Alternatives to X.509

Michael Gielesberger
Supervisor: Ralph Holz
Future Internet Seminar - Winter Term 2012/2013
Chair for Network Architectures and Services
Faculty of Computer Science, Technische Universität München
Email: gielesbe@in.tum.de

ABSTRACT

Secure connections on the Internet are crucial to many applications. Nowadays infrastructure for securing connections on the Internet heavily relies on the X.509 public-key infrastructure (PKI), which is maintained by Certificate Authorities (CAs). Recent attacks on CAs showed the fragility of the current X.509 system.
In order to alleviate or entirely fix problems with the X.509 system, different techniques and systems have been proposed. The proposed systems can be categorized into 4 different concepts: DNS-based, Pinning-based, Notary-based, and Transparency-based approaches. This paper gives an overview over these concepts and outlines their advantages and disadvantages. It also covers some of the best-known representatives of the concepts.
Sovereign Keys, a Transparency-based approach, is discussed in detail.

Keywords

X.509, PKI, Sovereign Keys, Certificate Transparency, Convergence, Perspectives

1. INTRODUCTION

In nowaday's widespread use of the Internet, problems have arisen which were unforeseen when the core infrastructure was developed between the 60s and the 80s. In the small networks of those days attacks on the confidentiality or authenticity of connections were at most a theoretical issue. Today's usage of the Internet includes applications with high security requirements, such as e-commerce applications. Because of the ever growing need for secure communication, different protocols were developed to allow secure connections over the Internet. In particular the Secure Socket Layer/Transport Layer Security protocol suite (SSL/TLS) is widely used nowadays to guarantee authentication, data confidentiality, and data integrity. SSL/TLS is used in conjunction with many different protocols but the best-known domain is securing the Hypertext Transfer Protocol (HTTP) with HTTPS.
Like most protocols which are used at the moment to secure connections over the Internet, SSL/TLS is based on asymmetric cryptography, at least for key exchange purposes. This leads to a necessity of distributing public keys. As an approach in which every host exchanges its public key with every other host does not scale (there would be $O(n^2)$ needed key transfers) a hierarchical approach for key distribution was introduced. The X.509 public-key infrastructure (PKI) offers a solution where only a few root public keys need to be transferred to the hosts. The root keys belong

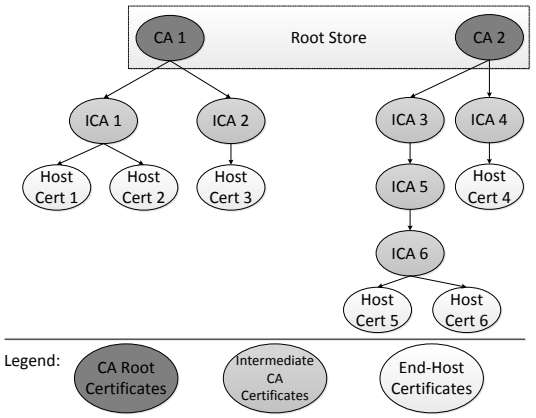

Figure 1: The X.509 hierarchy

to Certificate Authorities (CAs). The CAs issue certificates to other entities, in which they assert the binding of the entity's public key to their identity, in case of the WWW a domain name. CAs may also delegate signing privileges to other entities, which constitute an Intermediate Certificate Authority. X.509 builds up a certificate hierarchy with the CAs' certificates as roots, some optional intermediate certificates, and the hosts' certificates as leaves. The hierarchy is depicted in Figure 1. This results in trust chains. Clients, e.g. web browsers, have a so called root store, where they store root certificates which they trust. With these trusted root certificates, clients are able to verify a host's certificate by going through the chain of certificates until they reach one which is in their root store. Commonly clients ship with many root certificates: A common web browser trusts over 600 entities which are able to issue certificates [2].
The biggest problem with this approach is that every CA is equally able to issue certificates for every entity. When an attacker makes a CA assert the binding of the attacker's public key with another entity's identity, he gains a rogue certificate, which cannot be distinguished from real ones and thus the whole security system is rendered useless. This makes the whole system as weak as the weakest CA in the hierarchy. Recent attacks on the existing PKI have shown the fragility of the current system. With attacks on DigiNotar [14] and Comodo [3] certificates for high profile sites such as `addons.mozilla.org` or `login.facebook.com` have been falsely issued and used. Such rogue certificates allow attack-

