```
                if $|\text{responders}| \geq \frac{2}{3}|\text{notaries}|$:        ▷ Quorum reached?
                    responder_shd ← $\bigcup_{r \in \text{respondrs}}$ r.shadows
                    Shd_Ch.send('get', D.name, responder_shd)
                    state ← 'Wait_Shd'
                else:
                    state ← 'Error'
        Event state = 'Wait_Shd' and
              Shd_Ch.recv(reply_map: Map⟨String, Obs_Sig⟩):
            for N ∈ reply_map.keys():
                o_sig ← reply_map[N]
                if ¬valid_sig$_{k_{shd}}$(o_sig.obs, o_sig.sig):        ▷ Verify shadow signature
                    state ← 'Error'
                obs$_N$ ← hst[N]             ▷ Observations received directly from notary
                ▷ Filter for key in question:
                obs$_{shd,k}$ ← {o ∈ o_sig.obs : o.k = cert$_D$.k}
                obs$_{N,k}$ ← (o ∈ obs$_N$ : o.k = cert$_D$.k)
                ▷ More than one observation from shadow means conflict:
                if $|\text{obs}_{shd,k}| > 1$:
                    state ← 'Error'
                ▷ Test if all observations are sane
                if ¬sane_obs({obs$_{shd,k}$} ∪ {obs$_{N,k}$}):
                    state ← 'Error'
                ▷ Test if there are overlapping periods
                if ¬overlap_obs(o_sig.obs)  and ¬overlap_obs(obs$_N$):
                    state ← 'Error'
                ▷ Test if consistent between shadow and notary
                if ¬consistent_obs(obs$_{shd,k}$, obs$_{N,k}$):
                    state ← 'Error'
            ▷ Done: $\frac{2}{3}$ of notaries have reported the key, and shadows
            ▷ have corroborated their observations. Store key in cache.
            cache[D.name] ← (cache[D.name] ∪ {cert$_D$.k})
            ▷ Finish normally
            ...

        Actor D:
            Channels:
                Ch: Channel(D, Cl)
            ... ▷ No changes at all to servers!

    Sessions:
        Dependency TLS_Perspectives(∗, ∗) not before Monitoring(∗,{∗})
        At random Monitoring with
               N ∈ Notaries, {N$_{S,1}$, ..., N$_{S,j}$} ⊂ Notaries ∖ {N}
        At random TLS_Perspectives with
               (Cl, D) ∈ Clients × Domains, {N$_{Cl,1}$, ..., N$_{Cl,i}$} ⊂ Notaries
```

If earlier observations exist, the process depends on whether the domain's public key is a new one or has been seen before. In the former case, a new observation is added (lines 144–148), with start and end time of the period being the current time. In the latter case, the notary updates the observation periods, either by extending the last period (for an ongoing observation) or by appending a new one. This is shown in lines 150–167. Note that notaries sign their observations with their private key.

Notaries are queried by clients: a client asks for a notary's observations for a domain and receives a full copy of these observations. We model this as a *Service Notarising* in our notation (lines 55–75 in Listing 14). We denote the requests that a client makes by having the client send the service the list of notaries whose observations it wishes to obtain. The service returns them as a *Map* with one entry per queried notary. This gives us a simple way to express that each client may communicate with different notaries, over different channels.

Perspectives introduces an additional concept to make the scheme robust against misbehaving or faulty notaries. Every notary acts as a so-called *shadow* for a number of other notaries. A notary is required to send new observations for a domain to its shadows. This is shown in line 169. A shadows stores the observations and does a sanity check on them. In our notation, we represent the shadowing as a service—this reflects that a number of shadows exists per notary, although they all execute the same steps[7]. Listing 15 shows the shadowing. A shadow verifies that a period has sane timestamps (end not before beginning) and that the observations do not overlap (this would be a faulty execution of the scanning protocol). We model these checks as procedures in lines 28–33 and lines 35–40, respectively. If the shadow has older observations from this notary, it verifies that the old observation periods are entirely contained within the new (i.e., either are the same or periods have been extended due to new observations of the same key). This is shown in lines 105–115. If the new observations are found to be consistent and sane, they replace the old ones. If they contradict them, they are stored as proof of misbehaviour. A shadow signs the data it stores with its own key. Our shadows do not accept observations from notaries where they cannot verify the notary's signatures[8].

The final piece in this description of Perspectives consists of the changes it makes to the TLS protocol flow. We show this in Listings 17 and 18. A run of TLS is initiated from within the *Sessions* block by passing the client participant and the notaries to use as arguments to the *Protocol TLS_DHE_Perspectives* (line 257). The protocol begins with a client requesting the list of shadows[9] (lines 187–191). Normal TLS follows, up to the point when the client receives the server's last message. In our notation, we use a simple caching concept for clients—keys that have previously been corroborated by notaries and shadows are added to a cache (line 243). The client checks if the key is in the cache (line 195). If it is, it will simply finish the TLS handshake, without any further notary checks. Otherwise, it queries the notaries for their observations (line 199). Perspectives allows to define the evaluation of both notary observations and shadow corroborations in a client policy. In general, however, a quorum of notaries is required to reply to queries. We use a somewhat relaxed policy in our example: two thirds of notaries must reply, and all of them must report the same public key that

[7]The publication on Perspectives does not reveal how exactly the data between notaries and shadows is protected in transit. The notaries' signatures protect the observations, but at least the name of the observed domain should be additionally protected. For simplicity, we chose to model this as a secure channel in our notation.

[8]This is not mentioned in the original paper, but we introduce it to avoid arbitrary notaries being able to flood shadows with nonsensical entries.

[9]The original paper does not describe how clients learn about shadows. We thus use this simple method.

the client observed as the latest known key (lines 204–214). If the quorum is reached, the client queries all shadows and requests their logged observations for the notaries in question (line 215). The policy requires that all of their observations are consistent with what the notaries sent the client. If not, the policy causes the client to reject the key and stop the protocol. This is shown in lines 221–240. If all observations are consistent, the client will accept the key.

As can be seen from the above descriptions, Perspectives does not make any changes to the behaviour of servers. Servers do not even need to be aware that clients use Perspectives. Note furthermore that clients do not check the validity of the server certificate—the only important factor is the public key the server uses.

8.4.3. Simplifications and choices in representation

Compared to the original paper, we made several minor simplifications. At some points, it was also necessary to interpret the intention in the original publication.

First, notaries in Perspectives observe 'services', i.e., combinations of host, port and protocol. This allows Perspectives to be used with any TLS-based protocol, not just HTTPS[10]. In our notation, the notaries observe domains and we assume HTTPS is the protocol used.

Second, Perspectives groups notaries in so-called notary groups, in which one notary is authoritative and can add and remove other notaries from its group. We omit this in our notation, which is consistent with the discussion in the original paper.

Third, the original publication allows clients to select a subset of shadows and define a second quorum on them. We use all shadows by default, which keeps the description more succinct.

Fourth, the original publication hints at more elaborate caching schemes and client policies. In our notation, we opted for the simplest possible caching: a key is added to the cache whenever notaries and shadows confirm it. We omit emptying and refilling the cache for now.

Finally, the original publication does not clearly specify the protocol flow a client executes when querying the shadows. In particular, it does not specify whether clients query shadows in parallel to notaries or afterwards. In our notation, we chose the second option as it results in a clearer protocol flow.

It should be noted that Perspectives is not yet fully implemented. At the time of writing, Perspectives notaries are run primarily by the designers of Perspectives themselves, based at Carnegie Mellon University, USA. Only five other notaries are listed on the project's homepage [247]. The designers stress that they could not carry out full background checks on notaries to assert their good intentions. Shadows are not yet implemented [255].

8.4.4. Assessment of Perspectives

We assess Perspectives with regard to its contributions to reinforcing the X.509 PKI, its robustness against different attackers and potential deployment issues.

Reinforcements to X.509

Perspectives makes the following contributions to reinforcing the X.509 PKI.

[10]In fact, the authors of Perspectives emphasise that it can also be used for SSH, although this does not seem to be implemented.

Out-of-band mechanism The purpose of Perspectives, as presented in the original publication, is to improve security for clients and protect them against man-in-the-middle attacks. The scheme supports the established certification processes, but does not require them: certificates do not have to be issued by CAs in Perspectives. As such, Perspectives is both an out-of-band mechanism to reinforce the current PKI as well as a replacement for it.

Incident detection Although Perspectives is well suited for incident detection, it does not actually support it. Mismatches between notary observations (and shadow corroborations) and server certificates could be indications of an ongoing man-in-the-middle attack, but Perspectives does not define any mechanisms to report these. The original publication mentions that shadows may store conflicting notary observations as a way to prove notary misbehaviour to some entity. This would be very useful—unfortunately, Perspectives does not describe a method to achieve this. Although Perspectives' purpose is not incident detection, this seems a viable path for improvement.

Monitoring and transparency Perspectives does not concern itself with observations of the X.509 PKI, neither with certification nor with deployment. While notaries store observations of the PKI, their data set is necessarily incomplete as notaries scan only a certain number of domains. Perspectives treats the corroborations for the authenticity of public keys as the single criterion for correctness. Misconfigurations, such as mismatches between a certificate subject and a hostname, are not considered.

Perspectives shows a good degree of transparency: notaries observe other notaries as their shadows. The transparency is limited, however, by the number of notaries that exist—at the time of writing, the number is rather small. Worse, the concept of shadows is not yet implemented. This level of transparency is thus inferior to the transparency offered by concepts that are accessible to, e. g., active scans. It is possible to extend Perspectives such that greater transparency is achieved. This would require an additional API that allows to download all of a notary's observations at once.

Robustness against attackers

Perspective's security rests on three pillars: its ability to communicate with notaries at different vantage points, the quorum needed for a client to accept a key, and clients acting on cached historical key information.

The authors of Perspectives discuss several attacks in their publication [77] and distinguish between localised attacks (similar to our Model A), attacks against paths to the server, and attacks against notaries. We structure our discussion in a similar way, but describe possible attacks in the context of our threat models.

In our analysis, we make the assumption that a client expects that at least a quorum of notaries replies to its query (with observations), plus all shadows for these notaries. Everything else is considered suspicious and the client aborts the connection attempt.

Model A We consider the weaker attacker of Model A. This attacker cannot compromise any entities, and is limited to attacking network paths in either the client or the server's vicinity. We consider proximity to the client first. In this case, the attacker cannot modify messages to and from notaries without detection and is limited to deleting them. Deletion affects the quorum that a client requires. It is thus a way to force a client to reject a key that might be valid. However, the attacker cannot force the client to accept a forged server key.

If the attacker is in the vicinity of the server, e. g., in control of a border router of a local network, he can stage a man-in-the-middle attack that affects all notaries connecting to the server. The security that Perspectives offers in this case depends entirely on the client's policy concerning historical data about keys. In a conservative setting, a client may assume a cached key to have been compromised in the meantime, and uncached keys to be invalid.

Note that stricter client policies may require observations by notaries to report the key in question for a certain period of time. This would increase confidence in the key: an attacker would have to maintain his attack for a prolonged period of time, which increases his risk of exposure.

Model B The attacker in Model B can control more network paths and also compromise entities. The entities of interest in this case are the notaries (recall that Perspectives aims at the authenticity of server keys, not at the correctness of certificate issuance). If clients inside the attacker's region use notaries outside the region, the attacker's control over network paths leads to results that are similar as in the discussion of Model A. There is one major difference: if the victim server is inside the attacker's region, the attacker can stage a man-in-the-middle attack against all notaries. Perspective's defence would be reduced to the historical observations of keys.

The attacker's chances are much better if we allow him to compromise notaries. In this case, the victim server's location does not matter, nor do network paths to notaries or shadow servers matter. The attacker can change the observations stored by the notaries themselves. Note that this also removes the need to continue the attack for a prolonged period. If the victim server is outside the attacker's region, Perspective's cross-validation protocol still affords some protection. If the attacker cannot compromise the shadow servers of the notary either, their reply to the client would yield conflicting observations. As clients can theoretically choose arbitrarily from the set of notaries, the attacker cannot predict which notaries and shadows to attack. If Perspectives succeeded in offering hundreds of notaries, the number of entities that the attacker would have to compromise in order to have a very good chance quickly reaches a large number—this should be detected sooner or later.

Model C As DNS and DNSSEC are not relevant for Perspectives, the greater strength of our attacker in Model C compared to Model B consists only of his more arbitrary control over paths on the Internet. Against this background, the most favourable case for this attacker is control over all paths from notaries to the server. The impact would be the same as in Model B, except that the attacker can also affect servers outside his region. Concerning the compromise of entities, the attacker in Model C has the same capacity as the one in Model B.

Deployment

Perspectives has two requirements for deployment: an infrastructure of notaries (doubling as shadows) and changes to the TLS implementations on client side. Server software does not need to be changed. We view this as a strong advantage as there is a very large number of Web servers, and we know from Chapter 4 and Chapter 6 that the pace of change on server-side seems to be slow. Concerning client-side implementations, Perspectives can be rolled out gradually.

Very likely, however, there is an entirely different obstacle that hinders fast deployment. In theory, clients should be able to choose from a larger set of notaries (or notary groups). At the time of writing, just a handful of notaries exist, and most are operated by private persons. Ideally, notaries should have a very high availability to avoid errors or warnings on the side of the clients.

Even if such notaries were available, however, it is an open question how users would select them in their client applications. Google engineer Adam Langley discussed this problem in [207] in the context of Convergence, which at that time was the better known notary system. To avoid frequent warnings, Langley argues that vendors like Google themselves would have to run notaries (to achieve high availability) and enable their notaries by default. Langley expresses concern that the vast majority of users would never change the default settings, making Google the single party everyone trusts. This would defeat the concept of distributing trust to many notaries.

Perspectives, like all notary concepts, has a further issue that may hinder deployment: notaries learn from queries which domains clients visit. They can thus track their activities on the Internet. Even if a notary does not engage in such an activity, such data can be a promising target for attacks on notaries. Caching key information on client-side allows to limit the impact on privacy to some degree. For users who still find the privacy issue unacceptable, the designers suggest a DNS-based proxy scheme. At the time of writing, this scheme is not implemented [272]. As Convergence demonstrates, the problem can be somewhat mediated by a simple form of onion routing.

In summary, the concept of notaries shows remarkable strength against attacks. However, its usefulness is severely limited due to obstacles to deployment.

8.5. Public log-based proposals: Certificate Transparency

Certificate Transparency (CT) is the last scheme we discuss. It is based on so-called *public logs*. Public logs are services that store certain information in a way that makes it accessible and verifiable for other parties. A fundamental property of public logs is that they are append-only data structures.

8.5.1. Choice of Certificate Transparency as subject to study

We are aware of only two PKI schemes that make use of public logs. CT is a proposal initiated by Google. The primary objective of the scheme is to make the certification process auditable by a larger public: every certificate that a CA issues is logged in a data structure that is extremely hard to tamper with (without such activity being detectable). CT does nothing to improve the security of the certification process itself—if a CA is compromised, the attacker can issue certificates. However, with CT, the CA cannot deny the compromise and it is known which certificates the attacker has issued as these must have been logged. At the time of writing, CT is in the last stages of the design phase, with a concept that is stable enough to discuss here. CT is described in several documents on the project homepage, and source code exists [154, 155]. An experimental RFC is available, RFC 6962 [109]. Still, some parts of CT are not fully specified yet, and as minor design decisions can still be made we advise the reader that some sections in the RFC are still subject to change.

Sovereign Keys (SK) is a project initiated by the Electronic Frontier Foundation (EFF) [187, 186, 185]. It was first proposed in 2011 [157]. The approach it takes is different from CT, despite the use of similar technology. The idea is that every domain owner registers a binding of a public key to a service running on the domain (e.g., HTTPS) in a public log. This ‘sovereign key’ is later used to cross-sign any operational keys that are deployed, i.e., TLS keys in our example. SK works like a directory of certification information that stores keys and evidence of domain ownership. SK bootstraps from other technologies to provide such evidence. The authors suggest, e.g., DNSSEC or X.509. However, SK is by far not as mature as CT. Despite an initial release of code, the project does not seem to have made significant progress since the early documents, although the EFF still plans to continue it[11].

Although SK is a very interesting concept, too, we chose CT as the subject to investigate here. It is the more mature scheme, with protocol flows and data structures developed to a degree that make it accessible to analysis. Furthermore, Google recently took first steps towards deployment. In a post to the CA/Browser forum's mailing list, Google announced that it would make CT obligatory for EV certificates in its Chrome browser [258].

8.5.2. Operation and representation in our notation

The objective of CT is to make it impossible for a CA to issue a certificate for a domain illegitimately without being detected. In particular, the owner of the domain should be able to notice it and take action against it (e.g., by legal means, notification of browser vendors and CAs, etc.). CT stores the information which CA has issued which certificate in publicly accessible and append-only logs. Logs are not synchronised—they do not exchange entries.

Logs issue a proof of inclusion when they accept a new entry. Domains use this to prove to clients that their certificate has been logged. As logs allow to retrieve logged certificates, domain owners and CAs can set up their own systems to observe which certificates a log contains and which ones are added to it. Domain owners and CAs can determine whether other certificates for a given domain have been issued and thus detect rogue certificates. An infrastructure of so-called *monitors* and *auditors* ensures the logs themselves cannot alter entries maliciously without that being detected. The public log mechanism is thus a defence against compromises as in the DigiNotar case (see Section 3.3). With CT, the legitimate domain owners, and possibly monitoring and auditing parties, would have been able to notice the attack at an early stage and countermeasures could have been taken earlier. Furthermore, DigiNotar would not have been able to hide the compromise for long.

CT is relatively complex and the RFC specifies alternative paths through the scheme, too. In the following, we restrict ourselves to the core mechanisms. We describe these in our notation in listings 22–27, where we made sensible choices where alternatives were possible. We will briefly describe what the alternative would have been for each decision we made.

CT introduces three new types of participants to X.509: *public logs*, *monitors* and *auditors* (lines 3–8). Public logs form the backbone of CT: they keep track of issued certificates (including certificate chains) and store additional information: issuing CA, time of issuance, and domain for which the certificate was issued. Logs maintain a list of acceptable root certificates (line 27); certificates must chain to a CA in the list. This list of root certificates is a defence against malicious parties trying to flood the logs with nonsensical entries. It is not meant to be a trust anchor: logs are not TTPs. Logs have a public/private key pair (line 28), which they use in their interactions with other parties.

Data structures

Logs use an efficient data structure, namely a *Merkle hash tree*. This data structure was originally defined in [219]. A binary Merkle hash tree is a binary tree where input values are represented as leaves. These are hashed to yield nodes. Moving up in the tree towards the root, two hashes are concatenated and hashed to yield a parent node. New inputs are added (from the right) by hashing them and recomputing the tree.

[11]This was confirmed to the author in private email.

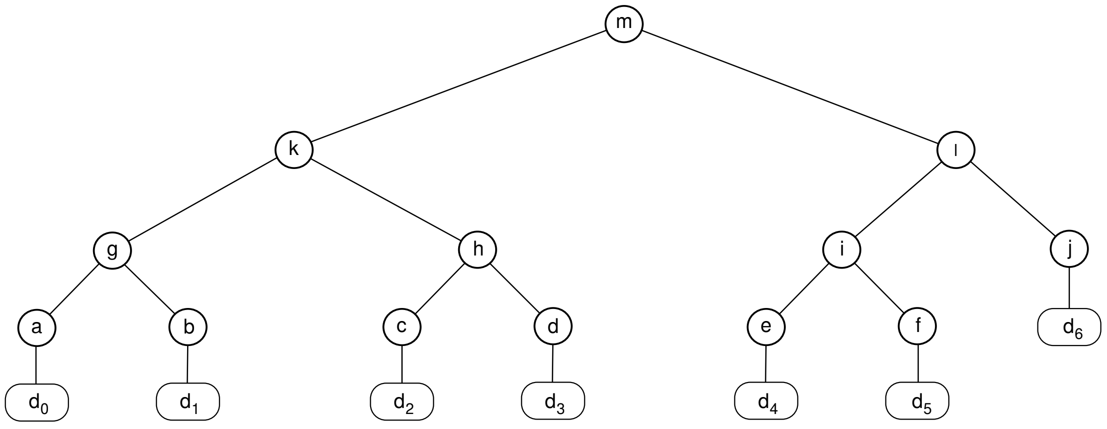


Figure 8.1. – Merkle hash tree as used in CT. Entries d_i are new certificates (including the chains). Figure follows RFC 6962.

Listing 19 Algorithm to compute a Merkle Tree Hash (MTH) as defined in RFC 6962.

```
Algorithm Compute MTH
    Requires: Ordered list of byte strings $D[n] = \{d_0, d_1, \ldots, d_{n-1}\}$
    Output: Merkle Tree Hash of list
    Procedure mth(D[n]):
        if $D = \varnothing$:
            return hash($\epsilon$)                                 ▷ Hash of empty string
        if $|D| = 1$:
            return hash('0x00'|$d_0$)                               ▷ prefix with null byte
        else:
            $k \leftarrow 2^i : (\nexists j : 2^i < 2^j < n)$       ▷ Largest power of 2 smaller than n
            $D[0{:}k] \leftarrow \{d_0, d_1, \ldots, d_{k-1}\}$
            $D[k{:}n] \leftarrow \{d_k, d_{k+1}, \ldots, d_{n-1}\}$
            return hash('0x01'|mth(D[0:k])|mth(D[k:n]))            ▷ prefix with 1 as byte
```

Thus, the final root hash is essentially a hash over the entire tree and changes with every new entry. Figure 8.1 shows a binary Merkle hash tree. Listing 19 gives the algorithm to compute the tree. Its final output is the root hash. In our case, the inputs d_i are certificates or certificate chains—as we have done before, we do not distinguish between them in our notation.

In our notation, we define a Merkle hash tree as a well-known record *Merkle_Tree* that is associated with several well-known processes. A *Merkle_Tree* holds leaves and nodes of the tree in an appropriate data structure (we abstract over the concrete implementation). The root hash of the tree is called Merkle Tree Hash (MTH)[12]. It is stored in a field *mth*. The value of this field is a String as that is the output of applying the well-known process *hash()*. A field *size* in the *Merkle_Tree* stores the current number of entries. We allow two ways to instantiate a *Merkle_Tree* with the operator *new*: either as a fresh Merkle tree without any entries, or by passing a set of entries. In the latter case, we assume the Merkle tree to be built from the entries[13].

The first well-known process associated with a *Merkle_Tree* is *add()*, which we denote as a field. Every time a new certificate (chain) is added to the Merkle tree, this process is executed. The result is an updated tree, where the fields *mth* and *size* have been

[12]Not to be confused with the Merkle hash tree, which is the entire tree.

[13]The Merkle tree must also be implemented in such a way that it allows arbitrary access to an entry by its index.

Listing 20 Algorithm to compute an audit path, as defined in RFC 6962.

```
Algorithm Compute audit path
    Requires: Ordered list of byte strings: $D[n] = \{d_0, d_1, \ldots, d_{n-1}\}$,
              input to compute audit path for: $d_m$, $0 < m < n$
    Output: Concatenated minimal list of nodes (hash values) to compute MTH
    Procedure audit_path(m, D[n]):                    ▷ $d_m$ is $m+1$th input
        if $m = 0$ and $D[n] = \{d\}$:                ▷ One-element D[n]
            return $\epsilon$                         ▷ Return empty string
        else:
            $k \leftarrow 2^i : (\nexists j : 2^i < 2^j < n)$   ▷ Largest power of 2 smaller than n
            $D[0{:}k] \leftarrow \{d_0, d_1, \ldots, d_{k-1}\}$
            $D[k{:}n] \leftarrow \{d_k, d_{k+1}, \ldots, d_{n-1}\}$
            if $m < k$:
                return audit_path(m, D[0:k]) | mth(D[k:n])
            if $m \geq k$:
                return audit_path(m – k, D[k:n]) | mth(D[0:k])
```

updated, too. The second well-known process is *retrieve()*. Called without a parameter, it returns all entries. Called with two parameters *from* and *to*, it returns all entries from the *from*-th to the *to*-th entry.

CT uses two more data structures: Signed Certificate Timestamps (SCTs) and Signed Tree Hashes (STHs). SCTs are used to prove to a client that a value has been accepted for inclusion by a log. They are shown in lines 16–20 in Listing 22 and contain a timestamp and the certificate (chain) in question, signed with the log's private key. Note that the log includes the hash value of its public key in the SCT: this value serves as a log's identifier in the scheme. It is not signed as these public keys are supposed to be distributed out-of-band: they are preconfigured in client software (line 37), much in the same way CA root certificates are distributed in root stores today. STHs, shown in lines 10–14, are used as part of consistency checks. They contain the log's current size (number of entries), a timestamp, and the current MTH, signed with the log's private key.

The Merkle hash tree and an initial STH are initialised as part of the global *Init* in our notation (lines 26–31). STHs are tracked and used by both monitors and auditors; they thus also initialise fields to hold them (line 33 and line 39).

Consistency and consistency checks

Merkle trees have one important advantage: a small subset of nodes and leaves is enough to verify the correctness of the MTH. This, in turn, allows to verify the append-only property, and also that a certain input was included in the tree at some point. We refer to the latter as consistency property. Violations of either append-only or consistency property indicate tampering with the log or incorrect behaviour. Monitors and auditors are responsible for carrying out checks that no violation has occurred. In CT, logs that are found to exhibit a violation are considered untrustworthy.

CT defines the terms *audit path* and *consistency proof*. An audit path allows to determine whether a certificate has been incorporated into the log. An audit path for a leaf in the Merkle tree is a minimal set of other nodes that are needed to compute the current MTH. It is exactly the list of missing nodes (hash values) from an entry to the root (i.e., the MTH). The RFC describes a recursive algorithm to determine an audit path [109]; we show the algorithm in Listing 20. Note that our output is a concatenated

Listing 21 Algorithm to compute a consistency proof, as defined in RFC 6962.

```
Algorithm Compute consistency proof
    Requires: Ordered list of byte strings: $D[n] = \{d_0, d_1, \ldots, d_{n-1}\}$,
              input to compute proof for: $mth(D[0:m])$, $0 < m < n$
    Output: Minimal and unique list of nodes to verify $mth(D[0:m])$
    Procedure cons_proof(m, D[n]):                    ▷ $d_m$ is $m + 1$th input
        return sub_proof(m, D[n], true)        ▷ Call sub_proof with flag set to true

    Procedure sub_proof((m, D[n], flag):
        if $m = n$:
            if flag = true:
                return $\epsilon$
            else:
                return $mth(D[m])$
        ▷ If $m < n$
        else:
            $k \leftarrow 2^i : (\nexists j : 2^i < 2^j < n)$        ▷ Largest power of 2 smaller than n
            $D[0:k] \leftarrow \{d_0, d_1, \ldots, d_{k-1}\}$
            $D[k:n] \leftarrow \{d_k, d_{k+1}, \ldots, d_{n-1}\}$
            if $m \leq k$:
                return sub_proof(m, D[0:k], flag) | mth(D[k:n])
            else:
                return sub_proof(m – k, D[k:n], false) | mth(D[0:k])
```

list of hash values—as hash functions have a fixed output length, it is no problem to derive the separate hash values from the output. Based on this algorithm, we assume a well-known process to exist in our description of CT: *audit_path()* represents this algorithm. The first parameter is a certificate, which is supposed to be an entry in the log. The second parameter is the size of the log as given in an STH. Furthermore, we assume a well-known process *verify_audit_path()* that computes a root hash from the entry in question (given as an index, see Figure 8.1) and the audit path (given as a set of Strings as nodes in the audit path are hash values, which we represent as Strings). It checks whether the result matches the given MTH.