doi: 10.2312/NET-2013-02-1_07

ers to perform Man-in-the-middle (MitM) attacks. There are also concerns that some state-run CAs, from which certificates are in root stores of clients, may issue rogue certificates for surveillance purposes [17]. Today's approach to deal with such attacks is to revoke compromised certificates, but the detection of a compromise may take several days or may never be detected [14]. And the techniques for revocation either do not scale well in the case of Certificate Revocation Lists (CRLs) or are flawed in the case of the Online Certificate Status Protocol (OCSP) [12], [9].
Many extensions or alternatives to the current X.509 system have been proposed to alleviate and fix the current problems before an attacker is able to use rogue certificates for a longer period of time. There are different approaches, which can be categorized into different categories: DNS-based (Domain Name System based) approaches try to store additional security information directly with the DNS records of the domains in order to prevent rogue certificates. Pinning-based approaches anchor certificates by comparing the certificates of a new connection with either already seen or out-of-band distributed certificates. Notary-based approaches try to expose rogue certificates by viewing connections to a secured server from different locations on the Internet and determine if there is a common consensus on the seen certificates. Transparency-based approaches force the publication of certificates or another validating key in a way that makes it possible for a client to compare and verify that the certificate it sees is the one the domain owner intended to use. Domain owners are also able to check all existing certificates or keys needed for validating a certificate for their domains with that approach.
The Sovereign Keys (*SK*) project by the EFF [6] belongs to the Transparency-based approaches. It has some unique features, which make it interesting to explore more in-depth.

The remainder of this paper is structured as follows: in section 2 the four mentioned approaches are introduced in general and some of the most important systems based on those approaches are presented. The following section 3 presents and analyses the unique features and techniques of the *SK* project more in depth. The final section 4 concludes the findings of this paper.

2. ALTERNATIVE CONCEPTS

In order to fix flaws in the current X.509 PKI system, several different techniques have been proposed. This section gives a brief overview of the most promising and common techniques. The best-known systems, which implement the technique, are named and some interesting specifics are explained for those systems.

2.1 DNS-based

DNS-based techniques take advantage of the fact that for every connection to a domain on the web a DNS-server has to be contacted in the first place to translate the domain name to the corresponding IP. As this connection needs to be established anyway (except for cached or locally assigned domain names), DNS servers seem to be a good place to store additional security relevant data. To be able to securely transfer data with the DNS the use of DNNSEC, which is still not wide-spread, is a prerequisite for most of these systems. With the use of DNS servers it is unnecessary to build up a whole new infrastructure to distribute certificates. The general idea is to publish additional security information as a new record type along with the other records in the DNS. One can differentiate between client side checked records and CA checked concepts:
For the client side concept a hash or a fingerprint of a public key is stored in the DNS. The hash or fingerprint is then compared with the actual certificate a client gets when connecting to the secured server. With this there is another channel which needs to be compromised by an attacker in order to conduct a MitM attack. The domain owner may decide if the single hash of the certificate, which belongs to the server's domain should be directly stated or the hash of a certificate further up on the certificate chain. By using the latter it is possible to state the fingerprints of the public key of the CA which issued the domain's certificate. This allows a domain owner to use different certificates but still narrow down which CA may issue certificates for the domain. A connecting client would check if there is a certificate along the certificate chain which matches the hash/fingerprint stated in the DNS and just allow the connection if this check succeeds. If an attacker is able to compromise a CA it is just possible to generate certificates for domains which have the fingerprint of the compromised CA stated in their DNS security record. When the attacker generates a rogue certificate for a domain, which states another, not compromised CA's public key fingerprint in the DNS security record, the certificate will be detected as invalid.
For CA side checked concepts the information is just checked by a CA which is asked to issue a certificate. If an attacker tries to get a certificate issued from a CA, the CA should look up the security information in the DNS for that domain and check if it is allowed to issue a certificate for it. Such a system does not offer any protection against CAs which just do not check those entries as there is no concept how to force CAs to check the entries.
DNS-based systems help against common attacks but have the disadvantage that they offer no protection against state-run attacks or surveillance activities. As states normally control their country-code top-level domain (ccTLD) they can completely change the DNS entries for domains in their ccTLD. E.g. Libya controls the `.ly` ccTLD, and thus can gain control over the domain `bit.ly`.
The best-known system, which belongs to the category of client side checked DNS-based concepts is DANE (The DNS-Based Authentication of Named Entities) [7].

2.2 Pinning-based

The general idea of pinning is to compare the certificate a client sees during a current connection attempt with other already seen certificates or hashes of certificates. From the first connection on it should additionally be required to use a secured connection, as otherwise simple attacks are possible, where a secured connection just gets redirected to an unsecured one. The easiest implementation is to store the certificates when a client first connects to a SSL/TLS secured server and compare the certificates seen in subsequent connections with the already seen certificate. There are different systems which use simple methods like this, e.g. Certificate Patrol [16]. Another approach to pin certificates is to send an additional header when connecting over SSL/TLS to a server, which contains a hash of a public key in the certificate chain of the real certificate. This pin has a lifetime, which is specified by the server the client got the pin from.

doi: 10.2312/NET-2013-02-1_07

When a client connects to a server, it compares the certificates in the trust chain with the hash of the pin and just allows the connection if the hash matches some certificate in the chain. By using the hash of a certificate which is not the certificate of the domain itself, but a certificate which is further up the certificate chain, a server is able to just allow certificates which belong to a specific intermediate CA or a specific root CA.
A specific problem of this concept is the distribution of the pins. As pre-sharing pins for all SSL/TLS secured domains is not practical, the pins are normally sent to the client when it first connects to the server. Thus an attacker which controls the connection at the moment a client first connects to a secured service will still be able to compromise the connection.
Google's Chrome browser uses public-key pinning by already shipping some hard-coded pins for domains of Google and the anonymization service Tor.