The append-only property can be proved with so-called consistency proofs. The necessary algorithm is also defined in the RFC; we reproduce it in Listing 21. Given the root hash of the current and the root hash of the Merkle tree at a previous time, the proof consists of the set of nodes in the Merkle tree that are required to verify that both trees are equal up to the point when the previous tree was created. We omit the exact description here and refer the reader to RFC 6962 [109]. For our notation, we assume the existence of two well-known processes: *cons_proof()* computes the proof, and *verify_cons_proof()* verifies it.

Certification and sending the SCT to clients

Logs cooperate with CAs in the certification process. The RFC describes several options. In our listing, we show the straight-forward one. A domain initiates the certification process by sending a request to a CA (line 62). The CA carries out its normal duties in validating the request and then generates the certificate. It sends the certificate to the log for inclusion, together with its root certificate. This is shown in lines 47–52. The log verifies the CA is in its list of acceptable CAs and incorporates the certificate. The log creates an SCT and also an STH. The SCT is sent back to the CA. Lines 75–82 show this. Upon receiving it, the CA sends the certificate to the requesting domain, together with the SCT (lines 53–55).

There are two alternatives here. The certification as we presented it above has a drawback: a server that participates in CT needs to send the SCT to clients as part of the TLS handshake (this is shown in lines 150–151). It must thus support TLS extensions. The alternative is to have the CA embed the SCT in the server certificate. In this case, a CA must issue a so-called precertificate and send this to the log instead of the real server certificate. This certificate carries special X.509 extensions that mark it as unusable, but otherwise holds the complete information. The log incorporates the information from the precertificate. This alternative has the advantage that servers do not need to change their configurations at all. The second alternative, which also avoids having to reconfigure servers, is to deliver SCTs as a part of an OCSP reply. In other words, every time a client carries out a revocation check (also see Section 2.7), an SCT is included in the reply.

Monitoring and auditing

Monitors and auditors are responsible for detecting log misbehaviour. Both activities are underspecified in the RFC at the time of writing[14].

Monitors A monitor verifies the consistency of one or more logs. Lines 83–139 show the monitoring process.

A monitor may keep copies of the Merkle trees of logs it observes. Monitors require comparatively many resources—they are meant to act on behalf of less powerful entities, such as domain owners or clients (e. g., browsers). Monitors could be operated by, e. g., ISPs or CAs. For each log that they observe, monitors carry out the same steps at periodic intervals.

The process begins with fetching the current STH from a log (line 92 ff.). Initially, the monitor will have stored an empty last-known STH, so the new STH is not meaningful to it yet. Thus, once it has verified the signature in the STH, it fetches all entries for this STH (line 99). The monitor reconstructs the Merkle tree from the entries and compares the MTH with the MTH stored in the STH (lines 106–111). If they do not match, that would be an indication of misbehaviour.

If the monitor is already in possession of earlier STHs, the process is slightly different. It first queries all entries since the last STH (line 104). This is primarily an availability check: logs must make entries available on request. However, the monitor may carry out any checks it wishes—it could, for example, verify now that no new certificates for a certain domain have been added. After this check, the monitor proceeds to request a consistency proof from the log (line 117). The log responds with such a proof (shown in lines 137–139), which allows the monitor to verify that the logs' entries are identical up to the point of the previous STH (line 123). These checks can be repeated arbitrarily from this point on. Note that the log does not need to recompute the Merkle tree for this verification of consistency—although it is free to keep an up-to-date copy. A monitor is expected to report any kind of misbehaviour on the part of the log. At the time of writing, CT does not define concrete actions that monitors must take, however.

Auditors Auditors in CT are entities that, in general, have less computational resources than monitors. They hold just enough information about the full Merkle tree to verify that some property they are interested in is consistent (and remains consistent) with

[14]The RFC version that we use in our analysis dates from June 2013.

the log. The process of auditing is currently only described in a short paragraph in the RFC, with two basic forms mentioned. One are the consistency proofs from above, with auditors just verifying that there is a correct consistency proof between two STHs, but not storing or validating any other information. The second form uses audit paths: in this case, an auditor is, e.g., a client who has carried out an TLS handshake earlier and is now in possession of an SCT. It can request STHs and audit paths from the log to verify that the certificate remains stored in the log. We present this second form in our listing (lines 174–220). We assume an earlier run of TLS, where the SCT has been acquired. If no previous STH is known, the auditor queries the log for one and just validates the signature (lines 187–199). If a previous STH exists and the auditor finds that it has received a newer one, it acts differently. We show this in lines 200–207. The auditor additionally queries an audit path from the log. It verifies that the path is correct—this proves that the certificate (for which the SCT has originally been issued) is still included in the log. The RFC also proposes that auditors obtain consistency proofs to verify the append-only property from STH to STH. We do not show this in our listing as the protocol is the same as for monitoring, except that auditors would usually not store or further validate the entries but delete them once they have verified a new STH.

Cross-validation The RFC recommends that clients, auditors and monitors should exchange information they have received from logs. The suggested method is to use a gossiping protocol; however, the current version of the RFC does not specify this any further.

Simplifications

In our listings, we made two simplifications. First, CT always stores the intermediate certificates in a chain. The rationale is that a CA might issue a certificate to a domain using an intermediate certificate and later choose to replace the intermediate certificate with another one, where the public/private keys are the same but the expiry date is different. CT views this as a new certification. We do not model this to avoid unnecessary complexity.

Second, logs do not update their Merkle trees every time a new certificate is added. Instead, they indicate an allowable delay. If they fail to carry out inclusion within this time span, this counts as log misbehaviour and the log is considered untrustworthy. This mechanism does not contribute to security (but contributes to performance). We did not model it either.

8.5.3. Assessment of Certificate Transparency

We give an assessment of CT's contributions to X.509 and its robustness in our threat models. We also analyse potential deployment issues.

Reinforcements to X.509

We analyse which contributions CT makes to reinforce the current X.509 PKI.

Out-of-band mechanism CT is not a preventive mechanism but a reactive one. The scheme does not aim to protect clients directly. Clients get more reassurance than in the normal TLS protocol, however. The information in the SCT tells them that the server's certificate has been logged. CA compromises and malicious behaviour become detectable with CT, and parties like browser vendors are expected to react quickly to news of a CA compromise. An SCT is a reassurance, albeit not a perfect one, that the certificate issuance was correct and the certificate in question is the intended one for the respective domain.

Incident detection Incident detection is the primary purpose of CT: it makes it possible to track and verify changes in the certification of domains. Every time a certificate is issued, this fact is stored in a log, in a way that is very hard to forge. This makes it possible for domain owners to determine whether the current entry in a log is the correct one, and CAs can verify whether other certificates for a certain domain exist.

Monitoring and transparency CT's explicit goal is to monitor the issuance of X.509 certificates and make these processes verifiable. Logs store issued certificates together with their intermediate certificate. However, as logs do not synchronise and exchange entries, they can only store partial information about the X.509 infrastructure. Furthermore, CT does not monitor the actual deployment, i.e., whether certificates are used on the intended host or other configuration problems exist. The concept is not easily extensible to achieve this, either. However, CT reaches into a region that is not accessible to active probing: certification events where certificates are never deployed on servers. This allows insights into CA activities that would otherwise remain unknown (e.g., mistakenly issued certificates with immediate revocation).

CT makes it extremely easy to assess its correct functioning from the outside: the notion of monitors and auditors allows to retrieve the entire data that is stored in a log and observe its correct behaviour.

Robustness against attackers

In the following, we assess the robustness of CT in the context of our threat models. Note that the scheme defines secure channels between logs and other participants (clients, monitors, auditors) as the logs' public keys are well-known and distributed out-of-band. Logs can always send authenticated information to other entities. At the same time, they only need to receive authenticated information at one point in the scheme, namely when they receive a certificate from a CA that is to be logged. The certificate itself carries the CA's signature, and thus this communication is secure.

Model A This attacker is located either close to the victim server or close to the victim client, and he is unable to compromise entities in the scheme. We show now that this attacker is defeated by CT.

We discuss the position close to the server first. The attacker cannot tamper with the certification process as the channel between domain and CA is secure (recall that this is a general assumption necessary to allow certification in the first place). He cannot tamper with SCTs sent from the server to the client, either: the transmission of SCTs is secure in all three variants. If it is sent in the TLS extension or embedded in the certificate, it is protected either by the TLS handshake itself or by the CA's signature on the certificate. If the SCT is delivered via OCSP, this attacker is additionally also in the wrong position to intercept it as the CA is queried for the SCT, not the domain.

Let us assume now the attacker is close to the client. The only part of CT that he can attack are again the SCTs. Once again, this transmission is secure, for the same reasons as above.

In summary, CT defeats the weak attacker as long as the domain in question is certified by a CA that appears in the log's list of acceptable CAs. Note, however, that other cases are not covered by CT (e.g., certificates issued from non-recognised CAs).

Models B and C As mentioned, the channels between logs and other participants (including clients) are secure, as is the channel between domain and CA. As we showed above, the attacker cannot interfere with the transmission of an SCT, either. Recall that he difference between Models B and C is the attacker's control over DNS and DNSSEC as well as his control over selected network paths. As a result, the options for the attackers in Model B and C are the same, and we discuss them together here.

We first assume the attacker was able to compromise a CA and obtain a forged certificate, but was not able to trigger the mechanism that causes the certification to be logged. In this case, he would not be in possession of an SCT, and his attempt to use the certificate would result in the client rejecting the connection. In order to be successful against CT, an attacker must thus compromise a CA to the degree he can control the entire process of certification, including the interaction with logs.

Let us now assume the attacker has compromised a CA and issued himself a rogue certificate, and he was able to trigger the interaction with the log. He is now in possession of an SCT. Initially, this attack works. However, it is detectable, although not by a client. Recall the monitoring process (see Listing 24): here, a monitor retrieves entries from the log and can monitor them for suspicious changes. The new certificate will be suspicious if, for example, the monitor is operated by the rightful CA that issued the real certificate. This example shows the importance of other parties running monitors. We can also see here that CT shows properties that are useful for forensics: it cannot prevent the attack, and some clients may be harmed, but it helps raise the alarms earlier and identify the responsible parties. The more decisive question is how long the delay between rogue issuance and detection is. This depends on how many monitors exist and whether they cover all logs and carry out their verifications in reasonably short intervals. If this is the case, CT shows great resistance to attacks against CAs. Let us now assume that the attacker attempts to compromise at least a part of CT's infrastructure in order to evade detection and sustain his attack. He must thus attempt to compromise a log and send false information to requests. Logs are distributed, run by different parties, and (at least with the current RFC) there is no mirroring between them. This means the attacker would not only have to compromise a CA but also the log with which this CA usually cooperates. If a CA pushes a certification to a larger number of logs, the attacker would need to compromise all of them. Even then, however, his attack is still detectable by the monitors and auditors who verify that the log is consistent and append-only. The attacker would have to compromise these entities, too—all without being detected even once. If we assume that CT becomes a popular concept and many monitors exist, this is a true obstacle, even for the globally active attacker of Model C.

The conclusion to draw is that even the most sophisticated attackers will find it difficult to make non-detectable changes to the logging infrastructure. CT's power lies in its capacity to allow detection of incidents shortly *after* they have occurred. However, much will depend on how many logs exist, how many monitors observe them, and at which intervals. Furthermore, CT offers only an indirect defence for clients. It it thus a useful complement for other methods to strengthen X.509, like TACK or notaries.

Deployment

CT introduces new participants to the X.509 PKI. Consequently, it has a number of requirements for deployment. The first question is which entities should operate logs. Logs need to guarantee a high availability, which translates into relatively high operating costs. The incentives for operators must be correspondingly high. The relevant parties that (arguably) have the strongest interest to improve X.509 are browser vendors and CAs. These parties generally also have the resources and the expertise to construct systems with high availability. The question will be whether CT provides enough benefits for them: in an email to the CA/Browser Forum's public mailing list, Symantec revealed that this was not the case for them [134]. At the time of writing, Google already operates a log [154]. It remains to be seen if and when other entities will follow. However, as Google has announced that it will enforce CT at least for EV certificates [258], it is conceivable that other parties will begin to run logs, too, if only out of peer pressure. It is an interesting question how many logs should exist, and how many will. The project's homepage remains rather vague on this topic—it gives a number of *'more than 10 but much less than 1000'* [154]. At the time of writing, it is too early to make any reasonable predictions.

Running monitors and auditors is much less costly than running logs. These systems do not need to have high availability as they do not communicate with clients. It seems quite conceivable that CAs and ISPs run monitors and several auditor add-ons may be developed for browsers. This would establish a very strong system of cross-validation. As we have seen in our discussion of attackers in Models B and C, it is crucial that a large number of such entities exist.

Assuming the hurdles towards deploying new infrastructure for logs, monitors and auditors are overcome, there is still a crucial issue to resolve: the need to make changes to existing systems and configurations. Clients are not the pressing problem here: modern browsers often have an auto-update functionality. The problem lies with CAs or alternatively with servers. In the same email to the CA/Browser Forum mailing list, Symantec argued that all of CT's delivery mechanisms for SCTs are flawed. It is true, at least, that each option requires at least one group of entities to make significant changes. If SCTs are to be delivered by a server in an TLS extension, this means server software must be upgraded to support newer TLS versions and the extension required by CT. The number of Web servers is immensely high, and we know from our results from active scans that changes are rarely made on server-side. Delivering SCTs embedded inside the server certificate seems to be the better option here as servers do not need to make any changes at all. However, the onus is now on CAs: they would need to change their issuance processes and create precertificates. According to Symantec, this is an extensive effort. As the security of this process is essentially a CA's product, a certain reluctance on the side of CAs is to be expected. A similar argument holds for delivering SCTs as part of the OCSP protocol: CAs would still bear the onus. With no middle ground left that would serve as a viable solution, it will remain to be seen whether the benefits that CT offers are incentive enough (for CAs) to make the necessary changes to adopt CT. The CA GlobalSign, at least, announced that it would adopt the scheme [201].

To summarise, success or failure is hard to predict for CT. There is a delicate trade-off between very strong security benefits and associated costs, with opt-in required from CAs. With Google advocating the concept, however, there seems to be a reasonable chance that CT will be deployed.

Listing 22 Scheme of CT, Part 1.

```
Scheme Certificate_Transparency:
    Participants:
        CAs : {Ca_0, Ca_1, ...}                                ▷ Set of all CAs
        Domains : {D_0, D_1, ...}                              ▷ Set of all domains
        Clients : {Cl_0, Cl_1, ...}                            ▷ Stand-alone, browsers, etc.
        Logs : {L_0, L_1, ..., L_l}, 30 ≤ l ≤ 100              ▷ Number of logs is an example here
        Monitors : {M_0, M_1, ...}                             ▷ Work on complete log information
        Auditors : {A_0, A_1, ...}                             ▷ Work on partial log information

    Record STH:                                                ▷ Signed Tree Hash
        size : Integer                                         ▷ Current size of Merkle tree
        t : Timestamp
        mth : String                                           ▷ Merkle Tree Hash, see Listing 19
        sig : sig(Integer|Timestamp|String)

    Record SCT:                                                ▷ Signed Certificate Timestamp
        $k_{log}$ : h(Pub_Key)                                 ▷ Hash of log's public key, acts as ID
        t : Timestamp
        cert : Cert                                            ▷ Certificate of server
        sig : sig(Timestamp|Cert)

    Init:
        for Ca ∈ CAs:
            (Ca.k, $Ca.k^{-1}$) ← new Key_Pair()
            Ca.root ← new $Cert_{Ca.k^{-1}}$(Ca.name|Ca.k)
        for L ∈ Logs:
            root_list ← {Ca.root : Ca ∈ CAs}                   ▷ Logs know CAs' root certificates
            (L.k, $L.k^{-1}$) ← new Key_Pair()
            L.mtree ← new Merkle_Tree()                        ▷ See Figure 8.1 and Listing 19.
            sig ← $sig_{L.k^{-1}}$(0, time('now'), mtree.mth)
            sth ← new STH(0, time('now'), mtree.mth, sig)
        for M ∈ Monitors:
            last_sth ← $\epsilon$                              ▷ Last known STH, initially empty string
        for D ∈ Domains:
            (D.k, $D.k^{-1}$) ← new Key_Pair()
        for Cl ∈ Clients:
            log_keys ← {L.k : L ∈ Logs}                        ▷ Clients have logs' public keys
        for A ∈ Auditors:
            A.sth_list ← ∅                                     ▷ initialise set of STHs
```

Listing 23 Scheme of CT, Part 2.

```
Protocol Certification((Ca, D, L)):
    Actor Ca:
        Channels:
            SecChD: Sec_Channel(Ca, D)                         ▷ CA ↔ domain
            ChL: Channel(Ca, L)                                   ▷ CA ↔ log

        Actions:
            Event state = ‘Start’
              and SecChD.recv(name_D: String, k_D: Pub_Key):
                if meets_cps(name_D):
                    cert_D ← new $\mathrm{Cert}_{k^{-1}}$(name_D|k_D)
                    ChL.send(cert_D, root)
                    state ← ‘Wait_SCT’
            Event state = ‘Wait_SCT’ and ChL.recv(sct: SCT):
                SecChD.send(cert_D, sct)
                state ← ‘Start’

    Actor D:
        Channels:
            SecCh: Channel(D, Ca)                              ▷ domain ↔ CA
        Actions:
            Event Start:
                SecCh.send(name, k)
                state ← ‘Wait_Cert’
            Event state = ‘Wait_Cert’ and
              SecCh.recv(cert_new: Cert, sct_new):
                cert ← cert_new
                sct ← sct_new
                state ← ‘Start’

    Actor L:
        Channels:
            Ch: Channel(L, Ca)

        Actions:
            Event Ch.recv(cert_D: Cert, root: Cert):
                if root ∈ root_list  and valid_cert(cert_D) :    ▷ Only verification
                    t ← time(‘now’)
                    sct ← new SCT(h(k), t, cert_D, $\mathrm{sig}_{k^{-1}}$(t|cert_D))
                    mtree.add(cert_D)                  ▷ Update tree, recompute MTH
                    sth ← new STH(mtree.size, t, mtree.mth,
                                      $\mathrm{sig}_{k^{-1}}$(mtree.size|t|mtree.mth))
                    Ch.send(sct)
```

Listing 24 Scheme of CT, Part 3.

```
Protocol Monitoring((M, L)):                                   ▷ L is log to monitor
    Actor M:
        Channels:
            ChL: Channel(M, L)

        Init:
            $k_{log}$ ← L.k

        Actions:
            Event Start:
                ChL.send(‘get_sth’)                        ▷ Fetch current STH from log
                state ← ‘Wait_STH’
            Event state = ‘Wait_STH’ and ChL.recv(sth: STH):
                if valid_sig$_{k_{log}}$(sth.size|sth.time|sth.mth, sth.sig):
                    ▷ If monitor is just starting up:
                    if last_sth = $\epsilon$:
                        ChL.send(‘get_entries’, 0, sth.size)
                        last_sth ← sth
                        state ← ‘Wait_Init_Entries’
                    else if last_sth ≠ sth:                ▷ Log has been updated
                        ▷ Fetch new entries
                        ChL.send(‘get_entries’, last_sth.size + 1, sth.size)
                        state ← ‘Wait_New_Entries’
            Event state = ‘Wait_Init_Entries’ and ChL.recv(entries: {Cert}):
                mtree ← new Merkle_Tree(entries)
                if mtree.mth ≠ sth.mth:
                    . . . ▷ Report misbehaviour (not defined at time of writing)
                else:
                    state ← ‘Start’                        ▷ Monitor is bootstrapped
            Event state = ‘Wait_New_Entries’ and ChL.recv(entries: {Cert}):
                ▷ Availability check
                if entries ≠ ∅:
                    . . . ▷ Monitors may run any checks they want on entries
                    ▷ Now get consistency proof
                    ChL.send(‘get_cons_proof’, last_sth.size + 1, sth.size)
                    state ← ‘Wait_Cons_Proof’
                else:
                    . . . ▷ Report misbehaviour (logs must supply entries)
            Event state = ‘Wait_Cons_Proof’ and ChL.recv(proof: {String}):
                ▷ If consistency proof is correct, monitor is done:
                if verify_cons_proof(last_sth, sth, proof):
                    state ← ‘Start’
                else:
                    . . . ▷ Report misbehaviour (not defined at time of writing)
```

Listing 25 Scheme of CT, Part 4.

```
        Actor L:
            Channels:
                ChM: Channel(L, M)

            Actions:
                Event ChM.recv(‘get_sth’):
                    ChM.send(sth)
                Event ChM.recv(‘get_entries’, from: Integer, to: Integer):
                    entries ← mtree.retrieve(from, to)
                    ChM.send(entries)
                Event ChM.recv(‘get_cons_proof’, from: Integer, to: Integer):
                    proof ← mtree.cons_proof(from, to)
                    ChM.send(proof)

    Protocol TLS_DHE_CT((Cl, D)):
        Actor Cl:
            Channels:
                Ch: Channel(Cl, D)

            Actions:
                Event Start:
                    ... ▷ Normal TLS

                Event state = ‘Wait_Kex’ and Ch.recv($n_D$: Nonce, $cert_D$: Cert,
                  $dh_D$: DHParm, sgn: String, sct: SCT):
                    ▷ SCT received, continue normally
                    ...
                Event state = ‘Wait_Accept’ and Ch.recv(fin_mac: String):
                    ... ▷ Finish handshake normally
                    ▷ Determine log’s public key and verify SCT:
                    $k_{sig}$ ← (k ∈ log_keys : h(k) = sct.$k_{log}$)
                    if valid_sig$_{k_{sig}}$(sct.time|sct.cert, sct.sig)
                      and sct.cert = $cert_D$:
                        ... ▷ Finish normally
                    else:
                        ▷ One of the checks failed
                        state ← ‘Error’

        Actor D:
            Channels:
                Ch: Channel(D, Cl)
            Actions:
                Event state = ‘Start’ and Ch.recv($n_{Cl}$: Nonce):
                    ... ▷ Normal TLS, but we add the SCT:
                    Ch.send($n_D$, cert, $dh_D$, sgn, sct)
                    state ← ‘Wait_Kex’
                ... ▷ Otherwise, normal TLS
```

Listing 26 Scheme of CT, Part 5.

```
Protocol Auditing((Cl, L)):
    Actor Cl:
        Channels:
            ChL: Channel(Cl, L)                              ▷ Auditor to log

        Init:
            $k_{log}$ ← L.k
            last_sth ← $\epsilon$

        ▷ We assume auditor has previously run TLS_DHE_CT and
        ▷ already has an SCT: sct.
        Actions:
            Event Start:
                ChL.send('get_sth')
                if last_sth = $\epsilon$:
                    state ← 'Wait_Init_STH'
                else:
                    state ← 'Wait_STH'
            Event state = 'Wait_Init_STH' and ChL.recv(sth: STH):
                if $\text{valid\_sig}_{k_{log}}$(sth.size|sth.time|sth.mth, sth.sig):
                    last_sth ← sth
                    state ← 'Start'
            Event state = 'Wait_STH' and ChL.recv(sth: STH):
                if $\text{valid\_sig}_{k_{log}}$(sth.size|sth.time|sth.mth, sth.sig):
                    if sth = last_sth:
                        state ← 'Start'
                    else:
                        ▷ Newer STH – fetch audit path. $\text{cert}_D$ is domain cert.
                        ChL.send('get_audit_path', $\text{cert}_D$, sth.size)
                        state ← 'Wait_Path'
            Event state = 'Wait_Path' and
              ChL.recv(index: Integer, path: {String}:
                if verify_audit_path(index, path, sth.mth):
                    state ← 'Start'              ▷ Audit path leads to correct result
                else:
                    ... ▷ Report misbehaviour (not defined at time of writing)

    Actor L:
        Channels:
            ChCl: Channel(L, Cl)

        Actions:
            Event ChCl.recv('get_sth'):
                ChCl.send(sth)
            Event ChCl.recv('get_audit_path', leaf: Cert, size: Integer):
                (index, path) ← audit_path(leaf, size)
                ChCl.send(index, path)
```

Listing 27 Scheme of CT, Part 6.

```
Sessions:
    Dependency TLS_DHE_CT(∗, ∗) not before Certification(∗, ∗, ∗)
    Dependency Auditing(∗, ∗) not before TLS_DHE_CT(∗, ∗)
    At random Certification with $(\mathrm{Ca}, \mathrm{D}, \mathrm{L}) \in \mathrm{CAs} \times \mathrm{Domains} \times \mathrm{Logs}$
    At random Auditing with $(\mathrm{Cl}, \mathrm{L}) \in \mathrm{Auditors} \times \mathrm{Logs}$
    At random Monitoring with $(\mathrm{M}, \mathrm{L}) \in \mathrm{Monitors} \times \mathrm{Logs}$
```

8.6. Assessment of schemes

Based on our findings, we derive a summarising assessment now which scheme is a hopeful candidate for a reinforcement of the X.509 PKI. We base this on the criteria we stated at the beginning of this chapter.

8.6.1. Contributions to security and robustness

In the following, we summarise the contributions of each scheme and contrast this with the strength it shows against attackers from out threat models A, B, and C. Table 8.2 provides a graphical reference for this section.

Protecting clients with out-of-band mechanisms The first criterion is to which degree a scheme can protect clients by using out-of-band mechanisms. Three of the schemes we discussed serve this goal directly, namely TACK, DANE-TLSA, and Perspectives. CT and CAA serve it indirectly.

Among these, TACK is the only scheme that can protect clients even against the strongest attacker. However, the prerequisite is that the first contact to a remote domain must have been secure or the client must have been bootstrapped with secure out-of-band information.

DANE-TLSA does not have this prerequisite, but fails against the two stronger attackers who are able to tamper with DNSSEC-protected resource records in zones that they control.

The security that Perspectives provides depends entirely on the locations of the clients and servers. Perspectives shows a very good strength against the attacker of Model A, although it has some weaknesses during prolonged attacks if the attacker is in control over the final segment of a network path to the server. The regional and global attackers (Models B and C) may additionally use their power to compromise notaries. However, under the assumption that a larger number of notaries exist and shadow notaries are employed, such attempts can be assumed to be detected after a short time. The global attacker does not have an advantage over the regional attacker.