2.3 Notary-based

The concept of a Notary-based approach is to look at a secured server from many different viewpoints on the network.

2.3.1 Concept

A way to find out about MitM attacks is to compare the connection to a specific server with connections to the same server from the perspective of different clients at different positions on the Internet. Those other clients, which also try to connect to the secured server, are called notaries. This includes not just notaries which are just connected to other Internet Service Providers (ISPs), but also in other countries, or even on other continents. The reasoning is the following: when many notaries on very different positions on the network see the same certificate, it is very likely the certificate the domain owner really intended to use. As attackers are normally not able to compromise the whole network, but just small parts of it, a rogue certificate used for a MitM attack in just a fraction of the whole network will lead to different views on the compromised domain and can thus be detected by the notary concept. With the Notary-based concept the whole checking of certificates boils down to finding out if there is a consensus on the certificate of a server throughout the whole network.
There are some problems with this concept: there are some sites which have different certificates for each single server. The best-known site employing such a scheme is Citibank [16]. Sites like that will raise alerts with a Notary-based approach as there will be no consensus on the seen certificates.
Another deficiency is that the Notary-based approaches cannot detect global attacks. An attack which would compromise every connection to a server may be possible when there is just one gateway connecting the server to the global network. When this single connection point to the network is controlled by an attacker, the attacker is able to compromise every connection to the server. This is a general problem with the concept as it does not check if the used certificate is actually the one the domain owner uses but just if everyone sees the same certificate [18].
Notary-based systems do not use an already established channel such as DNS. This is why they need to open a new channel, a side-channel, for their communication with the notary servers. As with every technique which uses side-channels there is also the practical problem with captive portals: normally when connecting through a pay-to-use portal to the Internet, e.g. at a hotel, all requests get directed to the login site and all services besides DNS are blocked until the user paid for the usage and the connection is opened up for normal use. As the notary service runs through a side-channel, at the time of connecting to the login site a user cannot validate this secured connection with a notary system.
Unlike some other approaches, Notary-based approaches are able to deal with self-signed certificates as it does not matter for the notaries if there is a path to a (trusted) CA or not.

2.3.2 Perspectives

One of the first and best-known systems using the Notary-based approach, is the Perspectives project by Carnegie Mellon University [19]. Perspectives was already published in 2008 and came with a proof of concept in the form of a Firefox Add-On.
The notary servers are the most important entities of Perspectives. Notary servers are decentrally organized and generally independent of each other. Notaries either already have a key history of the requested service and are able to answer with that key history or if the client requests the key history for a service the first time, the domain is added to the list of monitored services. Either way, the notary will get the certificate from the domain's service and answer with that key. A host running the Perspectives client asks a number of notaries when connecting to a secured service and asks them for the key history they have seen for the server. The list of monitored servers is periodically probed by each notary and all witnessed certificates are stored with a timestamp. This results in a key history for all monitored services. The key history includes all previously witnessed certificates, and with the timestamps, how long they have been seen. The connection to the notary servers is secured by public key cryptography where the notary servers' public keys need to be distributed to the Perspectives clients by an out-of-band mechanism.
As a notary server knows about all connection attempts of a client to secured services, the Perspectives system raises some privacy issues. This is one of the issues of Perspectives, which Convergence tries to fix.

2.3.3 Convergence

Moxie Marlinspike presented Convergence at the BlackHat USA conference in 2011 [13]. Convergence uses the same idea as the Perspectives project, but it tries to tackle a few deficiencies of the Perspectives system. In order to fix the privacy issues, Convergence locally caches certificates which have already been validated by the Convergence system for some time. This brings the advantage that notary servers will not find out about every connection attempt to a secure service. But on the other hand it introduces a time frame, where a previously validated certificate may get compromised, but still be trusted by the Convergence client. Another mechanism for the improvement of a client's privacy is the introduction of notary bouncers. A client may ask a notary to proxy its request to another notary server. This is an onion-style routing mechanism where the proxying notary knows which client requested a key history, but as it cannot read the request message it does not know for which service. The notary server, which gets contacted by the proxy notary, knows for which service the request was for but does not know which client requested that information.

doi: 10.2312/NET-2013-02-1_07

However as soon as an attacker controls the proxying notary and the second notary the client's requests are known to the attacker.
In contrast to Perspectives, Convergence is designed to be highly extensible. While Perspectives uses the current X.509 PKI to validate certificates, Convergence is able run notaries with different ways of validating certificates. It is possible to run notary servers which validate the certificates with just a certain set of trusted CAs or even with a completely different system, e.g. DANE or the OpenPGP Web of Trust. Convergence is the only system presented in this paper which views itself as a replacement rather than an extension for the existing X.509 PKI.

2.4 Transparency-based

The recent attacks on the X.509 PKI took a few days to be detected. This is due to the fact that there is no mechanism to look up all certificates issued for a certain domain. A domain owner may never find out about the existence of rogue certificates for the domain which were generated by some attacker. Such rogue certificates are just detected when a suspicious user finds out about such a certificate, or when a CA finds out that it got compromised and issued rogue certificates. The concept of Transparency-based approaches is to fix the existing X.509 PKI by publishing issued certificates or keys needed to validate a certificate in such a way that it is possible to find all valid certificates or keys needed to validate certificates for a certain domain. Transparency-based systems are a very promising approach, as they focus on the idea of clients being able to check if the certificate they see is really the one the service operator intends to be seen.

2.4.1 Concept

The general concept involves an infrastructure in which a certificate must be published before it becomes valid. One could describe the Transparency-based approaches as public log based approaches. This relatively easy idea brings up technical problems, which need to be solved to be usable in real world environments. Public logs need not just serve the currently valid certificates or keys needed to validate certificates but need to be able to give a history of the used certificates or keys for a specific domain. Otherwise trivial attacks are feasible where an attacker adds a rogue certificate to the log for a short period of time and deletes it from the log when the attack is over. Because of that Transparency-based systems use append-only data structures which ensure that all certificates ever issued for a specific domain are stored in the log. The result is that domain owners and other interested parties are always able to find out about all valid certificates or keys needed to validate a certificate for their domains. If an attacker is somehow able to add a rogue certificate to the log the real domain owner is at least able to easily find out about it and revoke it.

2.4.2 Certificate Transparency

Certificate Transparency [11] is a system currently developed by Google. Since the original proposal some details have been altered and the current design paper version 2.1a was recently released [10]. With Certificate Transparency all certificates have to be registered with public log servers. The log servers save the domains' certificates in an append-only data structure. This is done using an ever-growing Merkle Tree which is a cryptographic primitive offering the possibility to easily and continuously check the append-only property of the data structure and to ensure integrity.
When a domain owner registers a certificate with a log server, the hash of the certificate together with a timestamp of the registration is signed by the log server and sent back to the domain owner. This signature is the so called Signed Certificate Timestamp (SCT). The secured server has to send the SCT together with the actual certificate at every SSL/TLS connection attempt to the connecting client. For clients the SCT works as a proof that the certificate is registered with a log server. In addition to validating the certificate itself, like it is done in the current X.509 PKI system, the client checks if the SCT is correctly signed by a log server. Clients will get the log servers' public keys using out-of-bands mechanisms, like shipping them hard-coded in the client software. The system ensures that every certificate has to be registered with the log servers, because otherwise clients will not accept the connection. Domain owners should monitor the certificates which are registered for their domains and make sure that the rogue certificates get revoked.
To ensure that the log servers are not compromised and work honestly there are monitors and auditors. Monitors check that logged certificates are promptly visible on the log and that they are on the log legitimately. Domain owners should either take this role or use some service, which does it on their behalf. A domain owner is able to check if a new certificate gets added to the log after submission and to check if there are any certificates for the domain not created by himself.
Auditors can check that partial data on the logs is consistent with the current state of the log. E.g. clients can implement an auditor, which checks that all certificates they encounter are visible in the log. Both can be efficiently implemented because of attributes of the Merkle Tree.
Certificate Transparency has the advantage that it does not depend on side-channels, like many other systems do. The secured servers themselves send the needed data to the clients and they are able to verify the signature of a log server offline. The usage of side-channels is limited to additional checks, which are not necessary for every client for the system to work. As of today the proposal does not include any mechanisms for revocation of keys. However, the initial proposal brought up the idea of having another log for revocations together with a proof of non-existence similar to DNSSEC's NSEC records.
Although Certificate Transparency is meant to be deployed over a period of time, it is meant to be required for every domain certificate in the end. The authors suggest that a complete transition may be enforced by not allowing the issuance of any new certificates without their publication by the Certificate Transparency system after a certain date [8]. As certificates automatically expire and need to be renewed, this enforcement would gradually reach every certificate.

3. SOVEREIGN KEYS

Sovereign Keys is a project by the Electronic Frontier Foundation (EFF) [6] and is categorized as a Transparency-based system. Unlike Certificate Transparency, the *SK* system does not publish the certificates themselves but another type of key, the Sovereign Key. It still fulfills the classic definition of Transparency-based approaches as no certificate is considered valid without being cross-signed with a Sovereign Key

doi: 10.2312/NET-2013-02-1_07

and the Sovereign Keys themselves are indeed published. Note that there is no working prototype yet. This paper takes information from the proposal, which was first published in November 2011 and last updated in June 2012 [4] and the specifications writeup from July 2012 [20]. Those are still subject to change and a final implementation may differ in specifics. However, source code is already publicly available in the project's git repository.