CT does not address security for clients directly, but it allows them to verify that a certificate has been logged. The scheme aims at enabling domain owners, ISPs, and CAs to detect incidents early enough to raise the alarm. Effectively, CT shortens the time window during which an attacker can be successful.

CAA gives a CA additional means to decide whether it should issue a certificate for a given domain or not. This aims at the case where an attacker attempts to obtain a certificate from a CA without actually attempting to compromise it. One drawback of CAA is that it does not mandate DNSSEC. This means it fails in all settings where an attacker controls the network paths between a CA and the responsible DNS server. Even if DNSSEC is used, the protection offered by CAA remains relatively weak. Even the weak attacker may be able to get into a useful position to damage the system, namely if he is able to control the path over which an administrator sets the DNS record. CAA fails entirely for both the regional and global attackers as these can compromise CAs and thus side-step CAA. To summarise this, CAA is only a very indirect protection for clients and the weakest one we discussed.

Incident detection The second criterion we specified was (fast) detection of incidents. Two schemes address this: CAA and CT.

CAA's primary aim is to prevent misissuance of certificates. However, the *iodef* part of the resource record allows to set a contact address where deviation from the configured behaviour can be reported. This is a very interesting contribution as it constitutes an easy way to determine where to direct an incident report. Notably, all other schemes we discuss lack such a reporting mechanism, and CAA is thus useful as a complement to other schemes. We already discussed the strength of CAA in the previous paragraph. Everything that was said there applies to the use of the *iodef* property of the CAA record as well.

CT is a powerful scheme to monitor certification events. The premise is that many CAs opt into the scheme: logs do not exchange information, and without the fundamental understanding that many parties have opted into the scheme, its reach is very limited. CT's cross-validation and constant monitoring and auditing make it extremely difficult for attackers to be successful. Even the regional and global attacker would have to compromise a relatively high percentage of logs and monitors in order to stage a sustained attack.

Monitoring and transparency Our final criteria were whether a scheme facilitates observation of the X.509 PKI, in terms of certification and deployment, and whether it can be assessed via observations itself.

Concerning the latter, almost all schemes we discussed provide great transparency and allow to assess their correct working from the outside. The only exception here is Perspectives: while the concept provides good transparency via shadows, these are not actually implemented.

Concerning observations of X.509, most schemes are not concerned with either observing certification events nor deployment issues. Only one scheme addresses this: CT. Under the assumption that many parties opt in, CT is a very powerful logging mechanism that also tracks historical changes in certification. Active scans can theoretically achieve the same, but are more resource-intensive and cannot detect certification events where the certificate was never deployed. Thus, CT offers somewhat more transparency than what could be achieved with scans alone.

However, CT does not address an issue that we identified as critical in our own scans (see Chapter 4), namely the poor state of deployment, which has historically resulted in too many Web sites showing incorrect certificates and triggering browser warnings. To detect such problems, one will still need active scans.

8.6.2. Issues of deployment

The schemes we discussed have very different requirements to deployment. There are two factors to consider: introduction of additional parties in X.509 processes and number of entities that have to make changes to their software and configurations.

The strongest concept, TACK, is also the one that needs a very high number of entities to opt in, namely server operators. This may or may not be a serious problem. If TACK comes packaged with popular Web servers and can be activated by just setting a small number of options, it may have a reasonable chance to see wide-spread deployment. Clients can simply ignore the TLS extension if they do not support it, which should mediate potential fear on the side of server operators that they may lose clients when enabling TACK. As TACK is a pure client-server concept, it does not introduce new infrastructure.

CAA and DANE-TLSA require changes in the configuration of DNS, but do not introduce any new infrastructure or participants to the X.509 PKI (DNS is already needed by TLS clients to resolve hostnames). They do not require support by servers, either. However, clients only achieve strong security if they validate DNSSEC records themselves and do not leave this to their ISP. This may well be the most important issue to resolve. Recent studies showed that deployment of DNSSEC has grown, although operational problems remain [50]. Much will depend on DNS administration and Web server configuration to remain in a synchronised state. Larger organisations should have less problems with this as they operate their own IT services.

Different considerations apply to Perspectives. The scheme has the benefit that only clients need to support it but servers do not. Clients—especially browsers—are easier to update, whereas server operators always have to consider whether they can still support all clients after they have made a change. Perspectives needs to introduce new infrastructure to the PKI, however, and this may be crucial. Several years after its introduction, the number of notaries remains low and, worse, there are no assurances that notaries that are not run by the project itself are trustworthy (although there is no evidence to the contrary, either). In our discussion of Perspectives, we showed that a concentration on very few notaries is damaging to the concept as a whole. In summary, Perspectives has to overcome higher hurdles than TACK or the DNS-based schemes.

CT, finally, is a particularly interesting case. While it can be rolled out gradually, it needs to win the support of most CAs to become truly useful. As we showed in our discussion, this can be quite difficult with the distribution of SCTs being the central problem. If precertificates are used, CAs need to make an important change to their issuance processes. If OCSP is used, CAs need to invest in changing their OCSP infrastructure. Both are hurdles, although the potential gains seem to outweigh this. If neither precertificate nor OCSP are used, CT shows an entirely different problem: it would need support by server operators as these need to change their software to support the TLS extension. This is probably much harder to achieve than convincing CAs to support the scheme. CT has the unique advantage here that it is supported by one of the largest Internet corporations that currently exist.

8.6.3. Choosing the appropriate candidate

TACK is the scheme that shows the highest resistance to attacks. At the same time, it provides the strongest security for clients. It is a relatively simple scheme that can be further improved by introducing two forms of monitoring of the X.509 PKI. The first form would help with bootstrapping: scanning the most important WWW sites over a longer period, and from different points of view, would help create a set of pins that can be preloaded into user clients. This would be a notary element within TACK. The second form would be monitoring of the tacks that servers offer. The only real issue seems to be the need to make changes to server software, although this is somewhat balanced by the fact that TACK can be rolled out gradually. Our conclusion here is that TACK is a very worthwhile concept to deploy as it is the only one that addresses the global attacker satisfactorily.

We also argue that TACK should be reinforced with at least one strong mechanism to detect attacks. In this category, only CT prevails against stronger attackers. However, CT may suffer from a deployment problem if it cannot win the support of CAs. At this time, it is too early to predict the outcome, but it is worthwhile to note that CT is the only scheme that provides strong guarantees that incidents are detected.

	local, weak (A)	**regional (B)**	**global (C)**
Pinning			
TACK: client protection	succeeds (TOFU)	succeeds (TOFU)	succeeds (TOFU)
DNS/DNSSEC			
CAA: rogue issuance	fails if attacker controls DNS configuration	fails on CA compromise	fails on CA compromise
DANE-TLSA: client protection	succeeds if clients validate records themselves	fails for all TLDs attacker controls	fails for all TLDs attacker controls
Notaries			
Perspectives: client protection	may fail in case of longer attack on path to server	may fail if only few notaries exist	may fail if only few notaries exist
Public logs			
Certificate Transparency: rogue issuance, incident detection	succeeds	succeeds if enough logs, monitors, and auditors exist	succeeds if enough logs, monitors, and auditors exist

Table 8.2. – Summary of contributions the schemes make in reinforcing X.509 and evaluation of robustness against attackers from Models A, B, and C. TOFU is short for Trust-On-First-Use.

We view TACK and CT together as the two schemes that should be used to reinforce the X.509 PKI. They provide the largest benefit while requiring only a reasonable number of changes to enable deployment.

However, this is not to say that the other schemes should not be implemented. CAA, for example, is unique in that it provides a pointer where to report incidents. Concerning DANE-TLSA, much will depend on whether clients will be enabled to validate DNSSEC records themselves. If so, DANE-TLSA is reasonably simple to deploy and gives relatively good security guarantees, except in the case of the state-level attackers.

There are two issues that we see with all schemes. One is that they all fail to consider the deployment of certificates on servers, which we know to be problematic. CT only logs certification events. For the moment, it seems that active scans are the best way to detect faulty configurations.

The other issue is that only one scheme addresses the issue of *reporting* incidents—CAA, which is the least secure of all schemes. Unfortunately, CAA does not prescribe a coordinated way of reporting. Furthermore, it would be very worthwhile to obtain more information about an incident. In particular, it remains unclear in all schemes where the actual attack is happening. The concept that could form a basis here is the notary concept. We address this topic in the next chapter.

8.7. Related work

All discussed schemes are rather new proposals, and thus there is relatively little (academic) work that would have analysed them, in particular with threat models like ours and taking deployment issues into account.

Grant provides a summary of issues with Perspectives, Convergence, DANE-TLSA, CAA, and CT and carries out a brief analysis of the concepts in her thesis [30]. She proposes several categories to rate the systems and then assigns a score to each one. The author does not define or employ a threat model in her analysis, and it is not entirely clear how the score is derived other than being based on a verbal summary of properties. She concludes that *'there is no cure-all for the weaknesses identified in the existing [CA] infrastructure'* and *'the current [CA] architecture is arguably the strongest and most scalable system'* [30]. We agree with the first conclusion, but not with the second, and cite the incidents documented in Chapter 3 as well as the revocation issues of the current PKI (see Section 2.7) as counterexamples.

Soltani provides another analysis of Perspectives, CT, and DANE-TLSA in her Master's Thesis [80]. The schemes are not rated but summarised with respect to security risks, improvements, usability and costs. Deployment is not discussed. The analysis does not use attacker models, either.

In [252], Ritter very briefly summarises the problems of today's X.509 PKI and the key concepts of Perspectives and CT in a white paper. In his conclusion, he emphasises that users are unlikely to change default settings and also states that an important question for deployment is which entities have to make the necessary changes.

Osterweil *et al.* develop a methodology to define and quantify the attack surface of the current CA system versus the CA system together with DANE-TLSA. Their approach is to decompose the respective system into its constituents, expressed as processes and graphs. Using their model, they derive a much smaller attack surface for DANE-TLSA than for the current X.509 PKI.

Gutmann, finally, provides a case study of DNSSEC in general in [86]. His criticism is directed at the fact that use of DNSSEC constitutes the use of a second channel. Should this second channel fail to work, DNSSEC-based name resolution requires the client to abort its operation. The result would be a very poor usability for every DNSSEC-based concept. It is unclear to which extent this argument may hold. Lian *et al.*, at least, showed that the current DNSSEC deployment is slightly more likely to cause failure of name resolution [50].

8.8. Key contributions of this chapter

In this chapter, we addressed Research Objective O3.1. We briefly recapitulate our contributions:

Threat models We defined three threat models based on the attack vectors we identified in Chapter 3. These are the local attacker (close to client or server), the regional attacker with the capacities of a country, and the global attacker, with the capacities of a country plus added possibilities and willingness to control network paths and the DNS.

Schemes and key technologies We described and analysed schemes to reinforce the X.509 PKI, using the notation we introduced in Chapter 7 to highlight participants, their interactions, and the existence of (secure) communication channels. The key technologies the schemes employ are pinning (TACK), DNS/DNSSEC (CAA, DANE-TLSA), notaries (Perspectives), and public logs (CT). Perspectives and CT also use a fifth technology: cross-validation. This greatly improves their robustness against attacks.

Protecting clients Three schemes serve the purpose to protect clients: TACK, DANE-TLSA and Perspectives. TACK is the only one strong enough in the presence of the strongest attacker, but requires the first contact to a domain to be secure. Perspectives provides good security, too, with some weaknesses in the face of very powerful attackers. DANE-TLSA fails in the face of the two stronger attackers.

Schemes to detect incidents CAA and CT serve the primary purpose to detect incidents and raise the alarm. CAA is a weaker scheme, in particular as it does not mandate DNSSEC. It also fails against the two stronger attackers as they may compromise CAs. CT is a very robust construction that is strong enough to resist even the globally active attacker.

Deployment The schemes suffer from deployment issues to varying degrees. Unfortunately, TACK is a scheme that requires many participants, namely server operators, to make changes to their configuration. This hinders deployment. Perspectives has a similar requirement, but for clients. The weakest scheme that protects clients directly, DANE-TLSA, has the lowest requirements for participants, but its success depends on operational practices, namely the synchronisation of WWW and DNS administration.

Highest potential and improvements The two schemes that show the highest potential to reinforce the X.509 PKI are TACK and CT, each for its own purpose. TACK could be improved by creating a set of pins to preload in clients—this requires longer-term monitoring of important WWW hosts. Scans of TACK-enabled hosts can also help add transparency where WWW hosts, and thus WWW services, have been compromised. CT is a very useful tool to detect incidents and determine the responsible CA. However, it lacks a way to determine the position of the attacker.

9 Chapter 9. Crossbear: detecting and locating man-in-the-middle attackers

This chapter is an extended version of our previous publication [38]. Please see the note at the end of the chapter for further information.

9.1. Introduction

We have analysed the state of the X.509 PKI in previous chapters. In Chapter 3, we showed that a number of attacks on CAs have happened in the past, some of which were very successful and could be linked to man-in-the-middle attacks. In Chapter 4, we showed that the X.509 PKI is in a poor state to begin with and in dire need of reinforcement. In Chapter 8, we analysed schemes that aim to strengthen X.509 against attacks and, among other things, investigated the schemes with respect to incident detection. Although at least one scheme provides important functionality in this respect, we found that no scheme so far provides an *automated mechanism for attack detection or reporting*. This correlates with another finding: while we have strong reasons to suspect man-in-the-middle attacks have happened, there is a curious lack of documentation of such incidents. Most reports seem to exist only in the form of blog posts or forum entries. The most famous incident concerned DigiNotar, where the incident was reported in a Google forum [145]. There are also less well-reported incidents like the discovery of a misbehaving wireless access point in a hotel [188]. Generally it seems that affected users are very unlikely to store the rogue certificate they encounter; nor do they provide information how they connected to the Internet during the incident or to which server. Such information, however, would be very valuable. The security community would be able to learn where man-in-the-middle attacks are carried out and possibly even by which parties. Proper documentation would likely help raise public pressure against these parties.

This section presents our own approach to incident detection: Crossbear. It is intended as a response to the lack of hard data and aims to gather information about man-in-the-middle incidents and provide supporting evidence. We pursue two goals with Crossbear, which we define as our research questions:

Automated detection and reporting Detect an ongoing man-in-the-middle attack on the Internet and report this to a central entity. The report will include the position of the victim, the time of the attack, and details about the affected connection.

Localisation and reporting After detection, initiate a process to locate the attacker's position with a fair degree of confidence. The central entity will enrich data from reports with information obtained from other (public) sources to support manual inspection of reports.

9.2. Crossbear design and ecosystem

In the following, we describe how we designed Crossbear and explain how it achieves its goals.

9.2.1. Methodology

We chose to build Crossbear on a notary concept as we know it from our discussion of Perspectives in Chapter 8.4. This is, in part, based on the findings in our analysis: notary concepts allow to detect man-in-the-middle attacks while they are ongoing. The difference to Perspectives is that there is a central Crossbear server with dedicated functionality. Just like Perspectives, Crossbear can also warn users of an ongoing attack and report it automatically. More importantly, however, we make use of the geographical distribution of notaries and add an important twist that 'conventional' notary systems do not share: Crossbear does not just confirm or disconfirm a server's public key. Rather, Crossbear clients are distributed over the Internet and use their position to participate in so-called *hunting*. On request, they can retrieve and compare certificates from a TLS server and record the IP route they have to that TLS server. This is reported to Crossbear's central server, where certificates and routes can be analysed and further hunting initiated. A comparison of the IP routes from hunters that are affected by the man-in-the-middle and by those who are not yields an approximation of the attacker's location in the network. The accuracy increases with the number of hunters that participate.

Based on the incomplete reports of man-in-the-middle attacks, and our findings of X.509 weaknesses from Chapter 3, we designed Crossbear under a working hypothesis concerning the types of attackers we expect to encounter. We assume the first kind of man-in-the-middle attacks to be carried out by an attacker who is operating in close proximity to the victim client (in terms of network hops). This is a subclass of the attacker in our Model A: the attacker is only allowed to occupy a position close to the client, but not close to the server. A typical example would be a poisoned wireless access point or company networks that eavesdrop on incoming and outgoing traffic without benign intent or the consent of their employees. These attackers are in a position to control traffic going into and coming out of a local network. The second kind of attacker is practically identical to the regional attacker of our Model B. We assume an attacker with the power of a country who is in control of the traffic flowing into and leaving a geographically restricted region. A typical example would be a state with tight control over its ISPs and with surveillance devices planted on border routers that connect the networks to the outside world. The typical goal of such an attacker would be to inspect traffic for external services, e. g., Web mailers or social networks.

By its very nature, Crossbear is a tool to counter attacks that are already ongoing. There is a very important effect to take into consideration here: Crossbear is a reactive system, and the attacker always has the first move and may also attempt to counter a step that Crossbear takes. This leads to a practical limitation in Crossbear's effectiveness in locating the attacker (but not in detecting him). We discuss this in Section 9.3.3. Also note that Crossbear must be prepared to defend itself against an attack on its own infrastructure. We discuss this in Section 9.3.4.

9.2.2. Intended user base

There is an important design choice we made: Crossbear is not intended as a tool for common Internet users without deeper technical knowledge and without a willingness to be active in efforts to improve Internet security. Although it provides protection against man-in-the-middle attacks and warns users, this is *not* its primary purpose. Rather, Crossbear is intended as a useful tool for those participants in the security community that may be referred to by the description of 'travelling hacktivists'.

Part of the rationale is that Crossbear users need to be knowledgeable concerning certificates and, to some degree, also concerning TLS. They need to accept a (relatively rare) false-positive warning as an unavoidable annoyance in contributing to the analysis of man-in-the-middle attacks. To allow them to interpret a warning, Crossbear provides them with information about server certificates and its own observation history.

There is a further important point to consider: Crossbear is a tool that has the potential to uncover malicious activities as they may be carried out by repressive governments. As such, these governments may attempt to threaten or even harm users of Crossbear. Any user should be aware of the political situation in the country where he or she chooses to use Crossbear and, when in doubt, should not use it. In particular, we discourage use of Crossbear if the sole goal is protection of the TLS connection. For such users, we recommend tools like Perspectives or Convergence, which provide a similar level of security but are not associated with the counter-analysis activities that Crossbear carries out.

9.2.3. Principle of operation

In the following, we describe Crossbear's principle of operation. We refer the reader to Figure 9.1 for a visual reference.

Key ideas Crossbear is based on two mechanisms: attack detection and attacker localisation. Clients are called *hunters*, and Crossbear deploys a large number of them on the Internet, distributed over as many ASes and networks as possible. We distinguish two forms of hunters: one that carries out both detection and localisation, and one that carries out only the latter.

The first kind of hunter is designed to monitor a user's normal surfing behaviour: for every TLS connection, it queries the Crossbear server whether it encounters the same certificate for the host in question. If it finds a mismatch, it reports this to the server. The Crossbear server creates a so-called *hunting task*, which it sends to the hunter. The hunter will then carry out a traceroute and send the result to the server.

The second kind of hunter is only concerned with localisation. It connects periodically to the Crossbear server to download and execute new hunting tasks. The hunting tasks are executed by connecting via TLS to the reportedly attacked servers, extracting the certificate chain each server sends, and recording the IP route to each server by doing a traceroute. This information is then sent to back to the central server, where it can be analysed.

We implemented both kinds of hunters. The first kind has been implemented as an add-on for the Mozilla Firefox browser. It can additionally also act like the second kind of hunter, i.e., download and execute hunting tasks that are not related to its own detection processes. The second kind of hunter has been implemented as a stand-alone application that can be deployed on test beds like PlanetLab [248].

Detection and localisation When a request for verification of a certificate is received, the Crossbear server determines whether a hunting task should be created. We elaborate on this in Section 9.2.4. Naturally, a user of the browser add-on is warned if an ongoing man-in-the-middle is detected.

The position of the attacker can be approximated by *cross bearing*, i.e., comparing the routes that hunters recorded and determining the intersection points for routes that

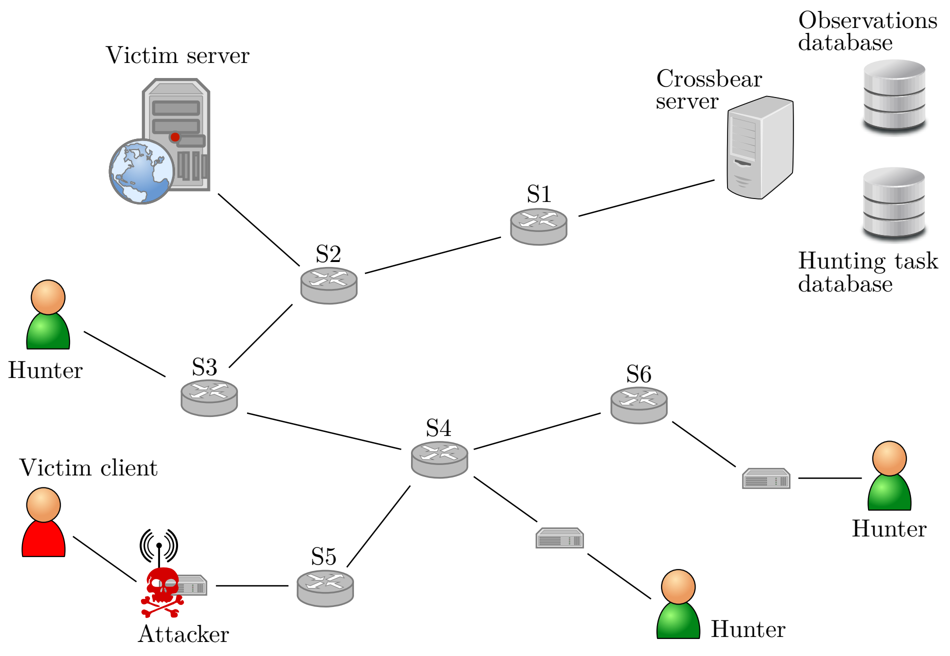


Figure 9.1. – Components of the Crossbear system.

have been found to be poisoned and those that have been found to be clean. This can be done on the router-level or on the level of ASes.

For a visualisation, consider Figure 9.1: the victim client would report a poisoned connection and a certificate mismatch to the Crossbear server. At the same time, the other hunters would report clean connections (and the correct certificate). This allows the Crossbear server to estimate that the attacker is located in the vicinity of the victim, most likely on system S5, because it connects the victim client to S4, which we know not be to compromised thanks to the reports from other hunters. This accurate way of cross bearing works in many, but not all scenarios; we discuss its effectiveness against attackers at various positions and with different strategies in Section 9.3.

Further vantage points Crossbear also retrieves observations made by other notary systems. Currently, Crossbear uses Convergence [268] as a source of independent observations from other vantage points. At the time Crossbear was developed, Convergence was more popular than the related Perspectives project. Information from Convergence notaries is also always forwarded to users of the Firefox add-on in order to allow them to make a more accurate assessment of a certificate used in a TLS connection.

At the time of writing, we found the Perspectives project to be gaining momentum again and offering notary servers that could be used by Crossbear. Whether Crossbear will additionally enable use of Perspectives or not has not been decided yet.

Out-of-band information sources Reports about alleged attacks consist only of the encountered certificates and the network paths from hunters. To facilitate better analysis of reports, the Crossbear server carries out several more lookups and stores information from other sources every time a report is received. We currently use the following sources:

AS and WHOIS We retrieve the AS number of all hosts in a traceroute and retrieve further information with WHOIS queries, e.g., the name and registered location of the organisation running the AS.

Geolocation Hosts in a traceroute are looked up in geo-IP databases. Although imperfect, this gives us a rough estimate which countries were on the network path from hunter to alleged victim server.

CAs used We store which CAs a domain seems to use. The motivation is that certain domains like, e.g., Google have always remained customers of the same CAs for longer periods of time[1]. We set triggers when the issuing CA for a domain changes as this could be an indication of an attack.

Certificate properties We extract certificate fields from each certificate we receive. This helps us determine whether two forged certificates share similar properties. Fields like issuer, public key, and serial number have the highest relevance here. The rationale is that attackers may use similar configurations when creating forged certificates. Naturally, this applies (mostly) to certificates that attackers forge without prior compromise of a CA.

Clustering reports by source Thanks to the information we store about the countries from which reports reach us (geographic position and official country of registration of the corresponding AS), it becomes possible to identify clusters of reports from the same country.

We elaborate on the use of this information in Section 9.3.

9.2.4. Details of detection and hunting processes

Typically, man-in-the-middle attacks are detected with the add-on for the Web browser. In the following, we provide details about this process. Listings 28–33 provide a reference in our notation.

Protecting the communication with the server

All Crossbear clients (add-ons and hunters) communicate with the Crossbear server via TLS. To protect this channel against man-in-the-middle attacks, the server's public key is hard-coded into clients. We express this as secure channels in lines 76 and 205. If a client finds that the key in the received server certificate does not match the hard-coded one, its current behaviour is to refuse to operate and offer the user to send an automatic email to the Crossbear team that contains all details about the incident (including the forged certificate). We do not show this latter feature in our listings.

Convergence as a service

We represent Convergence as a service. As is also the case in our current implementation, we only use two Convergence notaries. When queried, the service returns the observation histories of the notaries. Note that we do not give a full description of Convergence in our listings but represent it in the style of the API that we use in our implementation. Convergence's methodology is very similar to Perspectives, however. The service is shown in lines 62–71.

Certificate verification

Listings 30–33 show how certificate verification via the Crossbear server works. In the following, we use the term client to refer to a hunter implementation in the browser add-on, i.e., a hunter that implements both hunter types (see Section 9.2.3).

[1]For Google, this has been confirmed to the author in private email.

When a client connects to a Web server via TLS and the connection is under attack by a man-in-the-middle, it may receive a forged certificate (lines 227–228). Thus, the client always sends a request to the Crossbear server at the end of the handshake to verify the certificate (line 236). The message includes the observed certificate and the domain name of the server. When the server receives this message (line 145), it connects to the domain itself (line 148) and stores the result. In the next step, it queries the Convergence service for known certificates for the domain (line 153). When the response arrives, it checks the signatures (lines 160–162). If the server's observation matches the certificate and Convergence reports it has also observed the certificate in question, the Crossbear server stores the certificate in its observation database. This is shown in lines 165–178. It does not do this if it has conflicting views or Convergence never encountered the certificate. This is a protection against an attacker who tries to flood the Crossbear server with observations of a rogue certificate in an attempt to tamper with the score Crossbear computes (we elaborate on the latter below).

The Crossbear server will create a hunting task in one of two cases (lines 180–185). Either, the server was not able to carry out a handshake with the domain in question. This may be an indication that traffic may be dropped somewhere and is thus a reason to investigate. The other case is when the Crossbear server encountered a different certificate *and* Convergence never encountered the reported certificate, either.