3.1 The basic design

The major problem with the existing X.509 system is that CAs are able to issue valid certificates for a domain without needing any information or consent of the real domain owner. With *SK*, certificates are not valid without being cross-signed with a special private key of the domain owner, the so called Sovereign Key. This private key is supposed to

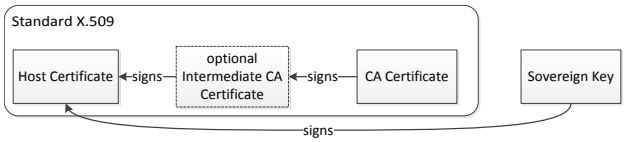

Figure 2: Relation between Certificates and Sovereign Keys

be secret and just known to the real domain owner. Because of that no one besides the real domain owner is able to carry out the final step of making a certificate valid.
In order for clients to be able to check the cross-signatures, the corresponding public keys need to be made available in some form. This is where the transparency aspect of the *SK* design comes in: all public keys of domain owners' Sovereign Keys need to be published in so called Timelines in order to be valid.
Timelines offer an append-only data structure for storing all Sovereign Keys ever used for domains. The integrity of the timelines can be cryptographically checked. When registering a domain name as a Sovereign Key in a Timeline, the Timeline server checks if the registration is authorized (see section 3.2.1). This relatively easy concept becomes quite complex when one addresses some real-world problems like the possibility of compromised Sovereign Keys or scalability.

3.2 Architecture

SK sees itself as an extension to the current X.509 PKI and keeps that infrastructure at least for bootstrapping. The system however introduces some new entities: Timeline servers, Mirrors and clients.

3.2.1 Timelines

The Timelines are the backbone of the *SK* system. They hold mappings between a (domain) name and Sovereign Keys. The concept envisages a semi-centralized data structure with the use of 10 to 30 Timeline servers [4]. The reason to use not just a single centralized server is to provide a Śdiversity of jurisdictions, operational philosophies and security implementations, and ideally to provide these properties even if some of the servers are compromised/disabledŠ [4]. Timeline servers may be identified with their Timeline Address (TADDR). The TADDR consists of the domain name and a port, by which they can be reached, and

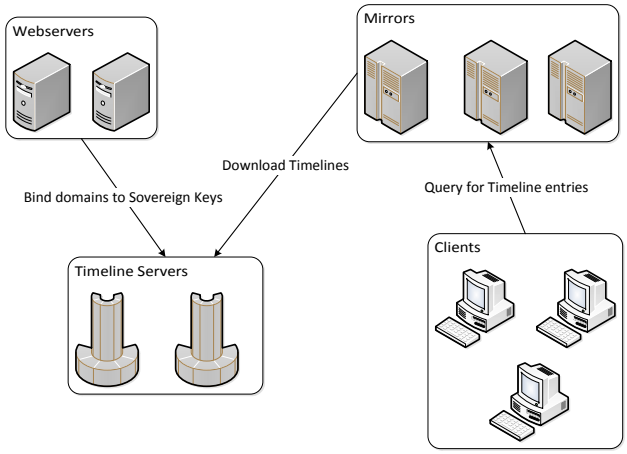

Figure 3: The Sovereign Keys architecture

a public key, which is used for secure connections to them using SSL/TLS. Besides that they have a unique Timeline ID (TID), which should only be used for a specific Timeline server. Even when a Timeline server gets shut down and removed from the system, the TID must never be assigned to another Timeline server. Each entry in the Timeline includes the Timeline server's TID so that it is possible to distinguish from which Timeline server an entry came from when they are merged into a single dataset.

Registration of a Sovereign Key: When a domain owner registers a Sovereign Key, a master-write is done to one Timeline server. However entries should be written to many Timeline servers, preferably in different geographic regions, to be safe in case a Timeline fails. The registrations to additional Timeline servers is done by reference where just a hash of the entry at the other Timeline server and that server's TID is saved. However when a Timeline server just adds a reference it has to cache a copy of the entry from the other Timeline server. This is done in order to have copies in case of a failure of the referenced Timeline server but to still have a globally unique entry.
In order to register or bind a domain name to a Sovereign Key, a domain owner sends a `Bind` message to a Timeline server. A `Bind` message includes the domain name for which the entry should be created. To prove the authorization, a domain owner must present a claim proof for the domain. This may be a CA-signed certificate from the current X.509 system or a key published with DNSSEC using DANE [7]. This proof of claim includes the whole certificate chain from the host certificate to the trusted CAs certificate, in order to be able to reproduce the trust decision later. The claim proof is also appended to the `Bind` entry in the Timeline. In order to not be fooled by compromised, but revoked keys Timeline servers have to verify the OCSP status of the certificates.

Cryptographic features of the Timelines: Timeline servers are not able to create fake entries as the data structure of the Timelines allows a cryptographic verification of the entries. All entries have an incrementing serial number which is local to every Timeline server and a monotonically non-decreasing timestamp. Every Timeline server has to

doi: 10.2312/NET-2013-02-1_07

sign every entry in its Timeline with its private key. All the listed properties may be checked easily:

- If there is an entry which is not correctly signed with the Timeline's private key, every checking party will notice and the entry will not be processed.
- If two correctly signed entries with the same serial number exist, a checking party can notice and has signed evidence of the violation of the incrementing serial number property.
- If there is a signed entry with a lower serial number but a higher timestamp, a checking party has signed evidence of the violation of the monotonicity of the timestamps.
- If a correctly signed entry with non-self-consistent data exists (like a non-verifiable certificate chain for a CA signed domain claim proof), a checking party has signed evidence of that inconsistency.