The Crossbear server sends the result of its observation back to the client in a so-called *assessment* (lines 187–201). In particular, the assessment includes whether the Crossbear server encountered the same certificate in its handshake to the domain. If the server decided to create a hunting task, it sends it together with the assessment to the client. We explain the information in the assessment next.

Assessment and score

The Crossbear server sends information to the clients that allows them to compute a score. The motivation for this score is to give the human user a quick summarising view of what might be happening to his or her TLS connection, while the assessment itself is meant to allow them to make an informed decision about whether to continue with the connection. In our notation, the evaluation of the assessment is defined in a procedure (lines 208–221).

The score is a weighted sum over a number of properties. The primary criterion in the score is the comparison of the certificates that client and server have encountered, which is sent as a *String* that may be either 'yes', 'no' or 'none'. We also take the *last continuous observation period (LCOP)* into account, which is very similar to what Perspectives and Convergence compute, except that it is expressed as an interval. The LCOP states for how long *only* the certificate in question has been observed. Further criteria are the number of previous observations and the LCOP that Convergence reports (which is 0 if Convergence has no record of the certificate). Our weights are chosen thus that 'critical' combinations of properties yield a score of less than 100. This is a threshold value that is user-adjustable. When a score is below the threshold, the add-on displays a warning, together with the data in the assessment. We list the relevant factors for the score in Table 9.1. Each information item is sent by the server to the client in the assessment.

When the certificate score is above the threshold, the add-on accepts and caches the observed combination of host and certificate. The latter is a performance optimisation that we do not include in our listings. When the threshold is not reached and a warning is displayed, the user is asked if the combination should be exempted (accepted) and cached. We show this in lines 240–243. If the server also sent a hunting task, the client it going to execute it by doing a traceroute, just like any hunter (line 246).

Property and score	**Rationale**
Certificate comparison:	
80 if $C_c = C_s$	S observes same certificate
0 if $C_c \neq C_s$	Potential attack
-100 if S cannot get certificate from V	S likely blocked
LCOP:	
$\frac{\text{days}\cdot 2}{3}$ if LCOP ongoing	C_c still observed
$\frac{\text{days}}{3}$ if LCOP ended in the past	C_c observed in the past only
Observations:	
$\frac{\text{count}}{30}$	High number of observations makes certificate more trustworthy
Convergence:	
$\frac{\text{days}\cdot 2}{3}$ if certificates match	Confirmation
-20 if never observed	Weak indication of attack
0 if no reply from Convergence	Inconclusive

Table 9.1. – Parameters used in computing a score for a reported certificate. C_c is the certificate observed by the client, C_s the certificate observed by the Crossbear server (S). V is the victim server.

We experimented with our default settings over the course of several months and found that false positives occur rarely enough to be acceptable for the savvy user base that we assume.

Details of the hunting process

Hunting is the process of determining a suspected attacker's network location, i.e., his position in an AS, subnetwork or (with the help of a geo-IP database) approximate geographic position. The latter is done by the Crossbear server; we use well-known processes in our notation to describe the corresponding lookup processes. Table 9.2 provides a summary of them.

In the following, we describe the hunting process as it is implemented in the second kind of hunter (see Section 9.2.3). Note, however, that the first kind of hunter naturally also includes this functionality (see above). Listings 29 and 30 show the hunting process. Every hunter pulls the list of active hunting tasks from the Crossbear server at regular intervals (line 79 f., called from the *Sessions* block in line 262 ff.). The hunter then executes the tasks (lines 81–93). The hunting starts with a full TLS handshake with the alleged victim server to extract the certificate chain. The next step is to record the route that IP packets take towards the destination. This is done with a standard ICMP traceroute (line 89). Certificate chain and route are reported to the Crossbear server (line 92).

When the server receives results from hunting tasks (line 103), it will update its database of hunting tasks (lines 104–118). For each hunting task, it extracts the traceroute and then carries out lookups for each hop: geographical location, AS number and WHOIS information. This is stored in the database.

The Crossbear server deactivates hunting tasks that are older than a certain time or for which enough results have been received. This is shown in the *Protocol Maintenance* in lines 252–260.

Listing 28 Scheme of Crossbear, Part 1.

```
Scheme Crossbear:
    Participants:
        Domains : $\{D_0, D_1, \dots\}$                          ▷ Set of all domains
        Clients : $\{Cl_0, Cl_1, \dots\}$                        ▷ Browsers
        Hunters : $\{H_0, H_1, \dots\}$                          ▷ Stand-alone hunters
        Server : $\{S\}$                                         ▷ Crossbear server
        Cnv_Ntr : $\{Cn_0, Cn_1\}$                               ▷ Two Convergence notaries

    Record Observation:                                          ▷ Crossbear observation
        cert : Cert
        times : {Timestamp}

    Record Period:                                               ▷ Represents a time interval
        min : Timestamp                                          ▷ Start of period
        max : Timestamp                                          ▷ End of period

    Record Cnv_Observation:                                      ▷ Convergence's observations
        hash : h(Cert)
        p : Period
        sig : sig(h(Cert)|Period)

    Record Assmnt:                                               ▷ Assessment, sent to client
        domain : String                                          ▷ Domain name
        match : String                    ▷ Result from Crossbear server: matching cert?
        $lcop_{CB}$ : Integer                                    ▷ LCOP from Crossbear server
        $lcop_{Cnv}$ : Integer                                   ▷ LCOP from Convergence
        ongoing : Boolean                                        ▷ Is cert observation ongoing?
        num_obs : Integer                 ▷ Number of observations by Crossbear server

    Record Hunting_Task:
        domain : String
        t : Timestamp
        active : Boolean

    Record Hunting_Result:                                       ▷ Result, sent by hunter
        $ht_{name}$ : String              ▷ Name of hunting task (acts as identifier)
        t : Timestamp
        cert : Cert                                              ▷ Encountered cert
        trace : Map⟨Integer, String⟩                             ▷ Traceroute

    Record IP_Info:                       ▷ Additional information about an IP address
        ip : String                                              ▷ IP address
        geoloc : String                                          ▷ Geographic location
        asn : String                                             ▷ AS number
        whois : String                                           ▷ Information from WHOIS
```

Listing 29 Scheme of Crossbear, Part 2

```
Record Ext_Hunting_Result:        ▷ This is how Crossbear server stores results
    hunter_ip : String                                      ▷ Reporting hunter
    hunting_time : Timestamp                                ▷ Time of hunting
    domain : String                                         ▷ Domain in question
    cert : Cert                                             ▷ Certificate observed
    ext_trace : Map⟨Integer, IP_Info⟩           ▷ IP_Info for every hop on trace

Init:
    S.history ← new Map⟨String, {Observation}⟩()              ▷ Observations
    S.hunting_db ← new Map⟨Hunting_Task, {Ext_Hunting_Res}⟩()
    for D ∈ Domains:
        D.cert ← new Cert()

▷ The Convergence API allows to obtain observations as defined in record
▷ Cnv_Observation. Convergence returns a set of Cnv_Observation for the
▷ domain. We assume these are stored internally in a field ‘history’.
Service Cnv represents Cnv_Ntr:
    Channels:
        Chs : {Channel(N, S) : N ∈ Cnv_Ntr}                ▷ S is Crossbear server
    Actions:
        ▷ When asked, return the notaries’ histories:
        Event Ch ∈ Chs: Ch.recv(‘get_hst’, $name_D$: String):
            hst_map ← ∅
            for N ∈ Cnv_Ntr:
                hst_map[N.name] ← N.history[$name_D$]
            Ch.send(hst_map)

Protocol Hunting((H,S)):
    Actor H:
        Channels:
            SecCh: Sec_Channel(H, S)        ▷ S’s cert is hard-coded into hunter!
        Actions:
            Event Start:
                SecCh.send(‘get_ht_list’)              ▷ Request hunting task list
                state ← ‘Wait_HT’
            Event state = ‘Wait_HT’ and Ch.recv(ht_list: {Hunting_Task}) :
                results ← ∅
                for ht ∈ ht_list:
                    ▷ Connect via TLS_DHE to domain and retrieve cert:
                    $D_{ht}$ ← (d ∈ Domains: d.name = ht.domain)
                    TLS.TLS_DHE(H, $D_{ht}$)              ▷ $D_{ht}$’s cert now in $cert_D$
                    t ← time(‘now’)
                    ▷ Do a traceroute to domain:
                    $tr_D$ ← traceroute($D_{ht}$)
                    $res_{ht}$ ← new Hunting_Result(ht.name, t, $cert_D$, $tr_D$)
                    results ← results ∪ {$res_{ht}$}
                SecCh.send(results)
                state ← ‘Start’
```

Listing 30 Scheme of Crossbear, Part 3

```
    Actor S:
        Channels:
            SecCh: Sec_Channel(S, H)

        Actions:
            Event SecCh.recv('get_ht_list'):
                ▷ Retrieve active hunting tasks:
                ht_list ← {ht ∈ hunting_db.keys() : ht.active = true}
                SecCh.send(ht_list)
            Event SecCh.recv(hunting_results: {Hunting_Result}):
                for r ∈ hunting_results:
                    ▷ Fetch corresponding hunting task:
                    ht ← (h ∈ hunting_db.keys() : h.name = $r.ht_{name}$)
                    ▷ Create Map for extended traces with additional info:
                    ext_trace ← new Map⟨Integer, IPInfo⟩()
                    for hop ∈ r.trace.keys():                ▷ Loop over all hops
                        geo ← lookup_geo(r.trace[hop])
                        asn ← lookup_as(r.trace[hop])
                        whois ← lookup_whois(r.trace[hop])
                        ext_trace[hop] ←
                                     new IP_Info(r.trace[hop], geo, asn, whois)
                    ext_res ← new Ext_Hunting_Result
                                     (hunter_ip, r.t, ht.domain, r.cert, ext_trace)
                    new_res ← hunting_db[ht] ∪ {ext_res}
                    hunting_db[ht] ← new_res

Protocol TLS_CB((Cl, D), S):
    Actor S:
        Channels:
            Ch: Sec_Channel(S, Cl)                   ▷ Crossbear server ↔ client
            Cnv_Ch: Channel(S, Cnv)                  ▷ Channel to Convergence
        Init:
            cnv_keys ← new Map⟨String, Pub_Key⟩()
            for N ∈ Cnv_Ntr:
                cnv_keys[N.name] ← N.k

        Procedure assess_cnv(obs: {Cnv_Observation}):
            if obs = ∅:
                return 0
            return longest_period($\bigcup_{o \in obs}$ {o.p})
        Procedure longest_period(pds: {Period}):
            if pds = ∅:
                return 0
            ▷ Get the periods of longest duration:
            $pds_{long}$
              $\leftarrow \{p_i \in pds : (\forall p_j \in pds, i \neq j : p_i.max - p_i.min \geq p_j.max - p_j.min)\}$
            ▷ All resulting periods are of equal length, choose any:
            return $(p.max - p.min), p \in pds_{long}$
```

Listing 31 Scheme of Crossbear, Part 4

```
        **Actions:**
          ▷ Cert verify request received:
          **Event** state = Start
            **and** SecCh.recv(‘ver_req’, $name_D$: String, $cert_{D,Cl}$: Cert)**:**
              D ← (d ∈ Domains : d.name = $name_D$)
              $cert_D$ ← ∅
              TLS.TLS_DHE(S, D)                          ▷ TLS handshake with D
              ▷ D’s cert is now in $cert_D$.
              ▷ If TLS failed, $cert_D$ = ∅.
              ob_time ← time(‘now’)
              ▷ Observations from Convergence:
              Ch_Cnv.send(‘get_hst’, $name_D$)
              state ← ‘Wait_Cnv’
          **Event** state = ‘Wait_Cnv’
            **and** Ch_Cnv.recv(hst: Map⟨String, {Cnv_Observation}⟩)**:**
              cnv_obs ← ∅
              own_ob ← ∅
              all_obs ← history[$name_D$]                ▷ All observations of S
              **for** $name_N$ ∈ hst.keys()**:**                  ▷ Check signatures
                  $k_N$ ← cnv_keys[$name_N$]
                  $obs_{val}$ ← $\{o \in hst[name_N] : valid\_sig_{k_N}(o.k|o.pds, o.sig)\}$
                  ▷ Filter observations for cert in question:
                  cnv_obs ← cnv_obs ∪ $\{o \in obs_{val} : o.hash = h(cert_{D,Cl})\}$
              cert_match ← ‘no’
              **if** $cert_D$ = $cert_{D,Cl}$**:**
                  cert_match ← ‘yes’
              **else if** $cert_D$ = ∅**:**
                  cert_match ← ‘none’
              ▷ Store observation if certs match and Convergence confirms
              **if** cert_match = ‘yes’ **and** |cnv_obs| ≠ ∅**:**
                  ▷ Fetch own observation:
                  own_ob ← (o ∈ all_obs : o.cert = $cert_{D,Cl}$)
                  **if** own_ob ≠ ∅**:**          ▷ Add new timestamp for observation
                      own_ob.times ← (own_ob.times ∪ {ob_time})
                  **else:**                              ▷ Create new observation
                      new_ob ← new Observation($cert_{D,Cl}$, {ob_time}
                      history[$name_D$] ← {new_ob}
              ▷ Create hunting task:
              ht ← ∅
              **if** cert_match = ‘none’
                **or** (cert_match = ‘no’ **and** |cnv_obs| = ∅)**:**
                  ht ← new Hunting_Task($name_D$, ob_time, True)
                  ▷ Store hunting task as key in hunting_db:
                  hunting_db[ht] ← ∅
              ▷ Prepare assessment. First, get times. . .
              $times_{all} \leftarrow \bigcup_{o \in obs\_all} o.times$          ▷ . . . for all observations
              $times_{cert} \leftarrow \bigcup_{o \in own\_obs} o.times$      ▷ . . . for observations for cert
```

Listing 32 Scheme of Crossbear, Part 5

```
                ▷ Is observation ongoing (latest timestamp)?
                ongoing ← False
                if $times_{cert} \neq \emptyset$ and $\max(times_{all}) = \max(times_{cert})$:
                    ongoing ← True
                ▷ Determine LCOP. Choose continous periods first:
                cont_pds ← $\{$new Period$(t_1, t_2), t_1, t_2 \in times_{cert}$ :
                                $(\nexists t \in times_{all} : (t \notin times_{cert} \wedge (t_1 < t < t_2)))\}$
                $lcop_{CB}$ ← longest_period(cont_pds)
                $lcop_{Cnv}$ ← assess_cnv(cnv_obs)
                ▷ Send assessment:
                assmnt ← new Assmnt($name_D$, cert_match, $lcop_{CB}$, $lcop_{Cnv}$,
                                        ongoing, |own_obs|)
                SecCh.send(assmnt, ht)

        Actor Cl:
            Channels:
                Ch: Sec_Channel(Cl, S)
                Ch: Channel(Cl, D)

            Procedure assess(a: Assmnt):
                if a.match = ‘yes’:
                    score ← 80
                if a.match = ‘none’:
                    score ← -100
                score ← score + $\lfloor \frac{a.num\_obs}{30} \rfloor$
                if $a.lcop_{Cnv} = 0$:
                    score ← score − 20
                else:
                    score ← score + $\lfloor \frac{a.lcop_{Cnv}}{86400} \cdot \frac{2}{3} \rfloor$
                if a.ongoing:
                    return $(score + \frac{(a.lcop_{CB}.max - a.lcop_{CB}.min)}{86400} \cdot \frac{2}{3})$
                else:
                    return $(score + \frac{(a.lcop_{CB}.max - a.lcop_{CB}.min)}{86400} \cdot \frac{1}{3})$

            Actions:
                Event Start:
                    ▷ Normal TLS until end of handshake
                    ...
                Event state = ‘Wait_Kex’ and Ch.recv($n_D$: Nonce, $cert_D$: Cert,
                  $dh_D$: DHParm, sgn: String):
                    ▷ The server certificate may be forged.
                    ▷ We check this at the end of the handshake.
                    ... ▷ Continue for now.
                Event state = ‘Wait_Accept’ and Ch.recv(fin_mac: String):
                    ▷ Normal TLS checks
                    ...
                    ▷ And now, start check with Crossbear server
                    SecCh.send(‘ver_req’, D.name, $cert_D$)
                    state ← ‘Wait_Assmnt’
```

Listing 33 Scheme of Crossbear, Part 6

```
            **Event** state = ‘Wait_Assmnt’
             **and** SecCh.recv(assmnt: Assmnt, ht: Hunting_Task):
              **if** assess(assmnt) $\geq$ 100:
                 ... ▷ Score large enough, finish normally
              **else:**
                 ... ▷ User decides whether to accept this connection
              **if** ht $\neq \emptyset$:
                 ▷ Client acts as hunter and executes steps from
                 ▷ Hunting protocol, beginning with line 88
                 ...

      **Actor** D:
         ... ▷ No changes to domains

   **Protocol** Maintenance(S):
      **Actor** S:
         **Actions:**
            **Event** Start:
               ▷ Deactive hunting tasks if enough results or too old:
               three_days ← 259200                        ▷ 3 days in seconds
               **for** ht $\in$ hunting_db.keys():
                  **if** |hunting_db[ht]| > 30  **or** (ht.t - time(‘now’) > three_days):
                     ht.active ← False

   **Sessions:**
      **At random** Hunting **with** (H $\in$ Hunters, S)
      **At random** Maintenance **with** S
      **At random** TLS_CB **with** ((Cl, D) $\in$ Clients $\times$ Domains, S)
```

Name	Explanation	Result
traceroute(d: Domain)	Traceroute	*Map⟨Integer, String⟩* mapping hop number to IP for *d*
lookup_geo(ip: String)	Lookup in geo-IP database	*String* representing position
lookup_asn(ip: String)	Lookup of AS number for IP	*String* representing AS number
lookup_whois(ip: String)	WHOIS lookup for IP	*String* representing WHOIS result

Table 9.2. – List of well-known processes in the Crossbear scheme.

9.2.5. Simplifications for the representation in our notation

The current implementation of Crossbear makes some performance optimisations that we did not describe yet and did not include in our notation.

Before a hunter can send results to the Crossbear server, it must obtain a so-called *public IP notification* from the server. This data structure contains the public IP address of the client *that the Crossbear server observes* plus a HMAC of it, keyed with a secret key that only the Crossbear server knows and changes every 30 minutes. It can be obtained via a simple HTTP request to the server. The rationale is as follows: hunters pull the hunting task list relatively frequently, but skip hunting tasks which they have executed recently from the current IP address. However, hunters may be positioned behind Network Address Translation (NAT) devices like middle-boxes, and thus they need to know their public IP address to decide whether a hunting task can be skipped or not.

A further optimisation that we make is that the Crossbear server sends hash values of known certificates for the domain in question with each hunting task. This allows hunters to simply reply with the hash value instead of the full certificate unless they encounter a previously unknown certificate.

Although we do not show this in the notation, the Crossbear server actually stores all received information greedily, including observations of certificate mismatches. It just does not use this information in its internal counts, e.g., the number of previous observations.

9.3. Analysis and discussion of effectivity

In the following, we analyse the degree to which Crossbear can be an effective tool. We also discuss counter-attacks against Crossbear.

9.3.1. Attacker model

We first define the threat model for Crossbear. Our attackers are refined versions of the attackers described in Section 8.1. In general, our attacker is always assumed to have the full control over a 'system' on the path from the client to the victim server. A system can be either a router or an entire AS through which traffic is forwarded. The attacker does not control any other path in the network. An attacker controlling several systems is modelled as separate attacks carried out by the same attacker. The attacker can 'impersonate' IP addresses (i. e., spoof them *and* intercept replies addressed to them) from the system he controls or systems that are attached to it, and whose upstream and downstream traffic is routed through it.

We structure our discussion along two dimensions. First, we distinguish attacker types by their selectivity against clients:

Non-selective attacker: The non-selective attacker stages his man-in-the-middle attack against all clients attached to his system.

Selective attacker: The selective attacker stages his attack against a *subset* of clients attached to the system he controls.

Second, we distinguish by the position of the attacker in the network. The positions we consider are essentially those in our threat models from Section 8.1.

Localised attack We assume one kind of attacker to be either positioned towards the periphery of the Internet and close to the client, or towards the periphery and close to the victim server (Model A).

Regional attack Furthermore, we consider the regional attacker of Model B who is in control of border routers or entire ASes.

Attack in the core Finally, we also allow the attacker a position in the core of the network. This is a refined version of the attacker in Model C.

Our focus will be on the first two positions: these were the positions of the alleged attackers in the reports we have, such as [188, 183, 145].

Figures 9.2(a)–9.2(d) depict attackers at different positions and with either selective or non-selective behaviour. Figure 9.2(a) shows a non-selective attacker who operates close to the client, e.g., a poisoned wireless access point. Figure 9.2(b) shows the much more powerful but still non-selective attacker who controls an entire system to which several subsystems are attached. This corresponds to the regional attack.

Figures 9.2(c) and 9.2(d) show attackers that cross bearing, and indeed any tracing system, are less effective against. Figure 9.2(c) depicts a selective attacker that is located close to the client but acts only against a subset of the clients attached to the system he controls. Note that this is still an attacker in our Model A, just with a different behaviour. Figure 9.2(d) is a very powerful and cunning attacker: he is in control of an important system in the Internet core (e.g., an important transit AS) and stages his attack against just a subset of client systems at the periphery. This is our Model C, once again with a special kind of behaviour on the attacker's side. A possible example is state-condoned industrial espionage where a government agency stages a man-in-the-middle attack on traffic passing through their AS. Note that man-in-the-middle attacks become more difficult the more the attacker moves towards the core of the network: the attacker needs to modify both directions of the traffic; but phenomena such as hot-potato routing [71] and BGP peering policies like valley-free routing [63] often cause IP packets to take different return paths.

We discuss now how effective Crossbear is for each scenario and which additional steps can be taken to aid detection and localisation.

9.3.2. Detection

In general, all ongoing man-in-the-middle attacks can be reliably detected by Crossbear because the queried Crossbear server observes a different certificate for the victim server. This is true for all attacker types we focus on and for all attackers in Figure 9.2, except for the last, and then only if he chooses to manipulate BGP (see below). Note that if the attacker chooses to attack the connection to the Crossbear server, this is detected and the add-on will react to it (see Section 9.2.4).

The only attack that cannot be reliably detected by certificate comparison is when the attacker is on all paths to the victim server. This is a weakness all notary systems share. Such an attacker would either have to hijack BGP routes (as proposed in [199]), and covered by our Model C) or position himself at a point in the network where all paths to the destination have already converged, i.e., close to the victim server. If the victim server has been observed previously, however, Crossbear can still profit from historical information available at the server and from previous observations by the Convergence notaries. When important certificate properties like the issuing CA change, this will flag a client report for manual verification.

9.3.3. Localisation

Excluding the attacker who controls all paths to the victim server, the ability to accurately trace the attacker's position in the network depends entirely on the attacker

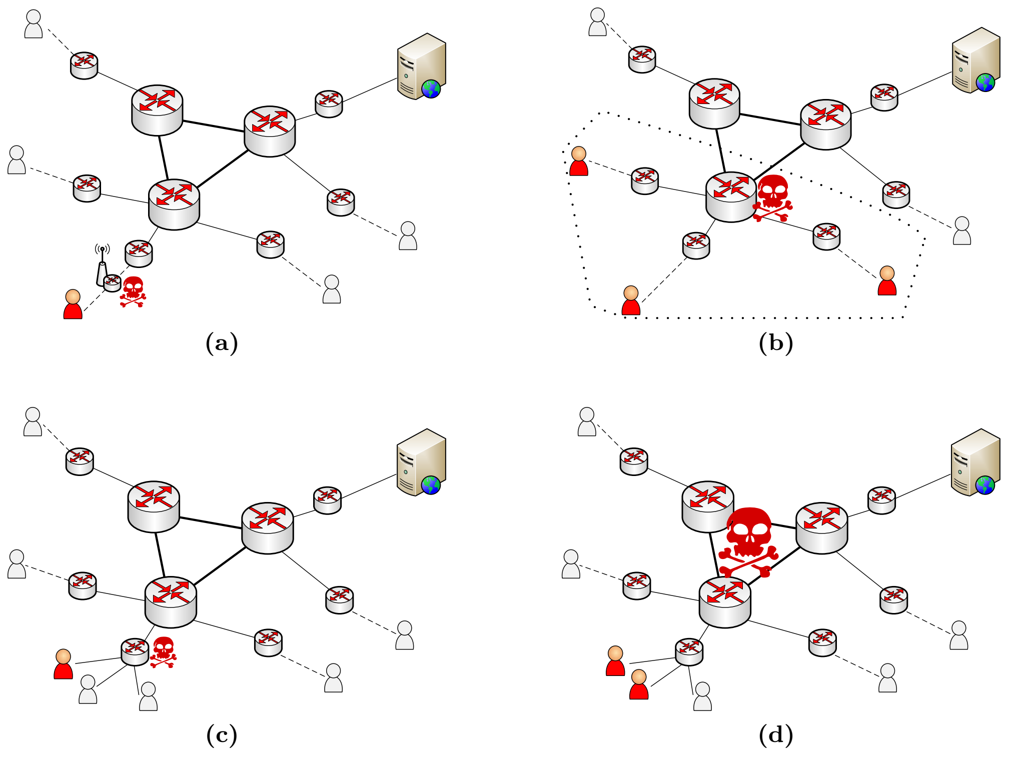


Figure 9.2. – (a) Non-selective attacker in vicinity of client. (b) Non-selective state-level attacker. (c) Selective attacker in vicinity of client. (d) Selective (super-)attacker in core of network.

acting selectively or non-selectively. In the following, we describe localisation for the non-selective and the selective attacker.

The non-selective attacker

The non-selective attacker lends itself well to localisation. In order for this to work, Crossbear needs a traceroute from the victim client and from at least one hunter that is attached to an upstream system (from the attacker's point of view) which reports a clean connection. The accuracy increases the closer that upstream system is towards the attacker's own position and if that view is corroborated by other hunters either downstream (reporting poisoned connections) or upstream (reporting clean connections). Giving a precise estimate of the accuracy is a difficult undertaking as there is a definite lack of data about routing paths on the Internet. However, it is still possible to give a *rough* estimate of how many hunters are required in order to locate a non-selective attacker. To this end, we require a number of simplifying assumptions to make the model suitable for analysis, a fact that we fully acknowledge.

Basic observation In the following, we derive a closed-form model to estimate the average number of hunters needed to locate a man-in-the-middle attacker with a certain probability. Our analysis is based on an observation that holds for most Internet traffic: once two traffic flows with the same destination converge at a point in the network, they will not separate again until they reach their target. This is a characteristic of standard IP routing, which is based on the destination but not on the source address. Exceptions exist (e.g., ECMP [122]) but are rare; hence our model will hold for most cases. Given a path from victim client to victim server via an attacker, traceroutes from hunters will join the path at some point. Due to the genericity of our model, we can apply it at router level (i.e., to find the router conducting the man-in-the-middle attack) as well as at AS level (i.e., to find the AS conducting the attack). In the following, we will thus use the generic term *node* to denote a router or an AS. Our model will only require the distribution of path lengths between victim client and victim server and the distribution of node degrees as input. Such data can be derived from publicly available sources and from measurements from our own university network.

Closed-form model for estimating the number of hunters We construct our model for the case of exactly one victim client, called C, and one victim server, called V. We use the symbol $\leftrightsquigarrow$ to denote a path between source to destination, i.e., an ordered set of nodes, and write $C \leftrightsquigarrow V$. We denote the nodes on the path as X_j. $X_1 = V$ is the victim server, and X_2 is connected to X_1, and so forth. X_ℓ is connected to C. Figure 9.3 shows a visualisation, with $X_\ell = X_7$. We now denote an ongoing man-in-the-middle attack on the path $C \leftrightsquigarrow V$ by denoting the position of the attacker on the path as $M =: X_m$. In our example in Figure 9.3, we have $M = X_5$.

We now let the hunting process begin, i.e., we assume that C has caused a certificate verification by the Crossbear server S, and S found a mismatch according to the conditions we defined (certificate mismatch and certificate unknown to Convergence). Hunters $H_1, \ldots, H_n$ begin their work and carry out TLS handshakes with V, followed by tracerouting. The question we want to answer is what value n must take for localisation to be accurate.

Figure 9.3 gives this intuition. The attacker can be accurately located if, at the very least, traffic from a hunter joins the path to V exactly at X_m (in our example, this hunter would be H_1) and if traffic from another hunter joins the path at the first unpoisoned system X_{m-1} (here: H_2 and X_4). The goal of our model is thus to give a formula to estimate the probability of hunters being placed in these positions.

Assumptions We need to make some simplifying assumptions in order to be able to derive a model that can be evaluated with the few data sources that are available about Internet routing:

1. We assume the attacker behaves as described in our threat model and there is only one point in the network where the interception takes place: node M.

2. The attacker must work non-selectively.

3. We assume symmetric traffic paths, i.e., $V \leftrightsquigarrow C$ is symmetric, even if the path leads over M or a hunter H_i. This assumption may often not hold in real-world routing, where paths can be asymmetric. However, our final model will only depend on path lengths. This means that even if asymmetric routing occurs, our model will still remain relatively accurate if the lengths of the asymmetric paths are not significantly different.

4. We assume packets are forwarded based only on their destination addresses. Only the attacker M is exempted here as he may divert traffic. In particular, if two traffic flows have converged at some node and have the same destination node, they will not separate again. For real-world routing, this assumption will mostly hold. Exceptions may occur, however. One example is so-called hot-potato routing on the level of ASes when traffic streams from two nodes have ingress points far away from each other.

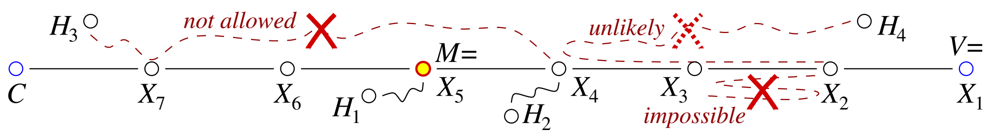


Figure 9.3. – Visualisation to derive closed-form model.

5. We assume the hunters are distributed uniformly over the network—i.e., the likelihood that a hunter is placed at a certain location is the same as for any other location. In real-world scenarios, this assumption may often not hold as Crossbear clients are more likely to be distributed in stub ASes. Different distribution functions could be used, but they would need to pass a plausibility test, too. They would also complicate the model considerably and make it much less accessible to analysis.

6. Finally, we assume the following. Let H_1 be a randomly placed hunter node carrying out a traceroute to S. Let X_j be an intermediate node on the path, and X_{j-1} its successor. We denote the probability that traffic from H_1 passes X_{j-1} as $\Pr[X_{j-1}]$. We now assume that X_j forwards traffic to all its neighbours with equal probability (this includes successor X_{j-1}). The only node that will not receive its traffic is X_{j-2} as this would imply a routing loop: $H_1 \rightsquigarrow X_{j-2} \rightarrow X_{j-1} \rightarrow X_{j-2}$. In Figure 9.3, we label this the 'impossible' path.

Probability that a hunter covers a node With these assumptions in mind, our first step in deriving the model is to determine the probability that traffic from a hunter H_1, which is placed randomly, will traverse a given intermediate node X_j. We only analyse one traffic direction here, namely the one that is of most interest: from sources towards the server V.

We denote the degree of node X_j (i.e., its incoming and outgoing links) as d_j. Due to our assumption of equal probability, the probability that traffic arriving at H_1 does so via X_j can be given as $1/(d_j - 1)$. Taking the entire path into account, we can give the overall probability that some traffic from H_1 passes X_j as the product $\Pr[X_j] = \prod_{k=1}^{j} 1/(d_k - 1)$.

Interestingly, this assumption works in favour of our model as our estimate is lower than what might be expected realistically: in real settings, certain neighbours are never possible due to routing behaviour like hot-potato routing, valley-free routing, or simply certain topological positions. In our figure Figure 9.3, for instance, we place H_4 'close' to X_2 to hint that a direct path between them is more likely than a long route via other hops (which we labelled 'unlikely'). Applied to our computation of the probability, this means that d_j is often smaller than we assume it. Consequently, the probability that H_1's traffic goes via X_j is higher than we denote it in the model.

Probability for correct placement The next step we have to execute is to determine the probability that two hunters are placed in the position they need to be in. Recall our two requirements from above. If we want to accurately determine the position of the attacker to be X_m, there must be one hunter H_1 who experiences the attack because traffic coming from it crosses X_m, but not X_{m+1} (i.e., traffic coming from C and from H_1 merges exactly at X_m). We also need another hunter H_2 who does not experience the attack any more because its traffic crosses the *next* hop towards V from the point of view of M, i.e., the node X_{m-1}. The first requirement dictates that the traffic cannot have come via X_{m+1}. It cannot have come via X_{m-1}, either: this node is the successor of X_m, and thus this would be a routing loop, which we forbid. Our assumption 6 allows us now to derive the probability that the first requirement is met: we can express this as $\Pr[\text{req I}] = (d_m - 2)/(d_m - 1) \cdot \Pr[X_m]$. This is the probability that traffic from H_1 crosses X_m, times the probability that this traffic does *not* cross X_{m+1}. The probability that the second requirement is met can be expressed in the same way, namely as $\Pr[\text{req II}] = (d_{m-1} - 2)/(d_{m-1} - 1) \cdot \Pr[X_{m-1}]$.

We now use assumption 5 and exploit the fact that hunters are distributed uniformly over the network—in other words, the placement is a Bernoulli trial (either a hunter is placed at a certain position or not). This allows us to express the probability that at least one out of n hunters meets the second requirement, namely as $1-(1-\Pr[\text{req II}])^n$. Our goal is now that at least one of the $n-1$ remaining hunters meets the first requirement: this can now be expressed as $1 - (1 - \Pr[\text{req I}])^{n-1}$.

Since both first and second requirement must be fulfilled, we can derive the total probability that the attacker can be located at X_m as the following product: $\Pr[\text{locate}(X_m)] := (1 - (1 - \Pr[\text{req II}])^n) \cdot (1 - (1 - \Pr[\text{req I}])^n)$.

Note that the formula can be extended to express the following: if we are satisfied with less accuracy, e.g., with locating the attacker with an uncertainty of one or even more hops, we can express this as letting the traffic flows of our hunters cross nodes further away or closer to C or V, respectively.

Arbitrary positions for C, V, and M So far, we have always assumed that our path between C and V is of a fixed length, which we call ℓ. Naturally, path lengths on the Internet vary, and we need to reflect the dependency between a non-fixed path length and the possible positions of the attacker on such a path in our model as well. The probability of localisation can be expressed as an aggregated probability, namely summing over all possible path lengths while summing over all possible locations the attacker may then have for this path length. This yields the following formula, which is also our final closed-form model:

$$\Pr[\text{locate}] := \sum_{k=1}^{\textit{max path length}} \left(\Pr[\ell = k] \cdot \sum_{m=1}^{k} \Pr[\text{locate}(X_m)] \right)$$

Topological data for the closed-form model With our closed-form model developed, we now need to determine input data in order to obtain concrete probabilities. The input to our model is two-fold: we need a distribution that gives us ℓ, i.e., path length, and we need a distribution for the node degrees d_j. We thus collected data for both, on the level of Internet routers as well as ASes.

For node degrees on router-level, we used topological data provided by the Rocketfuel project [68]. Unfortunately, the data set does not reveal whether a node was a client or a server in a measurement. We thus computed the average node degree $\bar{d} = 3.98$ and set $d_j = \bar{d}$ in our model. We acknowledge this is imperfect, but with more precise data sets available, it is easy to do our computations again.

For the path lengths (on router-level), we relied on traceroute measurements from our own university network. We chose 30,000 random hosts from the Alexa list of the top 1 million most popular Web hosts [133] and determined the distribution of path lengths. We found a range of 5–28 hops, with a mean value of 15.28 and a median of 15 hops. In order to determine whether these values are representative for other vantage points as well, we downloaded the traceroute data sets from CAIDA [152], originally used in [52]. We determined both mean and average values for UDP traceroutes[2] that contained the complete path. Table 9.3 shows the results—our values fall into

[2]This is called Method 3 in [52]. The files are named `*3meth.[location.warts.gz]`.

Location	Mean	Median
Barcelona, Spain	15.08	15
Daejeon, Korea	16.24	16
Helsinki, Finland	17.72	18
Washington D.C., USA	15.03	15
San Diego, USA	16.27	16
Sydney, Australia	14.23	14

Table 9.3. – Path lengths for the CAIDA data set.

the middle of the range they span. We thus decided to use the distribution we had determined ourselves.

Concerning input data on the AS level we relied on the Route Views archive [131]. We downloaded the data set (MRT-formatted full-table RIBs[3]) for 7 July 2011 (12:00) for the following vantage points: Oregon IX, Equinix Ashburn, ISC/PAIX, KIXP, LINX, DIXIE/WIDE, RouteViews-4, Sydney, and São Paulo. We combined them and determined the average number of neighbour ASes that an AS has. We obtained an average node degree of 3.51. Path lengths were in the range 1–17, with a mean of 3.25 and a median of 3.

Results using the model We computed the localisation probabilities for two different settings: on AS-level and on router-level. For the latter, we computed values for exact localisation and for an uncertainty of one and two hops. Figure 9.4 shows the results. On the level of ASes, we find that a small number of hunters already gives quite satisfying results. About 100 hunters are enough to make the probability of accurate localisation rise to nearly 100 %. Findings at the router-level are less encouraging, at least at first glance. Although 100 hunters are sufficient for a localisation probability of roughly 25 %, this increases only slowly with a (much) higher number of hunters. 1000 hunters increase it to only 40 %, and even 100,000 hunters cannot increase it beyond 60%.

Both results must be viewed in the context of the attackers that Crossbear is designed to work against, however. Considering the weaker attacker who is only in control of a wireless access point (see Figure 9.2a), a successful localisation needs placement of hunters in the exact same ISP network anyway—it is only to be expected that a large number of hunters is required for this purpose. Considering the regional attacker (see Figure 9.2b), localisation on the level of the AS is sufficient evidence. Thus, the localisation probabilities on this level show that Crossbear works well against our attackers.

The challenge of selective attackers

Selective attackers can neither be localised directly nor on-the-fly. Indeed, the possibility of selective attackers requires that every reported attack is carefully analysed manually.

Consider Figures 9.2(c) and 9.2(d): no hunter, not even downstream, experiences the attack. As far as tracerouting is concerned, these attackers become indistinguishable from the one in Figure 9.2(a). A major challenge thus lies in telling them apart. The approach to take here is to look for clues that the attacker left behind. In particular,

[3]Routing Information Base, i.e., BGP dumps.

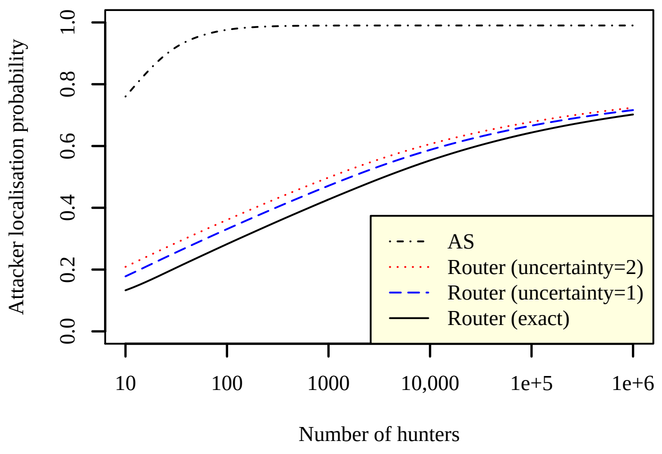


Figure 9.4. – Estimating the number of hunters required to pinpoint an attacker. Note the logarithmic x-axis.

raising the out-of-band information described in Section 9.2.3 may help find evidence that reveals the nature of the attack.

Let us assume a selective attacker, and let us assume that we are in possession of traceroutes from hunters that are affected by the man-in-the-middle. Ideally, we have traceroutes from seemingly non-affected hunters in the same AS, and ASes in the same country, and ASes that are attached to an AS that is further upstream towards the victim server. One mechanism that comes to our aid here is that the attacker cannot easily forge additional traceroutes and send us rogue traceroutes with forged source addresses. The hunting reports require him to carry out full TCP handshakes with the Crossbear server[4], i.e., intercept replies to the IP address he is spoofing. Thus, he can only choose his source IP from the system he controls or one that is attached to it further downstream. Also recall that a traceroute can be tested for plausibility, albeit only to some degree, with available BGP data (e.g., [131]). The hints we are looking for now are poisoned routes from different stub ASes, i.e., ASes on the network periphery. If we find such routes in our data, we can at least conjecture that the man-in-the-middle is located either where traffic streams from these ASes converge (the earliest possible location) or further upstream.

One plausible alternative would be simultaneous attacks against multiple ASes. This is expensive, but possible. One way to tell the two cases apart is to investigate if the forged certificates share properties (like issuer, key lengths, X.509v3 extensions). If they do, this points to a common rule set for creation, and hence two separate man-in-the-middle attacks are less likely.

The next step to execute is now to look up the ASes and countries of all hops in the traceroutes. A hint that a selective state-level attacker is indeed at work is then if we find that the source IPs in the traceroutes belong to an AS or country which we associate with radical monitoring of their own population. If the earliest possible location is already in that country, that is another hint.

If we do not find anything of the kind, however, our chances become slimmer. One pattern that is still worthwhile to look for is the one that the selective super-attacker in the core of the network (Figure 9.2(d)) should show. If the purpose of the attacker is industrial espionage, one may expect that the man-in-the-middle reports and traceroutes are primarily from organisations within a select few countries.

Naturally, all of the above is a mere test of plausibility, and we acknowledge that the proposed methods require (comparatively) intensive manual labour. However, we

[4]The public IP notifications have the same effect before hunting even starts.

wish to point out that until now the research community has practically no data at all about man-in-the-middle attacks occuring in the wild. Any such report providing hard data will advance current research. The man-in-the-middle attack in [183], for example, became known thanks to external reports and because someone made the effort to try and inform the outside world. Receiving automated reports is thus useful even where automatic localisation is not possible. This is why we advertise Crossbear as a tool to record as much data as possible about attacks, but not as a silver bullet in exposing attackers.

9.3.4. Attacks against Crossbear

We analyse to which degree Crossbear is vulnerable to attacks itself. Due to Crossbear's open nature, there are several options for particularly aggressive attackers. Many of these cannot be entirely avoided and have to be dealt with in a reactive way.

Denial-of-service attacks against the Crossbear server

The Crossbear server is, naturally, a single point of failure. There are a variety of denial-of-service attacks that can be staged against it. On the level of TCP/IP, the usual (pro-active and reactive) defences must be taken, e.g., firewalls, remote shell access to the server must be limited to trusted hosts, etc.

The Crossbear server must accept TLS-secured connections (HTTPS) for certificate verification requests. TLS shows some vulnerability to denial-of-service attacks as the TLS handshake requires more resources on server-side than on client-side. This is a vulnerability that cannot be well mediated (and is, in fact, shared by all TLS-secured hosts). One defence is to use TLS acceleration in hardware, which offloads the cryptographic operations to dedicated hardware. As an academic project, Crossbear does not (yet) employ such hardware.

Crossbear uses the Tomcat Java application container. Attacks are thus possible against weaknesses of Tomcat (which is itself written in Java). Both Tomcat and Java have so far a good security record in this regard. Continuous updates must be applied and libraries kept up to date, however.

A denial-of-service attack as described above would only render the Crossbear service unusable for the duration of the attack. As Crossbear's primary role is that of a data gathering tool, not a PKI reinforcement, we consider such attacks to be of only minor criticality for Internet security. After all, a denial-of-service attack on Crossbear would indicate that Crossbear is viewed as a threat by some parties—and it might actually leave forensic evidence behind that could be analysed later.

False reports and malicious hunters

Two other kinds of attacks have the potential to be much more damaging to Crossbear.

One way to cause high load on the Crossbear server is to flood it with a large number of reports of alleged man-in-the-middle attacks. This would cause Crossbear to initiate a hunting task for every report it has received. In turn, every participating hunter would pull and execute the tasks, and send reports to the server. At current deployment plans, the stand-alone hunters on PlanetLab alone would cause up to 150 reports per hunting task—a considerable amplification.

An attacker could pursue two goals with such an attack: either stage a more sophisticated denial-of-service attack or make real man-in-the-middle attacks hard to find in a mass of reports. Unfortunately, Crossbear's nature as an open reporting system makes it very hard to counteract such attacks. Clients and hunters do not have to register nor do they have IDs. This was a conscious choice to encourage user participation. As is true for all such systems, however, one consequence is that attackers can freely send forged data to the server. There is really only one defence: continuous monitoring of requests and reports, with special regard to the rate at which these are received, and alerts when the rate suddenly changes drastically. In such cases, it is more reasonable to assume an attack than a sudden increase in the number of Crossbear users.

There are a number of reactions the server can take, although none of these have been implemented yet. A fine extension would be to switch to a forensic mode. As long as resources are not nearing exhaustion, the server would still accept requests to verify certificates. Only when the system is approaching a critical state would the server deactivate the listening on TCP port 443. In the meantime, hunting tasks would still be created, but not written to the Hunting Task List for distribution to hunters. Instead, the received requests would be analysed with respect to origin (IP address, AS, and geolocation). Human intervention would be required from this point on to determine if the attack shows a certain pattern that is worth publishing and passing on to interested parties. Recurring entries would be of great interest. It is conceivable that an attacker makes use of a botnet to stage this attack. This would probably make the forensic data much less interesting as the attack would be distributed over a large number of systems. However, it would also require dedicated code implemented in the bot—this would be a highly interesting find in itself.

The above attacks can be further refined if the attacker employs 'malicious hunters'. Here, the attacker first drops the connections of all honest hunters in the system he controls or that are attached to it (note that this may lead to out-of-band reports). Then, his malicious hunters send forged reports stating that the connection via the attacker is fine and no man-in-the-middle is detected. The Crossbear server will thus have received only one report of a possible attack (from the original client victim) and a large number of forged reports. This hides the nature of the attack as it seems that only the original client is affected. The only defence that Crossbear has here is that the attacker's source IP is ascertained. As long as the attacker is not in the core of the network, this will result in a suspicious cluster of reports from the same AS or country.

Ultimately, the conclusion to draw here is that Crossbear is effectively always involved in an arms race. Unfortunately, this arms race favours the attacker: the attacker is (in all likelihood) in control of parts of the network and can introduce new attacks at his discretion. Crossbear can only react. This is a weakness it shares with all tools for observation. There is, by its very nature, very little that an observer can do to prevent an attacker from tampering with the observation data if the attacker has some control over the observed object.

9.4. Status of deployment and cooperation with OONI

The Crossbear server is hosted at Technische Universität München. Crossbear is currently available in version 1.5 in the master branch of the code repository [250]. The current roadmap envisages the release of Crossbear 2.0.

As a result of a technology grant, the Crossbear protocol has also been implemented for the Open Observatory for Network Interference (OONI) [24], a project supported by Tor [269]. Crossbear is now available as a module for OONI. We expect this to increase our user base considerably, in particular as OONI's intended audience is practically identical to ours. Within the same project, we developed a visualisation component that displays traceroutes from several hunting reports and highlights the intersection points, while providing meta information like the AS and country where a certain hop is located.

The original stand-alone hunters were implemented in Java and deployed on the PlanetLab test bed, peaking at 150 instances at one time. At the time of writing, they have been deactivated in preparation for a new Python-based stand-alone hunter. The experience with Java was that automated deployment is too difficult in a changing environment like PlanetLab due to the need for a modern run-time environment. The new Python-based hunters are a spin-off of the implementation of Crossbear for OONI. They will resolve the problems the Java-based hunters had.

There are several features that we plan to add to Crossbear. Among these, TCP-based tracerouting has a high priority as it gives us another, and possibly more conclusive, way to detect intermediate hosts and networks on the routing paths. Another option we investigate is whether we can route certificate verification requests and hunting reports via the Tor anonymisation network [269]—not to increase privacy, but due to its good properties in bypassing network blockades.

A further extension is to have the server sign its messages so clients can verify them even over a poisoned connection. This feature has been added in the meantime and found to be stable enough for the next major release of Crossbear, version 2.0.

At the time of writing, our database contains about 4000 certificate observations conducted by our server plus another 2000 retrieved from Convergence notaries. Results have been reported from more than 150 unique sources. We have not found indications of man-in-the-middle attacks so far, however.

9.5. Related work

Crossbear is a part of the movement that started after the DigiNotar incident showed how disastrous the compromise of a single CA can be.

Concerning its notary concept, Crossbear's closest relatives are, naturally, Perspectives [77] and Convergence [268]. A primary difference to Perspectives is that Crossbear does not employ a data redundancy protocol (i.e., cross-validation). It features both the spatial and temporal redundancy, however, as the Crossbear server uses Convergence notaries and keeps track of observations. Crossbear also initiates verification on demand. This is one of the changes that distinguishes Convergence from Perspectives, too. Just like Perspectives, Crossbear can include data from active scans of the IPv4 space for reference, however. Such data is available from our own empirical measurements (see Chapter 4). In contrast to both Convergence and Perspectives, Crossbear is designed to raise additional information about the hosts involved in the TLS handshake and on the IP path.

All notary concepts share the problem of lack of privacy: notary operators know which sites users access. Convergence employs a simplified onion-routing to mediate this. The original paper on Perspectives proposes a DNS-based method to limit the impact on privacy; this has not yet been implemented. Crossbear does not address privacy issues: as its purpose is to collect and report data about real attacks, privacy was a subordinate design goal. However, we do follow developments in this field and introducing privacy-sensitive modes remains an option.

Concerning reporting of attacks, Crossbear's closest relative is the Open Observatory for Network Interference (OONI) [24]. In contrast to Crossbear, OONI is a more comprehensive framework to report all kinds of interference with network connections, e.g., HTML headers or TCP properties. It does not feature a central coordination, however, i.e., OONI clients do not act on-demand and cannot be coordinated to achieve a common goal (like Crossbear's hunting).

9.6. Discussion

We conclude this chapter with a discussion of Crossbear's possibilities.

Crossbear is a tool that can be employed to detect and locate man-in-the-middle attacks on TLS, and we have analysed against which attacker types it is particularly effective. The hypothesis that Crossbear was built to verify or falsify was that two kinds of attackers are the prevalent ones on the Internet: a relatively limited attacker close to the victim client, and a very powerful attacker that has possibilities normally associated with a country.

We found that Crossbear's effectivity in localising the attacker's position in the network depends strongly on the behaviour of the attacker it faces. Best results can be expected against an attacker who stages a non-selective man-in-the-middle attack. Selective attackers cannot be accurately located. The recent development of Crossbear has thus focused on enriching data from hunters with information about traversed ASes, geographical location of routers, and keeping track of certain certification information. Any reported attack will always require manual analysis; and we consequently developed tools to facilitate this process.

It is not implausible at all that nations today are willing to deploy large-scale surveillance techniques. These can be used for good or for bad. Furthermore, while it was known that attacks against BGP can be used to stage man-in-the-middle attacks, the publication of the Internet Census 2012 [136] showed that a devastatingly high number of devices, even routers, lack any proper access control and can be easily compromised. The question is how Crossbear can help in such a scenario. The answer remains the same: much depends on the position of the attacker in the network and his activity. The less selective an attacker acts, the better are Crossbear's chances to locate the affected router or AS. We emphasise that even where accurate localisation fails, Crossbear will still provide evidence that an attack has occurred. This is a worthwhile objective in itself.

We also analysed active measures that an attacker can take against Crossbear. Like all open systems, Crossbear shows a vulnerability here. However, such counter attacks leave hints, too, and this may again be useful evidence. So far, we have not registered any malicious activity against our infrastructure.

We advertise Crossbear as a tool to make a step forward in the reporting and also in the localisation of man-in-the-middle attacks in the wild. We expressively do not market it as a silver bullet to expose all kinds of attackers.

9.7. Key contributions of this chapter

This chapter addressed Research Objective O3.2. We designed and developed Crossbear, a tool that is able to detect ongoing man-in-the-middle attacks and additionally locate the position of the attacker. In the following, we list the key contributions of this chapter:

Notary concept We use the notary concept as made popular by Perspectives and later Convergence (see Section 8.4). The rationale is that only the notary concept allows to detect and report attacks, using different positions on the Internet. We rely on the existing infrastructure of Convergence in addition to our own hunters to obtain a distributed view of a potential attack.

User base We explicitly advertise Crossbear as a tool for those users that wish to make a contribution towards a safer Internet *and* have the expertise to assess Crossbear reports. While Crossbear provides a function to warn users of a potentially ongoing attack, the reported ratings must be interpreted by a human user.

Detection: automated reports Crossbear is the first tool to incorporate automatic detection and reporting of man-in-the-middle attacks. To this end, we implemented it as a Mozilla Firefox add-on plus a server component. A report is triggered in a very conservative fashion with the Crossbear server as an authority: if a client reports an as-yet unknown certificate and the Crossbear server cannot corroborate the report, it will create a hunting task. It also sends a warning for the (experienced!) user, accompanied with information that aids in distinguishing a true attack from a false positive.

Hunting and localisation When confronted with conflicting reports about a certificate, the Crossbear server will initiate hunting tasks. These are downloaded from the server and executed by hunters. Hunters are implemented as part of the Firefox add-on and also in stand-alone versions. They report a server certificate from their point of view and additionally send a traceroute. The traceroutes can be evaluated to determine the position of the attacker.