Because there is always signed proof for dishonest entries it is easy to verify the claim of a Timeline's misbehaviour. The design offers a mechanism to distribute the information of Timelines' misbehaviour and a Timeline server which distributes false data will be blacklisted in the system.

3.2.2 Mirrors

To have a scalable system and in order to decrease privacy issues, clients do not directly query the Timeline servers themselves, but Mirrors.
Mirrors store the whole Timelines data structure, collected from all trusted Timeline servers. For an easy initial deployment process Mirrors may be bootstrapped by downloading a set of Timeline entries up to a specific timestamp. As this might be a huge dataset (up to a few hundred gigabytes [5]) it is envisaged to be done with bittorrent or sending a hard disk with the dataset on it via conventional parcel services (disk-over-snailmail). The dataset is then updated to the current state by querying each trusted Timeline server for all entries since the last serial number in the dataset, that the mirror already holds. With this process, the mirrors gather the complete Timelines data structure, which is semi-centrally spread across the Timeline servers. Using this data set, the Mirrors are also able and supposed to check if all Timeline servers are behaving correctly. In case of a misbehaviour the misbehaving Timeline server gets flagged as rogue and this information is then passed to clients and other mirrors. In the end, misbehaving servers get removed from the list of trusted Timeline servers.
To create a complete view of the semi-centralized Timeline data structure, mirrors combine all entries from the different Timeline servers to produce a complete view on the Timelines. This is the view a client gets when querying Timelines for a specific domain. This set of entries is first ordered by their timestamps and, for entries which have the same timestamp, by the entry's TID.
Clients are able to choose which Mirror they want to use, just like DNS servers. The EFF envisages that Mirrors will also be run by ISPs just like they run DNS servers today. This would decrease the risk of privacy concerns as the queries would not get out of the ISP's network and the ISP is supposed to know to which domains a client connects to anyway because of DNS lookups. In addition, the design introduces an onion-routing style proxy feature for mirrors, where a client can ask a mirror to proxy a query to another mirror. Although the mirror knows which domains a client connects to, no other party on the network is able to read the communication between the client and the mirror as this connection is secured with SSL/TLS.

3.2.3 Clients

When a secured server is uses *SK*, the client connecting to it will bypass the X.509 certificate chain checking, as this has already been done by the Timeline server. The client will just need to get the Sovereign Key for that server from a Mirror and then check if the server's certificate has correctly been signed with the current Sovereign Key.
One thing to note about the cross-signature of the Sovereign Key is that the X.509 data format does not offer the ability to sign a certificate with more than one key. To be able to add the Sovereign Key signature to a host's certificate, which should already be signed by a CA, an extension to the X.509 data format is needed. However, clients which do not know about *SK* should not be affected because the extension is done in a way that legacy clients will ignore the *SK* part.

Trust towards Mirrors: When connecting to a secured server, clients query Mirrors for all Timeline entries regarding the server's domain name since the last entry that the client still has in its cache. Mirrors answer those requests with a list of all entries since the last entry of the client's cache together with a Timeline Freshness Message (TFM) of all involved Timeline servers. A TFM includes a timestamp, the highest serial number to that point in time, the serial number of the last CA update entry to that time, and a signature of the timeline server. All this is additionally signed by the mirror so that the mirror is liable for it. The TFMs are used to check the honesty of Mirrors. Like Mirrors check Timeline servers' integrity and honesty, clients check Mirrors: When a client gets an entry which has a higher serial number or a higher timestamp than stated in the TFM, it knows that the Mirror is misbehaving. When such a dishonest Mirror is found, the evidence of their misbehaviour will be reported and in the end those Mirrors will be flagged as bad and not used anymore.

Connection to a secured server: When a client connects to a SSL/TLS secured service it will query a Mirror for the Sovereign Key for that domain and service. The Mirror will answer with a timestamp-sorted list of entries for that specific domain. As the next step the client goes through the timestamp sorted list of entries it received from the Mirror. The first `Bind` entry is trusted, from there on every additional entry to the timeline must be signed with the bound Sovereign Key, unless a signed `Unbind` entry exists. If there is an `Unbind` entry, the chronological next valid `Rebind` entry will be trusted next (see section 3.4 for details). All correctly signed entries will be processed in this way and ultimately the client knows which Sovereign Key should be trusted for the connection to a specific service on a specific domain.
This Sovereign Key is then used to validate the domain's certificate by checking if it has been signed with the Sovereign

doi: 10.2312/NET-2013-02-1_07

Key. In order to offer more privacy and to be more robust against Mirror outages, clients can cache Sovereign Keys.