Accuracy against different attackers Crossbear works best against the non-selective attacker who tries to stage a man-in-the-middle attack against all clients whose network paths he can control. The attackers we have in mind are subclasses of the attackers we described in our threat models in Chapter 8, in particular a weak and local attacker and a stronger, regional attacker. We evaluated the effectivity of Crossbear against these attackers and found a relatively low number of hunters is sufficient to trace the attacker to a specific AS.

Selective attackers We also evaluated the possibility of selective attackers. These cannot be located reliably with a notary concept as they affect just a single client. The possibility of these attackers makes it necessary to inspect reports from hunters manually.

Countermeasures We also described a variety of countermeasures that an attacker could use against Crossbear. These are generally applicable to any open system. Our position is that while attackers can carry out such countermeasures, it is likely that this still leaves traces of the attack that can be valuable for analysis. Furthermore, Crossbear raises the barriers for attackers.

Complementing other approaches We advertise Crossbear as a useful complement for approaches to reinforce the X.509 PKI. We integrated it with the Open Observatory for Network Interferences [24].

9.8. Statement on author's contributions

This chapter is an extended version of the following paper: R. Holz, T. Riedmaier, N. Kammenhuber, G. Carle. X.509 Forensics: detecting and localising the SSL/TLS Men-in-the-middle, *Proc. 17th European Symposium on Research in Computer Security (ESORICS)*, Pisa, Italy, September 2012 (reference [38]).

The above publication is based on, and a continuation of, the work that Thomas Riedmaier carried out in his Master's Thesis: T. Riedmaier. Distributed detection and localization of TLS men-in-the-middle. Master's thesis. Technische Universität München, Fakultät für Informatik, March 2012 (reference [64]). Under the guidance of the author of this thesis, Thomas Riedmaier developed the Crossbear protocols and the scoring mechanism. He implemented the initial versions of both the Firefox add-on and the Crossbear server. Later, Vedat Levi Alev and Jan Seeger implemented the OONI module for Crossbear, advised by the author. They also developed the out-of-band information collection carried out by the Crossbear server. The author made some contributions to the Firefox add-on and the OONI implementation.

The author contributed to the results in the publication and the chapter in the following way. The first draft outlining Crossbear's principle and operation was developed by the author. The author later contributed to the design decisions made during Thomas Riedmaier's Master's Thesis. He also added the out-of-band information the Crossbear server collects and redesigned the scoring algorithm for certificates. The author provided the analysis of Crossbear's effectivity in the context of the threat models (detection and localisation, non-selective and selective attacker) and also provided the analysis of traceroutes from the CAIDA data set. The author made significant contributions to the analysis of attacks against Crossbear. The definition of Crossbear in the notation developed in Chapter 7 was added by the author and did not appear in the publication.

The author also wrote the paper [38]. The exception is the closed-form model and its evaluation (the author provided the analysis of the CAIDA data sets later; it is included only in the thesis, not in the paper).

The following sections are adapted from the respective sections in the paper. Section 9.1 corresponds to the introduction in the paper, rewritten to serve as a motivation. For Section 9.2.1 the author added a significant amount of details on Crossbear's principles and embedded it in the threat model. For Section 9.2.2, the author added clarifications on the expected user base. Section 9.2.3 is a revised and restructured version of the corresponding text in the paper. The author made changes to embed it into the chapter and added details about out-of-band information sources. For Section 9.2.4, the author added the description by means of the notation developed in Chapter 7. The computation of the score was changed to reflect later design decisions. In Section 9.3.1, the author made changes to adapt it to the refined threat model. Section 9.3.2 is a shortened version from the paper. For Section 9.3.3, the author rewrote the derivation of the closed-form model entirely. He added new results from an evaluation of a data set from CAIDA and added some clarifications on selective attackers. Section 9.3.4 is mostly from the paper, but has been extended with the proposal of a forensic mode. The section on deployment, Section 9.4, has been extended by the author to contain the description of OONI. Section 9.5 is based on the related work from the paper, but with a more detailed discussion of Perspectives and Convergence as well as a new section on OONI. Finally, for Section 9.6, the author extended the discussion in the light of recent developments in 2013.

Part IV.

Summary and conclusion

Chapter 10. Summary and conclusion

In this chapter, we first summarise the key findings of this thesis. We then provide a number of conclusions concerning future research on PKIs, based on our findings.

10.1. Results from Research Objectives

This dissertation had three larger Research Objectives, which were in turn split into several smaller ones. Figure 1.1 in Chapter 1 provides a graphical reference.

Research Objective O1

Research Objective O1 addressed problematic issues in the X.509 PKI. The purpose of Research Objective O1 was:

O1.1 To identify weaknesses in X.509 that had been criticised previously and determine whether these were satisfactorily addressed. The approach here was documental, with empirical elements.

O1.2 To investigate known incidents in X.509 and identify their root causes. The method we chose here was a documental analysis.

O1.3 To derive conclusions what kind of reinforcements to the X.509 PKI are needed in order to strengthen it.

Research Objective O1.1

For Research Objective O1.1, we found that the documented weaknesses have mostly not been addressed. A primary criticism was that the existence of too many CAs, subordinate CAs and RAs multiplies the attack surface, and the weakest such entity determines the strength of the entire PKI. To gain a better picture how many entities with signing capacity exist, we analysed the root store of the Mozilla Firefox browser. We found that the number of root certificates has been increasing since late 2000 and continues to increase. It is now well over 130. We also verified how many organisations are owners of such certificates. We found 65 organisations with root certificates in the root store, and another 40 were applying for inclusion. A second documented weakness concerned the liability of CAs. Previous research had documented that CAs did not assume any liability for certificates they issued. We analysed the Baseline Requirements by the CA/Browser forum and found that this is still the case (they are not required to offer any liability). CAs offer very limited liability for Extended Validation (EV) certificates, however. The final criticism we analysed concerned the technical difficulties for CAs to verify an identity, in particular over insecure communication paths. We found that CAs are still allowed to rely on email for identity verification. Email is also allowed in the case of EV certificates as a communication medium with, e. g., legal entities that can confirm an identity. However, the checks that CAs need to apply for EV certificates are much more thorough.

Research Objective O1.2

For Research Objective O1.2, we analysed twelve incident reports and technical reports about compromises in the last twelve years, with particular respect to root causes and how the incidents were detected. Our analysis of the documents showed that three root causes were most commonly responsible. The first was failure to adhere to sound operational practices, which was a root cause in seven of twelve cases. Some CAs had been tricked into issuing certificates for the wrong entity, either because they did not execute checks at all (Comodo 2008) or because their checks were poorly designed (Thawte 2008, RapidSSL 2010). The incidents were often detected by third parties (often by the ones that exploited it successfully and then disclosed it). Technical vulnerabilities were the cause for some graver incidents. In the case of Comodo of 2011, an attacker had managed to compromise an RA, thus acquiring login credentials to Comodo's signing system. In the case of DigiNotar in 2011, the attacker had breached the CA to such a degree that he could issue a large number of rogue certificates. Allegedly, this attack was connected to a man-in-the-middle attack. The incidents were either detected by the CA (e. g., Comodo) or by an outside party (e. g., DigiNotar). In four of twelve cases, the causes were coupled to a third root cause: CAs operating subordinate CAs and RAs. The attacker did not target the main CA, but the subordinate entities. This shows that proliferation of entities with signing capacity can be very dangerous in hierarchical PKIs like X.509.

Research Objective O1.3

Based on our findings, our conclusion was that there is no possibility to fix the problems of the X.509 PKI in a *direct way*, i. e., with changes to X.509 itself. Operational practices can be violated by a CA, and options for technical control are limited. Vulnerabilities in software are likely to always exist—at least, we do not expect a major change in this area in the short or medium term. Although efforts are underway to add technical constraints for subordinate CAs and RAs, the number of entities owning root certificates will continue to remain high. Consequently, we derived three *indirect* mechanisms that can help reinforce X.509:

Out-of-band solutions Security for users can be improved by using out-of-band mechanisms, i. e., technology that does not rely on X.509 itself.

Incident detection It seems infeasible to prevent all attacks. Thus, good defences should also aim at fast incident detection and containment.

Monitoring of the deployed PKI Operational practices may be hard to enforce, but poor practices show up in the PKI as it is encountered by clients. Monitoring can help exercise economic pressure on CAs and server operators to execute their duties with due care.

Research Objective O2

Research Objective O1 led to the insight that monitoring the state of the X.509 PKI can be a sensible way to determine the quality of certification practices as they are reflected in the state of the deployed X.509 PKI. In Research Objective O2, we took one step back and investigated the state of three different PKIs, not just X.509. Each of the selected PKIs serves a different purpose: X.509 is used primarily for the WWW (HTTPS); OpenPGP is commonly used to secure email communication between end-users; SSH—as an example of a PKI without Trusted Third Parties (TTPs)—is meant primarily for network management. Our Research Objectives here were thus:

O2.1 To investigate the deployment and use of the X.509 PKI for the WWW, with a focus on the quality of certification and possible weaknesses.

O2.2 To investigate the OpenPGP Web of Trust, with a focus on the usefulness for users and the security the Web of Trust provides.

O2.3 To investigate the deployment of the SSH infrastructure on the Internet. A particular focus was to be on problems that may be caused by key distribution (i.e., network management).

In our investigations in Research Objective O2, we analysed the quality of each PKI with respect to whether it can achieve its purpose and provide a certain level of security. Necessarily, the methodology had to be different for each PKI. The X.509 PKI can be investigated by active scans and passive monitoring. OpenPGP, on the other hand, is not accessible to active measurement—but one can download snapshots of the certification graph and apply graph analysis to determine whether the PKI can serve its purpose. SSH, finally, is accessible to active measurement again, but conclusions with respect to network management require to enrich the data set with data from other sources, like DNS, WHOIS, and geolocation data.

Research Objective O2.1

We first analysed the X.509 PKI for HTTPS. We used two different methods. First, we carried out long-term scans (1.5 years) of the Alexa Top 1 Million list of popular Web sites. This allowed us to draw conclusions with respect to the configuration of Web hosts and the quality of their certificates. We enhanced this view with additional scans from other vantage points around the globe. For comparison, we added a data set from a third party that was obtained with a different scanning method. Second, we carried out passive monitoring of TLS connections in the Munich Scientific Network. Passive monitoring allowed us to draw conclusions with respect to the certificate problems that users really encounter (as opposed to how they occur as a result of deployment on servers which might be accessed irregularly by users).

Validity of certificates The finding that gave reason for most concern was the certification of hostnames: only 60% of certificates had verifiable chains. The majority was issued for a hostname that was different from the name of the domain on which the certificate was used. On the whole, only about 18% of certificates carried the correct hostname. Another problem were expired certificates, which accounted for almost 20% of the certificates we found. Interestingly, the results from our monitoring data sets indicate that at least the certificate chains in the second monitoring run were more often correct (about 80%). This may be a result of users visiting popular Web sites which had switched to TLS by default.

Self-signed certificates One might wonder if self-signed certificates, which may be used in Trust-On-First-Use scenarios, are issued with greater care. However, we found that this was not so. Only 1% were issued for the correct hostname.

Further problematic issues We found further problematic issues. One was the reuse of certificates on many domains, potentially across physical machines, which opens attack vectors. Another were that some certificates were issued directly from a root certificate, which hints at insecure practices at the CA. Where intermediate certificates were used, the chain lengths were no real reason for concern.

Correlation to rank We found correlations between a site's rank on the Alexa list and whether it offered TLS or not. Interestingly, hosts on the lower ranks seemed to often TLS more often than those on the very high ranks. This may be a result of performance optimisations (high-ranking sites) versus use of default configurations (low-ranking sites). Concerning certificates, we found that there is a positive correlation between high-ranking sites and the validity of certificates.

Global vantage points The view from our global vantage points did not show larger differences to the data sets obtained from Germany. It seems that operators of CDNs exercise due care when deploying certificates.

Cryptography Concerning cryptography, our findings were largely unproblematic, at least for the time when we carried out our observations. Lengths of both asymmetric and symmetric keys were generally sufficient and the chosen block ciphers secure. However, several attacks in the past two years point strongly at systematic weaknesses in AES-CBC and RC4, especially in the context of HTTPS, and thus it is wise to repeat our scans and assess these parameters in the light of the new findings.

Temporal developments We were surprised to find that changes in the certification properties or TLS connectivity of sites were very small during our observation period. Both HTTPS configuration as well as deployment of certificates seem to change very slowly. The most devastating incidents in X.509 happened after our observation period had ended, however. It would be interesting to repeat the scans to determine changes.

Research Objective O2.2

We chose the OpenPGP Web of Trust as an example of a user-driven PKI: entities may certify each other arbitrarily, and a trust model allows to determine the authenticity of a key. Certification in this PKI means to sign another entity's name and public key to create a basic certificate. We analysed this PKI with respect to its usefulness, i.e., whether certificate chains allow users to authenticate the public keys of other users. We used graph analysis for this purpose. We were also interested in whether the OpenPGP Web of Trust shows effects of social relations between participants as this may indicate a certain strength of trust relationships between entities.

Limited global usefulness Our first finding was that while the Web of Trust is relatively large at 2.7 million keys, only about 1.1 million signatures were issued between keys. Almost half a million keys were either expired or revoked. But even the remainder of the Web of Trust could not make full use of it: mutual authentication is only possible where certification paths between two users exist in both directions. We determined these so-called Strongly Connected Components (SCCs) and found that there is a single Large Strongly Connected Component (LSCC), to which many other keys connect only in one direction (incoming or outgoing signature). The second largest SCC is already magnitudes smaller, at just about some hundred keys. The Web of Trust can thus be useful for only a fraction of its users.

LSCC structure We focused the remainder of our analysis on the LSCC, which contained about 42,000 keys at the time. Viewing keys as nodes in a graph, we computed their indegree and outdegree as well as clustering within the LSCC. We found clear signs of a Small World effect. However, in contrast to previous work, we could not corroborate a power-law distribution in the LSCC. Rather, the structure of the LSCC is similar to a scale-free network, but with a much larger number of hubs of smaller size, which are well inter-connected. This should make the LSCC very robust against removal of keys.

Usefulness in the LSCC We determined the degree to which users can benefit from the LSCC by determining how many certification paths exist on average between any two keys. The results showed that relatively few hops are needed to reach other keys: several thousand other keys are at most three hops away, and almost the entire LSCC is reachable via five hops or less. However, as the default trust model of the popular implementation GnuPG limits evaluation of signature chains to at most five hops, this also means that about 5000 keys can, on average, not be reached from any (randomly selected) key. Unfortunately, while this was still a good finding for usefulness, we could also determine that a third of the keys had an outdegree of less than three, which is detrimental in GnuPG's trust model: users of such keys cannot make use of the so-called 'marginally trusted' certification paths and are essentially limited to authenticating such keys that they have personally signed.

Robustness We could confirm the robustness of the LSCC when we simulated both the random removal and targeted removal of keys. The LSCC remained remarkably stable—more stable than could be expected for a scale-free network.

Community structure Our analysis of further social aspects, namely the community structure of the Web of Trust, provided less conclusive results. While we could show that communities exist, there is too little information available to trace social relations really well. We found signs that the same top-level domains occur frequently within one community, but could not establish the same finding for second-level domains. Signature creation times hinted at social meetings where keys had been signed.

Cryptography Finally, concerning the security of cryptographic algorithms in use, we found little reason for concern, but must caution that our results must be interpreted again in the light of new attacks. For example, key sizes were sufficient at 1024 bits at the time of our investigation, but NIST already recommends longer key sizes now. It is an interesting question if, and how fast, OpenPGP users will migrate to new key lengths. Historically, the Web of Trust experienced a major shift towards RSA and away from ElGamal once, with high change rates in a short period of time—it would be interesting to see if such changes happen again.

Research Objective O2.3

The third PKI we investigated was the 'False PKI' of SSH, which does not use TTPs. SSH is often used for administrative purposes, with keys distributed to clients in an out-of-band process.

We carried out three Internet-wide scans, distributed over seven months. We used previous work by Heninger *et al.* [35] as a starting point. The authors had investigated occurrences of cryptographically weak and duplicate keys. We carried out similar scans, but analysed our data with a view to determining the overall state of the SSH PKI and with a focus on a special and as of yet not well-investigated phenomenon: the occurrence of duplicate keys that are *not* cryptographically weak, but might be the result of poor network management practices.

Confirmation of earlier results We could reproduce the results of Heninger *et al.* as we also found a number of SSH host keys that were weak due to the so-called co-prime weakness or due to a bug in Debian's version of a cryptographic library. However, the percentages that we could determine were lower than the ones previous work had found. One may take that as an indication that the authors' disclosure process has started to produce results. However, the percentages are so low now that further changes could also be interpreted as measurement artefacts.

Duplicate keys and network management Concerning duplicate keys, we decided to investigate where the ten most common such keys occur. We could show that network management practices are indeed sometimes responsible. For example, a larger German hosting provider confirmed to us that the key we had found was used on their SSH gateway, in an attempt to centralise administration and keep keys stable on the front ends even when the hosting machines were updated. This seems a secure pattern of use. However, we also found keys that we could trace to certain devices with default keys. These occurred in different networks around the world. One key, for example, occurred mostly in a single AS. This is likely more secure than other cases, where we found the same keys in, e.g., the networks of China and Taiwan. We also found an entertainment device that is sold globally and reuses the same key every time. None of these usage patterns is secure, but strictly localised use may be safer: where SSH is used for administrative purposes, it is likely that the responsible administrator is also part of the same network. Finally, we also investigated one case where duplicate keys occurred in the AS of a German provider, but the number of duplicates was always rather low. The provider confirmed two explanations to us. First, hosts may have more than one IP address assigned, but use the same key on every network interface. Second, the provider also confirmed that rarer cases exist where customers do reuse keys across physical machines. The former setup is safe, the latter is not.

Cryptography As before for X.509 for HTTPS and OpenPGP, we found that the choices of cryptographic algorithms and key lengths were not a real reason for concern, but should be monitored in the light of new developments: the attacks on TLS may or may not be applicable to SSH, but algorithms like RC4 have come under considerable pressure and moving away from them is recommended.

Server versions We found that older SSH server versions are predominant. This finding has to be taken with a grain of salt as our scans can only determine the server version as advertised by the server. However, it may well be an indication that updates to servers occur slowly, which is in line with our previous findings for TLS.

Research Objective O3

Research Objective O3 returned to the X.509 PKI again. Several schemes have been proposed to reinforce X.509. The first task in Research Objective O3 was thus to provide an analysis of these schemes. The second part of Research Objective O3 addressed one issue that is not covered by any scheme: automated reporting of incidents and localisation of the attacker. Our tasks were thus as follows.

O3.1 The first task was split into two steps. The schemes are presented in a variety of ways (RFCs, white papers), which are generally verbose descriptions without a common notation. The first step was thus to develop a formalised notation that allows to describe the schemes in a uniform way. Against this background, the second step was to carry out an analysis of the schemes with respect to the security they offer, their robustness against attackers, and potential issues in deployment. As a result, the most promising candidates to reinforce X.509 were to be identified.

O3.2 The second task was to design, develop, and deploy a scheme that is able to detect man-in-the-middle attacks *and* to raise data that allows to determine the location of the attacker with a certain degree of confidence.

Research Objective O3.1

In order to be able to compare different schemes that have been proposed to improve the X.509 ecosystem, we first developed a formalised notation as a common way to express the mechanisms of each scheme. The notation strikes a balance between conciseness and necessary abstraction.

Expressivity and conciseness The notation makes it easier for a reader to determine the key elements, participants, and communication paths in a scheme. It aids in an analysis of what a scheme can achieve. The notation allows several protocols per scheme as well as to define how they interact and which role participants play in each protocol.

Design elements Our notation is based on role scripts in order to allow precise definition of the behaviour of participants and interactions, both within one protocol as well as between different protocols. Decision processes are modelled with predicate logic, For-loops, and If-clauses. The necessary balance between abstraction and precision is achieved by defining abstractions for tokens and processes that are common in X.509 yet are difficult or impossible to define formally (e.g., legal descriptions). Concerning tokens, this allows us to keep the notation concise. Concerning processes, it enables us to treat these abstract processes as black boxes and only operate on the results that they yield.

We analysed five schemes with respect to the contributions they make in reinforcing the X.509 PKI. The first step in doing so was to define threat models for attackers.

Threat models We defined three attackers, based on the incident reports we analysed for Research Objective O1.2. We defined one weaker kind of localised attacker who is either in the vicinity of a victim client or in the vicinity of a victim server. This attacker is supposed to model the notion of a single attacker without the resources of a larger organisation. The two other attackers are much stronger. The second attacker reflects the notion of an attacker who is in control over the ASes of an entire country. The rationale here is to model what, e.g., an authoritarian government might do to conduct surveillance against its own population. The third attacker, finally, reflects the stronger notion of a globally active attacker who is additionally able to compromise the DNSSEC entries of zones outside his legal reach.

With these threat models, we proceeded to analyse the schemes. We used our notation to describe the decision processes, participants, and their interactions.

Most promising candidates Our analysis yielded two promising candidates to reinforce X.509. The first one is Trust Assertions for Certificate Keys (TACK), which is the strongest scheme to prevent attacks. As a pinning concept, TACK's major drawback is that it requires either a secure first contact or secure bootstrapping. In general, the latter does not scale, but there are use cases where it can be employed. TACK's major advantage is that it provides means for key rollover and changes to a server's certification in general. Under the assumption that neither client nor server can be compromised by an attacker, TACK remains secure even against the strongest of our attackers. However, TACK also has a problem concerning deployment: it requires changes to Web servers. The second promising candidate is Certificate Transparency (CT). This concept focuses on making attacks and certificate misissuance detectable and on identifying the responsible CAs. CT provides a system of logs, auditors and monitors. The scheme has the advantage that it requires opt-in only from CAs without requiring changes to servers or clients—at least in one of its operational modes. Its robustness is very good—attackers would have to compromise both a CA as well as many of the logs the CA cooperates with to avoid early detection. Together, TACK and CT provide a very strong reinforcement to the X.509 PKI.

Other schemes Other schemes do not achieve the same degree of security, are less robust against attacks, or are more difficult to deploy. Perspectives, a notary concept, provides a balanced level of security. Given a high number of notaries, and employing a cross-validation concept, it makes attacks either very hard or at least detectable as they must be sustained for a longer time to be successful. A problem of Perspectives are its deployment requirements: it requires many notaries with high availability. The scheme DNS-based Authentication of Named Entities: TLS Anchor (DANE-TLSA), which mandates DNSSEC, provides lower resistance against the regional and global attackers. A further drawback is that it achieves very good security only if clients validate DNSSEC records themselves. However, DANE-TLSA does not require changes to servers and is relatively easy to roll out, provided that the configuration concerning X.509 certificates and DNS records is tightly integrated. Certification Authority Authorization (CAA) is a scheme that also uses DNS to store entries that are relevant for X.509. As it assumes CAs to be secure against attack, it is the weakest of the schemes we analysed. CAA has one advantage: it provides a standardised way to report violations of CA duties.

Need for automated reports One conclusion we had derived as part of Research Objective O1.3 was that mechanisms are needed that allow to detect incidents fast. CT is a step in the right direction. However, there is one property that all schemes that we analysed lacked: none of them describes a way of automated attack detection and reporting.

Research Objective O3.2

We thus designed, implemented and deployed our own tool, Crossbear, for this purpose. Crossbear uses a notary concept as this is the only concept that allows to provide information about an affected victim's position and network path to a certain host on the Internet. Crossbear's primary purpose is not to protect users but to detect attacks and locate the attacker.

Centralised orchestration Crossbear introduces an element of centralisation to the notary concept. While attacks are detected in the same way as in Perspectives (mismatch of certificates), the Crossbear server takes a coordinating role in the response to a possible attack. When a mismatch is detected, it generates a hunting task that is executed by a (hopefully large) number of hunter entities on the Internet. They connect to the alleged victim server and report the encountered certificates to the Crossbear server, together with a traceroute indicating the network path. This allows Crossbear to run analyses to derive information about the location of the attacker.

Savvy users required Crossbear is a tool for professionals and users wishing to help secure the Internet, but not a tool for the common user. This is even more true as use of Crossbear in regions where the attacker is a government might be dangerous.

Effectivity against different attackers We analysed the scenarios where Crossbear achieves its greatest benefit. These are subcases of the attackers in our threat models, in particular the weak attacker and the regional attacker. We derived a closed-form model to analyse Crossbear's effectivity against an attacker acting non-selectively, i. e., targeting all users in the networks he has control over. Crossbear cannot locate the attacker very accurately if he targets single clients. This may initially seem like a drawback, but Crossbear can still provide reports that an attack against a certain server is ongoing.

Countermeasures by attackers We also discussed measures the attacker can take to target Crossbear itself. Crossbear shares the same problem as all open systems: they need to accept information from unreliable sources and are open to denial-of-service attacks. Crossbear forces an attacker to provide at least a valid IP address when reporting fake hunting results; however, the attacker is still free to be really located further 'downstream' from the Crossbear server's point of view. Ultimately, Crossbear and the attacker are always involved in an arm's race. Thus, we do not market Crossbear as a silver bullet. Instead, we view it as a first step: orchestration of countermeasures and automated reports can help raise public awareness and, in the best case, exercise pressure on attackers.

10.2. Quo vadis?—research directions for PKI

In this thesis, we provided a historical analysis of X.509 problems, empirical analyses of the deployment and properties of three different PKIs, an analysis of reinforcements for X.509, as well as our own tool to detect and locate attacks. Yet an 'ultimate' solution for PKI does not seem to be in sight.

Returning to the definition of authentication, and Boyd's theorem in Chapter 2, one observation to make is that PKIs are an attempt to side-step the fundamental problem of not having authenticated keying material between two entities at the time of their first contact. This problem has no real solution. PKIs can defer the problem to another entity, but security risks will always be associated with this. At the same time, we can observe that the need for better security mechanisms is likely to become more urgent as more devices are equipped with Internet connectivity. PKIs are a way to achieve the necessary scalability. So which directions should research into PKIs take? Our assessment is that further research focusing on the following aspects would be very valuable:

Better management of security One of our empirical findings was that server configurations on the Internet show a slow rate of change—sometimes so slow that obvious deficits are not addressed. This is quite understandable if we accept as a working assumption that administrators are reluctant to make changes to systems that already work and that have not (yet) been compromised. This makes it difficult to deploy solutions like TACK, although they provide excellent security properties. Rather than giving up on them, however, a research direction should be how to design systems such that such security mechanisms can be employed without imposing the danger of temporarily unavailable or, worse, inoperable systems on administrators. Methodologies and best practices to upgrade security protocols and server versions should be investigated. If systems like TACK could be introduced more easily, the security benefit for users would be enormous.

Continous monitoring In our empirical analyses, we found that a good part of the hosts on the WWW seem to have certificates of rather poor quality, with common problems being wrong hostnames or expired certificates. Regular monitoring can at least detect these problems and possibly create some pressure on the responsible entities. Monitoring can also help provide systems like TACK with the necessary information to distribute bootstrapping information.