Caching: As Sovereign Keys are supposed to be valid for long periods of time, clients cache the keys locally. However, revocation checks will be done occasionally. Caching will also lead to a very small amount of data which needs to be transferred when the Timeline for a certain domain has already been cached as there are not be many updates to be expected.

3.3 Other features

SK offers some additional features. The domain owner may declare for each domain which services should use the *SK* system or if every (secured) service should use *SK*. By using this feature, a server is able to secure some services with *SK* but keep on using the standard X.509 system for some other services.
There is also a feature which allows domain owners to state an alternative route to a domain's service in case the normal route is blocked or unavailable. Such an alternative route would be tried when the normal route is not available or seems to be under attack. Some proposed alternative route modes are the use of proxies, VPNs or, especially promoted by the authors of *SK*, the anonymization service Tor. This allows an automated fallback, which may be useful e.g. in restrictive countries where the government tries to block access to certain domains.

3.4 Key management

The *SK* design already includes several solutions for key management use cases:

Revocation: Unlike Certificate Transparency, *SK* already has a concept to revoke compromised keys. In order to be able to rebind a previously unbound Sovereign Key, a domain owner may state the domain names of so called Rebinders. First, a compromised Sovereign Key needs to be unbound. This is done using an `Unbind` message, which needs to be signed with the Sovereign Key. An attacker which compromised a Sovereign Key could do that as well, but because of the Rebinder concept the attacker is not able to rebind a new Sovereign Key to it.
When one wants to bind a new Sovereign Key to the domain, this may just be done with a `Rebind` message, which needs to be signed with the Sovereign Key of one of the domain names previously stated as a Rebinder.

Change of domain owner: When a domain is transferred to another owner, the same mechanism as with revocation is used. First the current owner unbinds the current Sovereign Key and then the new owner will be able to rebind the new Sovereign Key through one of the Rebinders.

Expiry: To not block a domain by binding a Sovereign Key with a domain forever (see next item), Sovereign Keys expire and need to be renewed just like certificates in the X.509 system. One can never bind a Sovereign Key for longer than one holds ownership of the domain. This property is checked when binding a domain name to a Sovereign Key. However, it is possible to get domains for a hundred years, and thus it is possible to bind a Sovereign Key to a domain for that period of time.

Losing a Sovereign Key: One problem with the Sovereign Key concept is that if a domain owner loses the Sovereign Key, no changes to the entries are possible anymore. This may even lead to losing control over the domain until the Sovereign Key expires. The domain owner is not able to add any entries for the domain in the *SK* system anymore, as all entries must be signed with the now lost *SK*.
A Sovereign Key may be compared to physical property in this case: Once it is lost, it cannot be regenerated. Revocation of a lost key is impossible as the system requires an `Unbind` message for a revocation, which needs to be signed with the Sovereign Key. Thus Revocation is really just useful in the case of a compromised key.

3.5 Discussion

SK fixes some issues which are prevalent in other concepts or systems. The system is not affected by state-driven attacks on the DNS system, like DNS-based approaches are. When a government abuses its control over the DNS system and changes entries in there, the rightful domain owners are still in power over their Sovereign Key and the government would not be able to forge connections to the secured services running on that domain. In contrast to normal Pinning-based approaches, even the first connection of a client to a service running *SK* is secured. Also, *SK* is generally compatible with every X.509 PKI based protocol. In comparison to Notary-based systems, *SK*, just like every other Transparency-based system, does not just rely on what other hosts see on a network. Clients are able to check if the certificate they see is really signed by the rightful domain owner, which in the end is the only party able to decide if a certificate is the right one or not. Unlike Certificate Transparency, the *SK* project already has a concept for a revocation system.
However, there are some issues, which will take some effort to be fixed. Problems with captive portals should not be a problem for *SK* in general. But as *SK* needs a side-channel to check if a domain name is secured with a Sovereign Key, this will need a workaround: The authors propose to collect a list of known captive portal domains and then whitelist them [1]. This collection is proposed to be done with the help of a common web browser or a plugin for one. As the domains of the captive portals do not use *SK*, this does not affect the security. When a captive portal provider decides to incorporate *SK*, the captive portal will be removed from the whitelist.
Another issue are Denial-of-Service attacks on the Timelines data structure by writing a big amount of entries to it. This is proposed to be fixed with special rules for TLDs, which need to be paid for and rate-limiting for all domain spaces, which may be registered for free.
It will also be needed to take precautions that it is not possible to register a Sovereign Key for a domain not owned by the registering party. This may even render domains useless (see section 3.4). The authors propose to have a multi-staged process, which includes some explicit changes to the services run on the domain over a longer period of time [1].
However one issue is immanent to the *SK* system: a Sovereign Key may be seen as physical property. When it is lost it cannot be recovered nor revoked. This may even lead to loss of the domain until the Sovereign Key expires.

doi: 10.2312/NET-2013-02-1_07

4. CONCLUSION

The 2011 attacks on CAs alerted the Internet community that the current X.509 PKI is in a fragile state. There are many different approaches to fix the current system. As each system has its weaknesses, it may take the combination of some of those approaches to build a system which is able to fix all problems while still being deployable.