Cross-validation Two of the schemes that provided good resistance even against strong attackers employed a method that we call cross-validation: Perspectives and CT. In the light of any TTP ultimately being fallible, we view this scheme as extremely useful: it effectively multiplies the effort an attacker must make to be successful. Cross-validation could be employed in other security ecosystems as well.

Early attack detection Finally, we take the view that containment of attacks will remain a topic that any good security concept needs to address. More research should be invested on how attacks in the network can be detected early, and how reports from different vantage points can be combined to yield a more precise picture. Our own tool, Crossbear, is just the starting point here.

Part V.

Appendices

List of frequently used acronyms

AS	Autonomous System
BGP	Border Gateway Protocol
CA	Certification Authority
CAA	Certification Authority Authorization
CDN	Content Distribution Network
CN	Common Name
CPS	Certification Practices Statement
CRL	Certificate Revocation List
CT	Certificate Transparency
DANE-TLSA	DNS-based Authentication of Named Entities: TLS Anchor
DNS	Domain Name System
DNSSEC	Domain Name System Security Extensions
EFF	Electronic Frontier Foundation
EV	Extended Validation
IPSec	Internet Protocol Security
KDC	Key Distribution Centre
LSCC	Largest Strongly Connected Component
MTH	Merkle Tree Hash
OCSP	Online Certificate Status Protocol
OID	Object Identifier
PFS	Perfect Forward Secrecy
PKI	Public Key Infrastructure
RA	Registration Authority
SAN	Subject Alternative Name
SCC	Strongly Connected Component

SCT	Signed Certificate Timestamp
SK	Sovereign Keys
SKS	Synchronizing Keyservers
SNI	Server Name Indication
SSH	Secure Shell
SSL	Secure Sockets Layer
STH	Signed Tree Hash
TACK	Trust Assertions for Certificate Keys
TLS	Transport Layer Security
TLSA	TLS Anchor
TSK	TACK Signing Key
TTP	Trusted Third Party

Academic resources

[1] D. Akhawe, B. Amann, M. Vallentin, and R. Sommer. Here's my cert, so trust me, maybe? Understanding TLS errors on the Web. In *Proc. Int. World Wide Web Conference (WWW)*, Rio de Janeiro, Brazil, May 2013.

[2] R. Albert, H. Jeong, and A.-L. Barabási. Error and attack tolerance of complex networks. *Nature*, 406:378–382, July 2000.

[3] M. R. Albrecht, K. G. Paterson, and G. J. Watson. Plaintext recovery attacks against SSH. In *Proc. 30th IEEE Symposium on Security and Privacy*, Oakland, CA, USA, August 2009.

[4] N. AlFardan, D. J. Bernstein, K. G. Paterson, and J. C. N. Schuldt. On the security of RC4 in TLS. In *Proc. 22nd USENIX Security Symposium*, Washington, D.C., USA, August 2013.

[5] M. Allman and V. Paxson. Issues and etiquette concerning use of shared measurement data. In *Proc. 7th ACM SIGCOMM Internet Measurement Conference (IMC)*, San Diego, CA, USA, October 2007.

[6] B. Amann, R. Sommer, M. Vallentin, and S. Hall. No attack necessary: the surprising dynamics of SSL trust relationships. In *Proc. 2013 Ann. Computer Security Applications Conference (ACSAC)*, New Orleans, LA, USA, December 2013.

[7] H. Asghari, M. J. G. van Eeten, A. M. Arnbak, and N. A. N. M. van Eijk. Security economics in the HTTPS value chain. In *Proc. 12th Ann. Workshop on the Economics of Information Security (WEIS)*, Washington, D.C., USA, March 2013.

[8] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, October 2008.

[9] M. Boguñá, R. Pastor-Satorras, A. Díaz-Guilera, and A. Arenas. Models of social networks based on social distance attachment. *Phys. Rev. E*, 70:056122, November 2004.

[10] C. Boyd. Security architecture using formal methods. *IEEE Journal on Selected Areas in Communications*, 11(5):694–701, June 1993.

[11] L. Braun, A. Didebulidze, N. Kammenhuber, and G. Carle. Comparing and improving current packet capturing solutions based on commodity hardware. In *Proc. 10th ACM SIGCOMM Internet Measurement Conference (IMC)*, Melbourne, Australia, November 2010.

[12] L. Braun, G. Münz, and G. Carle. Packet sampling for worm and botnet detection in TCP connections. In *Proc. 12th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Osaka, Japan, April 2010.

[13] R. Canetti. Universally composable security: a new paradigm for cryptographic protocols. Cryptology ePrint Archive: Report 2000/067. `http://eprint.iacr.org/2000/067`.

[14] A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Phys. Rev. E*, 70:066111, December 2004.

[15] A. Clauset, C. R. Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Rev.*, 51(4):661–703, 2009.

[16] C. J. F. Cremers, S. Mauw, and E. de Vink. Defining authentication in a trace model. In *Proc. 1st Int. Workshop on Formal Aspects in Security and Trust (FAST)*, Pisa, Italy, September 2003.

[17] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[18] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, March 1983.

[19] H. Dreger, A. Feldmann, M. Mai, V. Paxson, and R. Sommer. Dynamic application-layer protocol analysis for network intrusion detection. In *Proc. USENIX Security Symposium*, Vancouver, Canada, July 2006.

[20] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman. Analysis of the HTTPS certificate ecosystem. In *Proc. 13th ACM SIGCOMM Internet Measurement Conference (IMC)*, Barcelona, Spain, October 2013.

[21] Z. Durumeric, E. Wustrow, and J. A. Halderman. ZMap: Fast internet-wide scanning and its security applications. In *Proc. 22nd USENIX Security Symposium*, Washington, D.C., USA, August 2013.

[22] C. Ellison. The nature of a usable PKI. *Int. Journal of Computer and Telecommunications Networking—Special Issue on Computer Network Security*, 31(9):823–830, April 1999.

[23] C. Ellison and B. Schneier. Ten risks of PKI: what you're not being told about Public Key Infrastructure. *Computer Security Journal*, 16(1):1–7, 2000.

[24] A. Filastò and J. Appelbaum. OONI: Open Observatory of Network Interference. In *Proc. 2nd USENIX Workshop on Free and Open Communications on the Internet (FOCI)*, Bellevue, WA, USA, August 2012.

[25] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, February 2010.

[26] F. Fusco and L. Deri. High speed network traffic analysis with commodity multicore systems. In *Proc. 10th ACM SIGCOMM Conference on Internet Measurement (IMC)*, Melbourne, Australia, November 2010.

[27] O. Gasser. Understanding SSH: large-scale measurements and notary-based authentication. Master's thesis, Technische Universität München, Fakultät für Informatik, Garching b. München, Germany, February 2013.

[28] O. Gasser, R. Holz, and G. Carle. A deeper understanding of SSH: results from Internet-wide scans. In *Proc. 14th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Krakow, Poland, May 2014.

[29] D. Goldschlag, M. Reed, and P. Syverson. Onion routing. *Communications of the ACM*, 42(2):39–41, February 1999.

[30] A. C. Grant. Search for trust: an analysis and comparison of CA system alternatives and enhancements. Technical Report TR2012-716, Dartmouth College, Department of Computer Science, Hanover, NH, USA, June 2012.

[31] S. Gregory. Finding overlapping communities in networks by label propagation. *New Journal of Physics*, 12(10):103018, October 2010.

[32] P. Gutmann. PKI: it's not dead, just resting. *IEEE Computer*, 35(8):41–49, August 2002.

[33] P. Gutmann. PKI design for the real world. In *Proc. 2006 Workshop on New Security Paradigms (NSPW)*, September 2006.

[34] J. Heidemann, Y. Pradkin, R. Govindan, C. Papadopoulos, G. Bartlett, and J. Bannister. Census and survey of the visible Internet. In *Proc. 8th ACM SIGCOMM Internet Measurement Conference (IMC)*, Vouliagmeni, Greece, October 2008.

[35] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman. Mining your Ps and Qs: detection of widespread weak keys in network devices. In *Proc. 21st USENIX Security Symposium*, Bellevue, WA, USA, August 2012.

[36] C. Herley. So long, and no thanks for the externalities: the rational rejection of security advice by users. In *Proc. 2009 Workshop on New Security Paradigms (NSPW)*, Oxford, UK, September 2009.

[37] R. Holz, L. Braun, N. Kammenhuber, and G. Carle. The SSL landscape—a thorough analysis of the X.509 PKI using active and passive measurements. In *Proc. 11th ACM SIGCOMM Internet Measurement Conference (IMC)*, Berlin, Germany, November 2011.

[38] R. Holz, T. Riedmaier, N. Kammenhuber, and G. Carle. X.509 forensics: detecting and localising the SSL/TLS Men-in-the-middle. In *Proc. 17th European Symposium on Research in Computer Security (ESORICS)*, Pisa, Italy, September 2012.

[39] A. Jøsang. An algebra for assessing trust in certification chains. In *Proc. 1999 Network and Distributed Systems Security Symposium (NDSS)*, San Diego, CA, USA, February 1999.

[40] A. Klein. Attacks on the RC4 stream cipher. *Designs, Codes and Cryptography*, 48(3):269–286, September 2008.

[41] T. Kleinjung, K. Aoki, J. Franke, A. Lenstra, E. Thomé, J. Bos, P. Gaudry, A. Kruppa, P. Montgomery, D. Osvik, H. te Riele, A. Timofeev, and P. Zimmermann. Factorization of a 768-bit RSA modulus. In *Proc. 30th Int. Cryptology Conference (CRYPTO)*. Santa Barbara, CA, USA, August 2010.

[42] L. M. Kohnfelder. Towards a practical public-key cryptosystem. Master's thesis, Massachusetts Institute of Technology, CSAIL, Cambridge, MA, USA, May 1978.

[43] S. Kornexl, V. Paxson, H. Dreger, A. Feldmann, and R. Sommer. Building a time machine for efficient recording and retrieval of high-volume network traffic. In *Proc. 5th ACM SIGCOMM Internet Measurement Conference (IMC)*, Berkeley, CA, USA, October 2005.

[44] A. Lancichinetti and S. Fortunato. Community detection algorithms: a comparative analysis. *Phys. Rev. E*, 80:056117, November 2009.

[45] H. K. Lee, T. Malkin, and E. Nahum. Cryptographic strength of SSL/TLS servers: current and recent practices. In *Proc. 7th ACM SIGCOMM Internet Measurement Conference (IMC)*, San Diego, CA, USA, October 2007.

[46] A. Lenstra, J. Hughes, M. Augier, J. Bos, T. Kleinjung, and C. Wachter. Ron was wrong, Whit is right. Cryptology ePrint Archive, Report 2012/064. `http://eprint.iacr.org/2012/064`, February 2012.

[47] A. Lenstra, X. Wang, and B. de Weger. Colliding X.509 certificates. Cryptology ePrint Archive, Report 2005/067. `http://eprint.iacr.org/2005/067`, March 2005.

[48] D. Leonard and D. Loguinov. Demystifying service discovery: Implementing an Internet-wide scanner. In *Proc. 10th ACM SIGCOMM Internet Measurement Conference (IMC)*, Melbourne, Australia, November 2010.

[49] L. Li, D. Alderson, J. C. Doyle, and W. Willinger. Towards a theory of scale-free graphs: definition, properties, and implications. *Internet Mathematics*, 2(4):431–523, March 2005.

[50] W. Lian, E. Rescorla, H. Shacham, and S. Savage. Measuring the practical impact of DNSSEC deployment. In *Proc. 22nd USENIX Security Symposium*, Washington D.C., USA, August 2013.

[51] G. Lowe. A hierarchy of authentication specifications. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, Rockport, MA, USA, June 1997.

[52] M. Luckie, Y. Hyun, and B. Huffaker. Traceroute probe method and forward IP path inference. In *Proc. 8th ACM SIGCOMM Internet Measurement Conference (IMC)*, Vouliagmeni, Greece, October 2008.

[53] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. l-diversity: privacy beyond k-anonymity. *ACM Trans. Knowledge Discovery from Data*, 1(1), March 2007.

[54] U. Maurer. Modelling a Public-Key Infrastructure. In *Proc. 4th European Symposium on Research in Computer Security (ESORICS)*, Rome, Italy, September 1996.

[55] M. E. J. Newman. Assortative mixing in networks. *Phys. Rev. Lett.*, 89:208701, October 2002.

[56] M. E. J. Newman. The structure and function of complex networks. *SIAM Rev.*, 45(2):167–256, 2003.

[57] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69:026113, February 2004.

[58] M. E. J. Newman and J. Park. Why social networks are different from other types of networks. *Phys. Rev. E*, 68:036122, September 2003.

[59] R. Pastor-Satorras, A. Vázquez, and A. Vespignani. Dynamical and correlation properties of the Internet. *Phys. Rev. Lett.*, 87:258701, November 2001.

[60] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23–24):2435–2463, December 1999.

[61] R. Perlman. An overview of PKI trust models. *IEEE Network*, 13(6):38–43, November/December 1999.

[62] N. Provos and P. Honeyman. ScanSSH—scanning the Internet for SSH servers. In *Proc. 15th USENIX Systems Administration Conference (LISA)*, Baltimore, MD, USA, December 2001.

[63] S. Qiu, P. McDaniel, and F. Monrose. Toward valley-free inter-domain routing. In *Proc. IEEE Int. Conference on Communications (ICC)*, Glasgow, UK, June 2007.

[64] T. Riedmaier. Distributed detection and localization of TLS men-in-the-middle. Master's thesis, Technische Universität München, Fakultät für Informatik, Garching b. München, Germany, March 2012.

[65] S. B. Roosa and S. Schultze. The "Certificate Authority" trust model for SSL: a defective foundation for encrypted Web traffic and a legal quagmire. *Intellectual Property & Technology Law Journal*, 22(11):3–8, November 2010.

[66] M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proc. of the National Academy of Sciences of the United States of America (PNAS)*, 105(4):1118–1123, January 2008.

[67] C. Soghoian and S. Stamm. Certified lies: detecting and defeating government interception attacks against SSL. In *Proc. 15th Int. Conference on Financial Cryptography and Data Security (FC)*, St. Lucia, March 2011.

[68] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. In *Proc. 2002 ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, Pittsburgh, PA, USA, August 2002.

[69] M. Stevens, A. Lenstra, and B. de Weger. Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. In *Proc. 26th Ann. Int. Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt)*, Barcelona, Spain, May 2007.

[70] L. Sweeney. k-anonymity: A model for protecting privacy. *Int. Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, October 2002.

[71] R. Teixeira, A. Shaikh, T. Griffin, and J. Rexford. Dynamics of hot-potato routing in IP networks. In *Proc. Joint Int. Conference on Measurement and Modeling of Computer Systems (SIGMETRICS/PERFORMANCE)*, New York, NY, USA, June 2004.

[72] A. Ulrich. Analyse des OpenPGP Web of Trust. Studienarbeit, Universität Tübingen, Fakultät für Informations- und Kognitionswissenschaften. Tübingen, Germany, June 2010.

[73] A. Ulrich, R. Holz, P. Hauck, and G. Carle. Investigating the OpenPGP Web of Trust. In *Proc. 16th European Symposium on Research in Computer Security (ESORICS)*, Leuven, Belgium, September 2011.

[74] S. Čapkun, L. Buttyán, and J.-P. Hubaux. Small Worlds in security systems: an analysis of the PGP certificate graph. In *Proc. 2002 Workshop on New Security Paradigms (NSPW)*, pages 28–35, Virginia Beach, VA, USA, September 2002.

[75] N. Vratonjic, J. Freudiger, V. Bindschaedler, and J.-P. Hubaux. The inconvenient truth about Web certificates. In *Proc. 10th Workshop on Economics of Information Security (WEIS)*, Fairfax, VA, USA, June 2011.

[76] X. Wang, X. Lai, D. Feng, and H. Yu. Collisions for hash functions MD4, MD5, Haval-128, and RIPEMD. Cryptology ePrint Archive: Report 2004/199. `https://eprint.iacr.org/2004/199.pdf`, August 2004.

[77] D. Wendlandt, D. G. Andersen, and A. Perrig. Perspectives: Improving SSH-style host authentication with multi-path probing. In *Proc. USENIX 2008 Ann. Technical Conference (ATC)*, Boston, MA, USA, June 2008.

[78] Y. Xie, F. Yu, and K. Achan. How dynamic are IP addresses? In *Proc. 2007 ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, Kyoto, Japan, August 2007.

[79] S. Yilek, E. Rescorla, H. Shacham, B. Enright, and S. Savage. When private keys are public—results from the 2008 Debian OpenSSL vulnerability. In *Proc. 9th ACM SIGCOMM Internet Measurement Conference (IMC)*, Chicago, IL, USA, November 2009.

[80] S. Zaker Soltani. Improving PKI: solution analysis in case of CA compromisation. Master's thesis, Universiteit Utrecht, Faculteit Bètawetenschappen, Utrecht, Netherlands, January 2013.

Books

[81] M. Bishop. *Introduction to computer security.* Addison-Wesley, 2005.

[82] C. Boyd and A. Mathuria. *Protocols for authentication and key establishment.* Springer, 2003.

[83] M. Brinkmeier and T. Schank. Network statistics. In U. Brandes and T. Erlebach, editors, *Network analysis: methodological foundations*, volume 3418 of *LNCS*, pages 293–317. Springer, 2004.

[84] A. Croll and S. Power. *Complete Web Monitoring.* O'Reilly Media, 2009.

[85] N. Ferguson, B. Schneier, and T. Kohno. *Cryptography engineering: design principles and practical applications.* Wiley Publishing, 2010.

[86] P. Gutmann. *Engineering security (tentative title).* Work-in-progress, April 2013.

[87] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography.* CRC Press, 1997.

[88] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical recipes in FORTRAN 77: vol. 1 of Fortran numerical recipes: the art of scientific computing.* Cambridge University Press, 1992.

[89] I. Ristić. *Bullet-proof SSL and TLS.* Feisty Duck Ltd. (work-in-progress), July 2013.

RFCs

[90] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. RFC 4034: Resource records for the DNS Security Extensions. Standards Track, Proposed Standard. `http://tools.ietf.org/html/rfc4034`, March 2005.

[91] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. RFC 4035: Protocol modifications for the DNS Security Extensions. Standards Track, Proposed Standard. `http://tools.ietf.org/html/rfc4035`, March 2005.

[92] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. RFC 3546: Transport Layer Security (TLS) Extensions. Standards Track, Proposed Standard. `http://tools.ietf.org/html/rfc3546`, June 2003.

[93] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. RFC 4880: OpenPGP Message Format. Standards Track, Proposed Standard. `http://tools.ietf.org/html/rfc4880`, November 2007.

[94] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. RFC 5280: Internet X.509 Public Key Infrastructure certificate and Certificate Revocation List (CRL) profile. Standards Track, Proposed Standard. `http://tools.ietf.org/html/rfc5280`, May 2008.

[95] T. Dierks and C. Allen. RFC 2246: The TLS protocol version 1.0. Standards Track, Proposed Standard. `http://tools.ietf.org/html/rfc2246`, January 1999.

[96] T. Dierks and E. Rescorla. RFC 4346: The Transport Layer Security (TLS) protocol version 1.1. Standards Track, Proposed Standard. `http://tools.ietf.org/html/rfc4346`, April 2006.

[97] T. Dierks and E. Rescorla. RFC 5246: The Transport Layer Security (TLS) protocol version 1.2. Standards Track, Proposed Standard. `http://tools.ietf.org/html/rfc5246`, August 2008.

[98] D. Eastlake. RFC 2535: Domain Name System Security Extensions. Standards Track, Proposed Standard. `http://tools.ietf.org/html/rfc2535`, March 1999.

[99] D. Eastlake. RFC 6066: Transport Layer Security (TLS) Extensions: Extension definitions. Standards Track, Proposed Standard. `http://tools.ietf.org/html/rfc6066`, January 2011.

[100] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylönen. RFC 2693: SPKI certificate theory. Experimental. `http://tools.ietf.org/html/rfc2693`, September 1999.

[101] C. Evans, C. Palmer, and R. Sleevi. Public key pinning extension for HTTP. Internet-Draft, intended status: Standards Track. `http://tools.ietf.org/html/draft-ietf-websec-key-pinning-08`, July 2013.

[102] J. Galbraith and O. Saarenmaa. SSH File Transfer Protocol. Internet-Draft. `http://tools.ietf.org/html/draft-ietf-secsh-filexfer-13`, July 2006.

[103] P. Gutmann. Key management through key continuity (KCM). Internet-Draft, intended status: BCP. http://tools.ietf.org/html/draft-gutmann-keycont-01, September 2008.

[104] P. Hallam-Baker and R. Stradling. RFC 6844: DNS Certification Authority Authorization (CAA) resource record. Standards Track, Proposed Standard. http://tools.ietf.org/html/rfc6844, January 2013.

[105] P. Hoffman and J. Schlyter. RFC 6698: The DNS-based authentication of named entities (DANE) Transport Layer Security (TLS) protocol: TLSA. Standards Track, Proposed Standard. http://tools.ietf.org/html/rfc6698, August 2012.

[106] K. Igoe and D. Stebila. RFC 6187: X.509v3 certificates for Secure Shell authentication. Standards Track, Proposed Standard. http://tools.ietf.org/html/rfc6187, March 2011.

[107] S. Josefsson. RFC 4398: Storing certificates in the Domain Name System (DNS). Standards Track, Proposed Standard. http://tools.ietf.org/html/rfc4398, March 2006.

[108] S. Kent and K. Seo. RFC 4301: Security architecture for the Internet Protocol. Standards Track, Proposed Standard. http://tools.ietf.org/html/rfc4301, December 2005.

[109] B. Laurie, A. Langley, and E. Kasper. RFC 6962: Certificate Transparency. Experimental. http://tools.ietf.org/html/rfc6962, June 2013.

[110] B. Laurie, G. Sisson, R. Arends, and D. Blacka. RFC 5155: DNS Security (DNSSEC) hashed authenticated denial of existence. Standards Track, Proposed Standard. http://tools.ietf.org/html/rfc5155, March 2008.

[111] S. Lehtinen and C. Lonvick. RFC 4250: the Secure Shell (SSH) protocol assigned numbers. Standards Track, Proposed Standard. http://tools.ietf.org/html/rfc4250, January 2006.

[112] M. Marlinspike and T. Perrin. Trust Assertions for Certificate Keys. Internet-Draft, intended status: Standards Track. http://tools.ietf.org/html/draft-perrin-tls-tack-02, January 2013.

[113] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. RFC 4120: The Kerberos Network Authentication Service (V5). Standards Track, Proposed Standard. http://tools.ietf.org/html/rfc4120, July 2005.

[114] Y. Pettersen. RFC 6961: The Transport Layer Security (TLS) Multiple Certificate Status Request extension. Standards Track, Proposed Standard. http://tools.ietf.org/html/rfc6961, June 2013.

[115] B. Ramsdell and S. Turner. RFC 5750: Secure/Multipurpose Internet Mail Extensions (S/MIME) version 3.2 certificate handling. Standards Track, Proposed Standard. http://tools.ietf.org/html/rfc5750, January 2010.

[116] B. Ramsdell and S. Turner. RFC 5751: Secure/Multipurpose Internet Mail Extensions (S/MIME) version 3.2 message specification. Standards Track, Proposed Standard. http://tools.ietf.org/html/rfc5751, January 2010.

[117] E. Rescorla. RFC 2818: HTTP over TLS. Informational. http://tools.ietf.org/html/rfc2818, May 2000.

[118] S. Santesson. RFC 4262: X.509 Certificate Extension for Secure/Multipurpose Internet Mail Extensions (S/MIME) capabilities. Standards Track, Proposed Standard. http://tools.ietf.org/html/rfc4262, December 2005.

[119] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. RFC 6960: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol—OCSP. Standards Track, Proposed Standard. http://tools.ietf.org/html/rfc6960, Jun 2013.

[120] J. Schlyter and W. Griffin. RFC 4255: Using DNS to securely publish Secure Shell (SSH) key fingerprints. Standards Track, Proposed Standard. http://tools.ietf.org/html/rfc4255, January 2006.

[121] O. Sury. RFC 6594: Use of the SHA-256 algorithm with RSA, Digital Signature Algorithm (DSA), and Elliptic Curve DSA (ECDSA) in SSHFP resource records. Standards Track, Proposed Standard. http://tools.ietf.org/html/rfc6594, April 2012.

[122] D. Thaler and C. Hopps. RFC 2991: Multipath issues in unicast and multicast next-hop selection. Informational. http://tools.ietf.org/html/rfc2991, November 2000.

[123] P. Vixie, O. Gudmundsson, D. Eastlake 3rd, and B. Wellington. RFC 2845: Secret key transaction authentication for DNS (TSIG). Standards Track, Proposed Standard. http://tools.ietf.org/html/rfc2845, May 2000.

[124] T. Ylonen. The SSH (Secure Shell) remote login protocol. Internet-Draft. http://tools.ietf.org/html/draft-ylonen-ssh-protocol-00, November 1995.

[125] T. Ylonen and C. Lonvick. RFC 4251: the Secure Shell (SSH) protocol architecture. Standards Track, Proposed Standard. http://tools.ietf.org/html/rfc4251, January 2006.

[126] T. Ylonen and C. Lonvick. RFC 4252: the Secure Shell (SSH) authentication protocol. Standards Track, Proposed Standard. http://tools.ietf.org/html/rfc4252, January 2006.

[127] T. Ylonen and C. Lonvick. RFC 4253: the Secure Shell (SSH) transport layer protocol. Standards Track, Proposed Standard. http://tools.ietf.org/html/rfc4253, January 2006.

[128] T. Ylonen and C. Lonvick. RFC 4254: the Secure Shell (SSH) connection protocol. Standards Track, Proposed Standard. http://tools.ietf.org/html/rfc4254, January 2006.

[129] K. Zeilenga (editor). RFC 4510: Lightweight Directory Access Protocol (LDAP): technical specification road map. Standards Track, Proposed Standard. http://tools.ietf.org/html/rfc4510, June 2006.

Further resources

[130] H. Adkins. An update on attempted man-in-the-middle attacks. Blog post on Google Online Security Blog. http://googleonlinesecurity.blogspot.de/2011/08/update-on-attempted-man-in-middle.html, 29 August 2011.

[131] Advanced Network Technology Center, University of Oregon. Route Views Project. Project homepage. http://www.routeviews.org/, January 2005.

[132] R. Alden. Web browsers and Comodo announce a successful Certificate Authority attack, perhaps from Iran. Reply to thread with this topic in newsgroup mozilla.dev.security.policy. https://groups.google.com/d/msg/mozilla.dev.security.policy/zgKmHOTIxn8/5NNYcgPNqlgJ, 29 March 2011.

[133] Alexa Internet Inc. Top 1,000,000 sites. Link to online list (updated daily). http://s3.amazonaws.com/alexa-static/top-1m.csv.zip, 2009–2011.

[134] R. Andrews. Re: [cabfpub] upcoming changes to Google Chrome's certificate handling. Post on CA/Browser Forum public mailing list. https://cabforum.org/pipermail/public/2013-November/002336.html, 5 November 2013.

[135] Anonymous. The Spyfiles. Collection of leaked documents on WikiLeaks. http://wikileaks.org/spyfiles/, December 2011.

[136] Anonymous. Internet census 2012. Port scanning /0 using insecure embedded devices. Online paper and data sets. http://internetcensus2012.bitbucket.org/paper.html, 2012.

[137] J. Appelbaum. Detecting Certificate Authority compromises and Web browser collusion. Blog post on Tor Project blog. https://blog.torproject.org/blog/detecting-certificate-authority-compromises-and-web-browser-collusion, 22 March 2011.

[138] H. Asghari. pyasn—Python IP to ASN lookup module. Source code repository. http://code.google.com/p/pyasn/, 2013.

[139] AVISPA Project. Deliverable D2.1: the high level protocol specification language. http://www.avispa-project.org/delivs/2.1/d2-1.pdf, August 2003.

[140] AVISPA Project. Automated validation of Internet security protocols and applications. Project homepage. http://www.avispa-project.org/, 2013.

[141] E. Barker, L. Chen, A. Regenscheid, and M. Smid. Recommendation for pair-wise key establishment schemes using integer factorization cryptography. NIST Special Publication 800-56B. http://csrc.nist.gov/publications/nistpubs/800-56B/sp800-56B.pdf, August 2009.

[142] E. Barker and A. Roginsky. Transitions: recommendation for transitioning the use of cryptographic algorithms and key lengths. NIST Special Publication 800-131A. http://csrc.nist.gov/publications/PubsSPs.html, January 2011.

[143] D. J. Bernstein. Breaking DNSSEC. Keynote at 3rd USENIX Workshop On Offensive Technologies (WOOT). http://cr.yp.to/talks/2009.08.10/slides.pdf, August 2009.

[144] D. J. Bernstein. Failures of secret-key cryptography. Talk at 20th Int. Workshop on Fast Software Encryption (FSE). http://cr.yp.to/talks/2013.03.12/slides.pdf, March 2013.

[145] A. Borhani. Is This MITM Attack to Gmail's SSL? Post in Gmail Help Forum: https://productforums.google.com/d/msg/gmail/3J3r2JqFNTw/oHHZLJeedHMJ, 27 August 2011.

[146] N. Boyard. Bug 470897—investigate incident with CA that allegedly issued bogus cert for www.mozilla.com. Entry in Bugzilla@Mozilla. https://bugzilla.mozilla.org/show_bug.cgi?id=470897, 22 December 2012.

[147] CA/Browser Forum. Guidelines for the issuance and management of Extended Validation certificates (version 1.3). https://cabforum.org/wp-content/uploads/Guidelines_v1_3.pdf, 2010.

[148] CA/Browser Forum. Baseline requirements for the issuance and management of publicly-trusted certificates, v.1.0. https://www.cabforum.org/Baseline_Requirements_V1.pdf, November 2011.

[149] CA/Browser Forum. Guidelines for the issuance and management of Extended Validation certificates (version 1.4). https://cabforum.org/wp-content/uploads/Guidelines_v1_4.pdf, May 2012.

[150] CA/Browser Forum. Homepage. https://www.cabforum.org/, 2013.

[151] CAcert. History of risks & threat events to CAs and PKI. Post in CAcert Wiki. http://wiki.cacert.org/Risk/History, 2013.

[152] CAIDA. The traceroute probe method 2008-08 dataset. http://www.caida.org/data/active/trmethod-200808.xml, 2008.

[153] J. Cederlöf. Web of Trust statistics and pathfinder. Project homepage. http://www.lysator.liu.se/~jc/wotsap/, 2013.

[154] Certificate Transparency. Project homepage. http://www.certificate-transparency.org/, 2013.

[155] Certificate Transparency. Source code repository. https://code.google.com/p/certificate-transparency/, 2013.

[156] D. Chadwick. PKI, past, present and future. Keynote at 1st European PKI Workshop: Research and Applications (EuroPKI). http://www.aegean.gr/europki2004/Keynote.pdf, June 2004.

[157] Chaos Computer Club. 28C3—behind enemy lines. Homepage of 28th Chaos Computer Congress. http://events.ccc.de/congress/2011/wiki/Welcome, December 2011.

[158] M. Coates. Revoking trust in two TurkTrust certificates. Blog post on Mozilla Security Blog. https://blog.mozilla.org/security/2013/01/03/revoking-trust-in-two-turktrust-certficates/, 3 January 2013.

[159] Comodo. Comodo report of incident—comodo detected and thwarted an intrusion on 26-MAR-2011. Post on company Web site. `http://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html`, 23 March 2011.

[160] Comodo Hacker (pseudonym). Another proof of hack from Comodo Hacker. Post on Pastebin. `http://pastebin.com/DBDqm6Km`, 27 March 2011.

[161] Comodo Hacker (pseudonym). Another status update message. Post on Pastebin. `http://pastebin.com/85WV10EL`, 6 September 2011.

[162] Comodo Hacker (pseudonym). Comodo Hacker: Mozilla cert released. Post on Pastebin. `http://pastebin.com/X8znzPWH`, 28 March 2011.

[163] Comodo Hacker (pseudonym). Just another proof from Comodo Hacker. Post on Pastebin. `http://pastebin.com/CvGXyfiJ`, 28 March 2011.

[164] Comodo Hacker (pseudonym). A message from Comodo Hacker. Post on Pastebin. `http://pastebin.com/74KXCaEZ`, 26 March 2011.

[165] Comodo Hacker (pseudonym). Pastebin account. `http://pastebin.com/u/ComodoHacker`, September 2011.

[166] Comodo Hacker (pseudonym). Response to comments from ComodoHacker. Post on Pastebin. `http://pastebin.com/kkPzzGKW`, 29 March 2011.

[167] Comodo Hacker (pseudonym). Response to some comments. Post on Pastebin. `http://pastebin.com/GkKUhu35`, 7 September 2011.

[168] Comodo Hacker (pseudonym). Striking back.... Post on Pastebin. `http://pastebin.com/1AxH30em`, 5 September 2011.

[169] Comodo Hacker (pseudonym). Twitter account. `https://twitter.com/ichsunx2`, September 2011.

[170] Comodo Hacker (pseudonym). Two more little points. Post on Pastebin. `http://pastebin.com/jhz20PqJ`, 6 September 2011.

[171] CPA Canada. WebTrust homepage. `http://www.webtrust.org`, 2013.

[172] C. Cremers. The Scyther tool. Project homepage. `http://www.cs.ox.ac.uk/people/cas.cremers/scyther/index.html`, 2013.

[173] CVE Details. Web site publishing CVEs. `http://www.cvedetails.com`, 2013.

[174] DANE Working Group. DNS-based Authentication of Named Entities (dane). Charter for working group. `http://datatracker.ietf.org/wg/dane/charter/`, 2013.

[175] Debian Project. Debian security advisory: DSA-1571-1 openssl—predictable random number generator. `http://www.debian.org/security/2008/dsa-1571`, 13 May 2008.

[176] DigiCert Sdn. Bhd. 3rd clarification statement by DigiCert Sdn Berhard. Post on company Web site. `http://www.digicert.com.my/news/news_20111111.htm`, 11 November 2011.

[177] DigiCert Sdn. Bhd. DigiCert's announcement. Post on company Web site. `http://www.digicert.com.my/news/news_20111104.htm`, 5 November 2011.

[178] DigiCert Sdn. Bhd. 2nd clarification statement by DigiCert Sdn Berhard. Post on company Web site. http://www.digicert.com.my/news/news_20111107.htm, 7 November 2011.

[179] H. Dobbertin. The status of MD5 after a recent attack. RSA Laboratries' CrytoBytes 2(2). ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto2n2.pdf, 1996.

[180] R. Duncan. Certificate Authorities struggle to comply with Baseline Requirements. Blog post on Netcraft Ltd. Web site. http://news.netcraft.com/archives/2013/09/23/certificate-authorities-struggle-to-comply-with-baseline-requirements.html, 23 September 2013.

[181] R. Duncan. How certificate revocation (doesn't) work in practice. Blog post on Netcraft Ltd. Web site. http://news.netcraft.com/archives/2013/05/13/how-certificate-revocation-doesnt-work-in-practice.html, 13 May 2013.

[182] R. Duncan. Would you knowingly trust an irrevocable SSL certificate? Blog post on Netcraft Ltd. Web site. http://news.netcraft.com/archives/2013/05/23/would-you-knowingly-trust-an-irrevocable-ssl-certificate.html, 23 May 2013.

[183] P. Eckersley. A Syrian man-in-the-middle attack against Facebook. Blog post on EFF blog. https://www.eff.org/deeplinks/2011/05/syrian-man-middle-against-facebook, 5 May 2011.

[184] P. Eckersley and J. Burns. An observatory for the SSLiverse. Talk at DEF CON 18. https://www.eff.org/files/DefconSSLiverse.pdf, July 2010.

[185] Electronic Frontier Foundation. The Sovereign Keys project. Project homepage. https://www.eff.org/sovereign-keys, 2011.

[186] Electronic Frontier Foundation. Sovereign Keys design document (work-in-progress) https://git.eff.org/?p=sovereign-keys.git;a=summary, 2012.

[187] Electronic Frontier Foundation. Homepage. https://www.eff.org/, 2013.

[188] K. Engert. Man-in-the-middle experience in Warsaw. Blog post on personal blog. https://kuix.de/blog/comments.php?y=11&m=06&entry=entry110616-171707, 16 June 2011.

[189] Entrust, Inc. Entrust bulletin on certificates issued with weak 512-bit RSA keys by Digicert Malaysia. Statement on Web site. http://www.entrust.net/advisories/malaysia.htm, November 2011.

[190] G. Foest and M. Pattloch. DFN-PKI: Neues Konzept ermöglicht einfachen Einstieg in die Welt der Zertifikate. DFN Mitteilungen 68. https://www.pki.dfn.de/fileadmin/PKI/Konzept_DFN-PKI.pdf, June 2005.

[191] D. Forstrom. Microsoft releases security advisory 2607712. Post on Microsoft Security Response Center Web site. http://blogs.technet.com/b/msrc/archive/2011/08/29/microsoft-releases-security-advisory-2607712.aspx, 29 August 2011.

[192] Fox-IT. Black Tulip. Report of the investigation into the DigiNotar Certificate Authority breach. http://www.rijksoverheid.nl/bestanden/documenten-en-publicaties/rapporten/2012/08/13/black-tulip-update/black-tulip-update.pdf, August 2012.

[193] B. Fung. How Britain's new cyberarmy could reshape the laws of war. Article in The Washington Post, online edition. http://www.washingtonpost.com/blogs/the-switch/wp/2013/09/30/how-britains-new-cyberarmy-could-reshape-the-laws-of-war/, 30 September 2013.

[194] GlobalSign. Security incident report. GlobalSign's official response to the recent security incident. Press release. https://www.globalsign.com/company/press/121311-security-incident-report.html, 13 December 2011.

[195] D. Goodin. Web authentication authority suffers security breach. Article in The Register. http://www.theregister.co.uk/2011/06/21/startssl_security_breach/, 21 June 2011.

[196] I. Grigg. Brief history of attacks on CAs. Post in newsgroup mozilla.dev.security.policy. https://groups.google.com/d/msg/mozilla.dev.security.policy/GX18tf9D01U/QWKhAfho3m4J, 4 March 2012.

[197] P. Gutmann. Trouble at StartCom? Post in newsgroup mozilla.dev.security.policy. https://groups.google.com/d/msg/mozilla.dev.security.policy/7gW5yzk0wuc/MWTsSbWWn3QJ, 17 June 2011.

[198] N. Heninger and J. A. Halderman. Fast pairwise GCD computation. Web site with source code. https://factorable.net/resources.html, 2012.

[199] Hepner, Clint and Earl Zmijewski. Defending against BGP man-in-the-middle attacks. Talk at BlackHat DC. https://www.renesys.com/tech/presentations/pdf/blackhat-09.pdf, February 2009.

[200] R. Holz. root-store-archaeology. Source code repository. https://github.com/ralphholz/root-store-archaeology, 2013.

[201] R. Hurst. Trust the math. Choose your friends wisely. Blog post on GlobalSign Inc. Web site. https://www.globalsign.com/blog/trust-the-math-choose-your-friends-wisely.html, 10 September 2013.

[202] I. Ristić. State of SSL. Talk at InfoSec World. http://blog.ivanristic.com/Qualys_SSL_Labs-State_of_SSL_InfoSec_World_April_2011.pdf, April 2011.

[203] ICANN. Message from Doug Brent, ICANN Chief Operating Officer. Blog post on ICANN Web site. https://community.icann.org/display/alac/Trusted+Community+Representatives, 16 April 2010.

[204] ICANN and VeriSign Inc. Status update, 2010-07-14. Post on Root DNSSEC Web site. http://www.root-dnssec.org/2010/07/14/status-update-2010-07-14/, 14 July 2010.

[205] Internet Storm Center. Web site. https://isc.sans.edu, 2013.

[206] A. Langley. Issue 102530: Need to kill "Digicert Sdn. Bhd." CA (nothing to do with the better known "DigiCert" CA). Entry in Chromium issue tracker. http://code.google.com/p/chromium/issues/detail?id=102530, 1 November 2011.

[207] A. Langley. Why not Convergence? Blog post on personal blog. https://www.imperialviolet.org/2011/09/07/convergence.html, 7 September 2011.

[208] A. Langley. Revocation checking and Chrome's CRL. Blog post on personal blog. http://www.imperialviolet.org/2012/02/05/crlsets.html, 5 February 2012.

[209] A. Langley. CAA records on google.com. Post on CA/Browser Forum mailing list. https://cabforum.org/pipermail/public/2013-June/001716.html, 19 June 2013.

[210] A. Langley. Enhancing digital certificate security. Blog post on Google Online Security Blog. http://googleonlinesecurity.blogspot.de/2013/01/enhancing-digital-certificate-security.html, 3 January 2013.

[211] A. Langley. extract-nss-root-certs. Source code repository. https://github.com/agl/extract-nss-root-certs, 2013.

[212] R. Loden. Bug 556468—investigate incident with RapidSSL that issued SSL certificate for portugalmail.pt. Entry in Bugzilla@Mozilla. https://bugzilla.mozilla.org/show_bug.cgi?id=556468, 1 April 2010.

[213] G. Markham. Bug 698753—Entrust SubCA: 512-bit key issuance and other CPS violations; malware in the wild. Entry in Bugzilla@Mozilla. https://bugzilla.mozilla.org/show_bug.cgi?id=698753, 1 November 2011.

[214] G. Markham. Responses from Comodo. Post in newsgroup mozilla.dev.security.policy. https://groups.google.com/d/msg/mozilla.dev.security.policy/t5ItTYGsZYo/OnP3NoZ3MDYJ, 8 April 2011.

[215] M. Marlinspike and T. Perrin. tack. Source code repository. https://github.com/tack, 2013.

[216] Maxmind Inc. GeoLite free downloadable databases. Web site. http://dev.maxmind.com/geoip/legacy/geolite/, 2013.

[217] N. McBurnett. PGP Web of Trust statistics. http://bcn.boulder.co.us/~neal/pgpstat/19960101/, 1996.

[218] N. McBurnett. PGP Web of Trust statistics. http://bcn.boulder.co.us/~neal/pgpstat/, 1997.

[219] R. C. Merkle. Method of providing digital signatures. Patent US 4309569, January 1982.

[220] Microsoft. Erroneous VeriSign-issued digital certificates pose spoofing hazard. Microsoft Security Bulletin MS01-017. http://technet.microsoft.com/en-us/security/bulletin/ms01-017, 22 March 2001.

[221] Microsoft. Microsoft root certificate program. Article on Microsoft TechNet. http://technet.microsoft.com/en-us/library/cc751157.aspx, 15 January 2009.

[222] Microsoft. Microsoft security advisory (2641690). Fraudulent digital certificates could allow spoofing. http://technet.microsoft.com/en-us/security/advisory/2641690, 10 November 2011.

[223] Microsoft. Introduction to the Microsoft root certificate program. Portal to Microsoft's root store program. http://social.technet.microsoft.com/wiki/contents/articles/3281.introduction-to-the-microsoft-root-certificate-program.aspx, 2013.

[224] Microsoft. Microsoft security advisory (2798897). Fraudulent digital certificates could allow spoofing. http://technet.microsoft.com/en-us/security/advisory/2798897, 3 January 2013.

[225] O. Mikle. dns-scraper. Source code repository. https://github.com/hiviah/dns-scraper.

[226] Y. Minsky, K. Fiskerstrand, and J. Clizbe. sks-keyserver. Homepage of the SKS implementation. https://bitbucket.org/skskeyserver/sks-keyserver/wiki/Home, 2013.

[227] H. D. Moore. Debian OpenSSL predictable PRNG toys. Web site. Originally at http://digitaloffense.net/tools/debian-openssl/, mirrored at: http://downloads.corelan.be/debian-openssl, 2008–2014.

[228] Mozilla. Dates for phasing out MD5-based signatures and 1024-bit moduli. Post on MozillaWiki. https://wiki.mozilla.org/CA:MD5and1024.

[229] Mozilla. Mozilla CA Certificate Store. Portal to Mozilla's root store program. http://www.mozilla.org/projects/security/certs/, 2013.

[230] Mozilla. mozilla-central. Portal for viewing the source code in the browser. https://mxr.mozilla.org/mozilla-central/source/, 2013.

[231] Mozilla. Mozilla included CA certificate list. http://www.mozilla.org/projects/security/certs/included/, 2013.

[232] Mozilla. Mozilla pending CA certificate list. http://www.mozilla.org/projects/security/certs/pending/index.html, 2013.

[233] Mozilla. Mozilla source code (CVS). Instructions for obtaining the source code. https://developer.mozilla.org/en-US/docs/Mozilla_Source_Code_(CVS), 2013.

[234] P. Mutton. Compromised GlobalTrust database is published online. Blog post on Netcraft blog. http://news.netcraft.com/archives/2011/04/04/compromised-globaltrust-database-is-published-online.html, 4 April 2011.

[235] E. Nigg. Unbelievable! Post in newsgroup mozilla.dev.security.policy. https://groups.google.com/d/msg/mozilla.dev.tech.crypto/nAzIKSBEh78/7GEZ4f57F-cJ, 22 December 2008.

[236] E. Nigg. Untrusted certificates. Blog post on StartCom blog. https://blog.startcom.org/?p=145, 23 December 2008.

[237] E. Nigg. Cyber war. Blog post on StartCom blog. https://blog.startcom.org/?p=229, 9 September 2011.

[238] J. Nightingale. Removing the RSA Security 1024 V3 root. Blog post on Mozilla Security Blog. http://blog.mozilla.org/security/2010/04/06/removing-the-rsa-security-1024-v3-root/, 6 April 2010.

[239] J. Nightingale. DigiNotar removal follow up. Blog post on Mozilla Security Blog. https://blog.mozilla.com/security/2011/09/02/diginotar-removal-follow-up/, 2 September 2011.

[240] J. Nightingale. Firefox blocking fraudulent certificates. Blog post on Mozilla Security Blog. http://blog.mozilla.org/security/2011/03/22/firefox-blocking-fraudulent-certificates/, 22 March 2011.

[241] J. Nightingale. Fraudulent *.google.com certificate. Blog post on Mozilla Security Blog. https://blog.mozilla.org/security/2011/08/29/fraudulent-google-com-certificate/, 29 August 2011.

[242] J. Nightingale. Revoking trust in DigiCert Sdn. Bhd. intermediate Certificate Authority. Blog post on Mozilla Security Blog. https://blog.mozilla.org/security/2011/11/03/revoking-trust-in-digicert-sdn-bhd-intermediate-certificate-authority/, 3 November 2011.

[243] NIST. Secure Hashing. Approved Algorithms. Web site linking to specifications. http://csrc.nist.gov/groups/ST/toolkit/secure_hashing.html, 2006.

[244] NIST. Vulnerability Summary CVE-2011-3389. http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-3389, June 2011.

[245] P. Eckersley and J. Burns. Is the SSLiverse a safe place? Talk at 27th Chaos Computer Congress. https://www.eff.org/files/ccc2010.pdf, 2010.

[246] H. P. Penning. Analysis of the strong set in the PGP Web of Trust. http://pgp.cs.uu.nl/plot/, 2013.

[247] Perspectives Project. Project homepage. http://perspectives-project.org/, 2013.

[248] Planet Lab. Web site. https://www.planet-lab.org, 2013.

[249] Ralph Holz. Homepage with data sets of active scans. https://pki.net.in.tum.de, 2013.

[250] T. Riedmaier and R. Holz. Crossbear. Source code repository. https://github.com/crossbear/Crossbear, 2013.

[251] I. Ristić. Internet SSL Survey 2010. Talk at BlackHat. https://media.blackhat.com/bh-us-10/presentations/Ristic/BlackHat-USA-2010-Ristic-Qualys-SSL-Survey-HTTP-Rating-Guide-slides.pdf, July 2010.

[252] T. Ritter. New standards for browser-based trust—the recent acceleration of improvements. White paper. http://ritter.vg/p/2012-TLS-Survey.pdf, March 2012.

[253] F. Rosch. Stripping OCSP from Chrome will not improve browser security. Blog post on Symantec Connect blog. http://www.symantec.com/connect/blogs/stripping-ocsp-chrome-will-not-improve-browser-security, 18 December 2012.

[254] R. Sandvik. Security vulnerability found in Cyberoam DPI devices (CVE-2012-3372). Blog post on Tor Project blog. https://blog.torproject.org/blog/security-vulnerability-found-cyberoam-dpi-devices-cve-2012-3372, 3 June 2012.

[255] D. Schaefer. Implement 'shadow server' data validation. Entry in issue tracker, https://github.com/danwent/Perspectives-Server/issues/26, 9 October 2013.

[256] K. Seifried. Breach of trust. Article in Linux Magazine. http://www.linux-magazine.com/Issues/2010/114/Security-Lessons-Spoofed-Browsers, May 2010.

[257] K. Seifried. Improper SSL certificate issuing by CAs. Post in newsgroup mozilla.dev.tech.crypto. https://groups.google.com/d/msg/mozilla.dev.tech.crypto/qxwhY14H2Bg/iMDVmejNnI8J, 1 April 2010.

[258] R. Sleevi. [cabfpub] upcoming changes to Google Chrome's certificate handling. Post on CA/Browser Forum public mailing list. https://cabforum.org/pipermail/public/2013-September/002233.html, 24 September 2013.

[259] B. Smith. Bug 825022—deal with TURKTRUST mis-issued *.google.com certificate. Entry in Bugzilla@Mozilla. https://bugzilla.mozilla.org/show_bug.cgi?id=825022, 27 December 2012.

[260] A. Sotirov, M. Stevens, J. Appelbaum, A. Lenstra, D. Molnar, D. A. Osvik, and B. de Weger. MD5 considered harmful today. Online version of paper. http://dl.packetstormsecurity.net/papers/attack/md5-considered-harmful.pdf, 2008.

[261] Start Commercial (StartCom) Ltd. Critical event report. White paper, linked in StartCom blog post 'Full disclosure' from 3 Jan 2009 https://blog.startcom.org/wp-content/uploads/2009/01/ciritical-event-report-12-20-2008.pdf [*sic!*], January 2009.

[262] M. Stumpf. Bug 477783—Equifax not comforming to Mozilla CA Certificate Policy (7). Entry in Bugzilla@Mozilla. https://bugzilla.mozilla.org/show_bug.cgi?id=477783, 10 February 2009.

[263] P. Tate. Subordinate-CAs from trusted roots for 'managing encrypted traffic'. Post in newsgroup mozilla.dev.security.policy. https://groups.google.com/d/msg/mozilla.dev.security.policy/ehwhvERfjLk/XyHxrYkxdnsJ, 2 February 2012.

[264] Team Cymru. Team Cymru community services. Web site. http://www.team-cymru.org/Services/ip-to-asn.html#whois, 2013.

[265] The Hacker's Choice. SSL/TLS in a post-PRISM era. Entry in wiki. https://wiki.thc.org/ssl, 2013.

[266] The International Grid Trust Federation. Web site. http://www.igtf.net/, 2011.

[267] The OpenSSL Project. OpenSSL. Cryptography and SSL/TLS toolkit. Homepage. http://www.openssl.org/, 2013.

[268] Thoughtcrime Labs/IDS. Convergence. Project homepage. http://convergence.io, 2011.

[269] Tor Project. Project homepage. https://www.torproject.org/, 2013.

[270] Trustwave Spider Labs. Blog post on company blog. http://blog.spiderlabs.com/2012/02/clarifying-the-trustwave-ca-policy-update.html, 4 February 2012.

[271] C. von Loesch, G. Toth, and M. Modell. Certificate patrol. Mozilla Web page of Firefox add-on. https://addons.mozilla.org/en-us/firefox/addon/certificate-patrol/, 2012.

[272] D. Wendtland. Q: DNS scheme for privacy-sensitive users. Reply to thread with this topic in Google group perspectives-dev. https://groups.google.com/d/msg/perspectives-dev/IQq0uhJq9Mo/L_18Yebb5ckJ, 29 September 2013.

[273] K. Wilson. Recommend removing RSA Security 1024 V3 root certificate authority. Post in newsgroup mozilla.dev.security.policy. https://groups.google.com/d/msg/mozilla.dev.security.policy/tkk6KFunmZg/3PHvml-n_CYJ, 2 April 2010.

[274] K. Wilson. Mozilla communication: Action requested by March 2, 2012. Post in newsgroup mozilla.dev.security.policy. https://groups.google.com/d/msg/mozilla.dev.security.policy/6CX23NVaUvY/3S_TjkcebOwJ, 17 February 2012.

[275] S. Xenitellis. Certificate watch. Mozilla Web page of Firefox add-on. https://addons.mozilla.org/en-US/firefox/addon/certificate-watch/, 2012.

[276] M. Zusman. Announcement of Thawte compromise. Post on Twitter. https://twitter.com/schmoilito/statuses/1089348859, 31 December 2008.

[277] M. Zusman. Domain validated SSL certificates. Blog post on personal blog. http://schmoil.blogspot.de/2008/08/domain-validated-ssl-certificates.html, 25 August 2008.

[278] M. Zusman. Criminal charges are not pursued: hacking PKI. Talk at DEF CON 17. http://www.defcon.org/images/defcon-17/dc-17-presentations/defcon-17-zusman-hacking_pki.pdf, August 2009.

[279] M. Zusman. Nobody is perfect. Blog post on personal blog. http://schmoil.blogspot.de/2009/01/nobody-is-perfect.html, 1 January 2009.

List of figures

List of tables

List of algorithms and listings

ISBN 3-937201-41-6

ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)