SK in its current proposed form is quite comprehensive and may as well be combined with DNS-based approaches. As there is no working prototype yet, it has to be seen if the concept will hold up to its promises or if some unforeseen technical problems get in the way.

Some of the outlined security concepts require quite big changes to be made to clients, servers, or even to the operational practices of CAs. Those changes will take quite some time and e.g. the EFF thinks that it will take a couple of years to build up the Sovereign Key system [5]. Some of the systems could be faster deployed, such as Pinning-based approaches, which are already used by Google for its own domains within the Chrome browser. Notary-based systems are also already available for end users. Even if some of the concepts will not establish themselves, they might be used in the period before a more comprehensive system is deployed. There are also other applications for some of the concepts: e.g. Notary-based systems can already be used to identify and track down the source of MitM attacks [15].

Time will tell which concepts will be widely accepted and if they hold up to their security promises.

5. REFERENCES

[1] P. Eckersley. 28C3: Sovereign Keys - A proposal for fixing attacks on CAs and DNSSEC. `https://www.youtube.com/watch?v=18pFTo3zVxk`, Dec. 2011. [last retrieved in September 2012].

[2] P. Eckersley. How secure is HTTPS today? How often is it attacked? `https://www.eff.org/deeplinks/2011/10/how-secure-https-today`, 2011. [last retrieved in September 2012].

[3] P. Eckersley. Iranian hackers obtain fraudulent HTTPS certificates: How close to a Web security meltdown did we get? `https://www.eff.org/deeplinks/2011/03/iranian-hackers-obtain-fraudulent-https/`, 2011. [last retrieved in September 2012].

[4] P. Eckersley. Sovereign Key Cryptography for Internet Domains. `https://git.eff.org/?p=sovereign-keys.git;a=blob;f=sovereign-key-design.txt;hb=master`, June 2012. [last retrieved in September 2012].

[5] Electronic Frontier Foundation. Sovereign Keys: A Proposal to Make HTTPS and Email More Secure. `https://www.eff.org/deeplinks/2011/11/sovereign-keys-proposal-make-https-and-email-more-secure`, Nov. 2011. [last retrieved in September 2012].

[6] Electronic Frontier Foundation. The Sovereign Keys project. `https://www.eff.org/sovereign-keys`, 2011. [last retrieved in September 2012].

[7] P. Hoffman and J. Schlyter. The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. RFC 6698, IETF, August 2012.

[8] A. Langley. Certificate Transparency. `http://www.imperialviolet.org/2011/11/29/certtransparency.html`, Nov. 2011. [last retrieved in September 2012].

[9] A. Langley. Revocation doesn't work. `http://www.imperialviolet.org/2011/03/18/revocation.html`, 2011. [last retrieved in September 2012].

[10] B. Laurie and E. Kasper. Certificate Transparency v2.1a. `http://www.links.org/files/CertificateTransparencyVersion2.1a.pdf`, Sept. 2012. [last retrieved in September 2012].

[11] B. Laurie and A. Langley. Certificate Transparency. `http://www.certificate-transparency.org/`, 2012. [last retrieved in September 2012].

[12] M. Marlinspike. Defeating OCSP With The Character '3'. `http://www.thoughtcrime.org/papers/ocsp-attack.pdf`, 2009. [last retrieved in September 2012].

[13] M. Marlinspike. SSL And The Future Of Authenticity. `https://www.youtube.com/watch?v=Z7Wl2FW2TcA`, Aug. 2011. [last retrieved in September 2012].

[14] Mozilla Security Blog. DigiNotar removal follow up. `https://blog.mozilla.com/security/2011/09/02/diginotar-removal-follow-up/`, 2011. [last retrieved in September 2012].

[15] 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 2012)*, volume 7459/2012 of LNCS, pages 217–234, Pisa, Italy, Sept. 2012. Springer Verlag.

[16] T. Ritter. New Standards for browser-based trust - The recent acceleration of improvements. `http://ritter.vg/p/2012-TLS-Survey.pdf`, March 2012. [last retrieved in September 2012].

[17] C. Soghoian and S. Stamm. Certified lies: Detecting and defeating government interception attacks against SSL. In *Proc. 15th. Int. Conf. Financial Cryptography and Data Security (FC'11)*, Mar. 2011.

[18] A. Steingruebl. Perspectives on "perspectives". `http://www.thesecuritypractice.com/the_security_practice/2010/04/perspectives-on-perspectives.html`, April 2010. [last retrieved in September 2012].

[19] D. Wendlandt, D. Andersen, and A. Perrig. Perspectives: Improving SSH-style host authentication with multi-path probing. In *Proc. USENIX Annual Technical Conference*, Boston, MA, June 2008.

[20] J. Wierzbicki. Sovereign Key Draft Specification. `https://git.eff.org/?p=sovereign-keys.git;a=blob_plain;f=spec.txt;hb=master`, July 2012. [last retrieved in September 2012].