Universität Karlsruhe (TH)
Institut für Telematik

TELEMATICS TECHNICAL REPORTS

SpoVNet Security Task Force Report

Ralph Holz, Heiko Niedermayer – Universität Tübingen
Christoph P. Mayer, Sebastian Mies – Universität Karlsruhe (TH)
Muhammad Adnan Tariq – Universität Stuttgart

Dezember, 21th 2009

TM-2009-3

ISSN 1613-849X

`http://doc.tm.uka.de/tr/`

Institute of Telematics, University of Karlsruhe
Zirkel 2, D-76128 Karlsruhe, Germany

SpoVNet Security Task Force Report

R. Holz[2], *C. Mayer*[1], *S. Mies*[1], *H. Niedermayer*[2], *M.A. Tariq*[3],

[1]Institut für Telematik, Prof. Zitterbart, Universität Karlsruhe (TH)
[2]Wilhelm-Schickard-Institut, Prof. Carle, Universität Tübingen
[3]Institut für Parallele und Verteilte Systeme, Prof. Rothermel, Universität Stuttgart

December 21, 2009

Contents

1. Introduction

Today's Internet would be unthinkable without a number of protocols that allow entities to communicate securely or to authenticate to each other. However, the original assumption of the Internet was that of a number of friendly hosts, who would not misbehave. The need for security only arose when the Internet was opened for practical purposes, including commerce. Security protocols were thus only introduced some time after the application protocols that they were supposed to protect. The result of this organic evolution is a patchwork of security layers and half-layers. It has been emphasized before that great care has to be taken to use these correctly. The issue of key distribution for authentication and encryption has also proved to be too complex for normal users (see, e. g., [Gutm02]). In short, Internet security is not intuitive to use, and not user-friendly at all. Yet security is an important feature in today's communication – becoming more important by the day, as the Internet is quickly turning into a substrate on which most future communication channels will rely.

To prevent the security patchwork approach that has evolved in the current Internet architecture, SpoVNet pursues *integrated security*. This implies that security is an inherent part of the SpoVNet architecture, including the SpoVNet services. The advantage of this approach is threefold: First, security is implemented as a common component in the SpoVNet architecture, removing code and data redundancy, and hence leaving less room for errors and failures. Second, security mechanisms can easily and safely be accessed by all SpoVNet components in a sound manner. And third, node-specific security policies can be implemented more easily.

The *Security Component* copes with two major issues: First, it maintains security state that is used for securing communication between nodes. For example, if a secure confidential link is needed between SpoVNet nodes, the Security Component uses the cryptographic Base Overlay identifiers to perform an authenticated key exchange. The exchanged keys are stored in the Security Component. Second, the Security Component manages policies for authorization, authentication, and confidentiality. Policies are determined by the node that creates the SpoVNet Instance. To enforce policies, the Security Component provides an interface to control communication and link establishment between nodes in different scenarios, e. g., bootstrap, join, maintenance, and leave phases of SpoVNet nodes. Services and applications can use this interface when upcoming operations are sensitive to SpoVNet security.

SpoVNet services and applications may need security features that go beyond what the Security Component can offer. In this case, the required functionality must be provided by the service or application itself, using the mechanisms offered by the Security Component.

1.1 Organization of the Security Task Force

The SpoVNet *Security Task Force* (STF) is composed of members from TP1, TP2, and TP3. This includes all SpoVNet projects that work on the SpoVNet architecture and its services. The composition guarantees that all aspects of the SpoVNet architecture and SpoVNet services are taken into consideration and a sound security design can be achieved. Due to the composition of the task force the primary language is English.

1.2 Overview of This Report

This report is structured as follows. Chapter 2 gives an initial motivation to security in SpoVNet and the Security Task Force in general. Furthermore, the scope of security in SpoVNet and types of SpoVNets with respect to security are presented. Security concepts and research topics in the scope of the Security Task Force are presented in Chapter 3. Chapter 4 presents exemplary scenarios and policies for SpoVNet Instances. Threat models that are used for the evaluation of mechanisms developed within this Task Force are detailed in Chapter 5. Chapter 6 names requirements of cryptographic primitives and higher level security concepts that are taken into account in SpoVNet security. The architecture of the Security Component and its embedding into the overall SpoVNet architecture are explained in Chapter 7. Finally, Chapter 8 shows how several SpoVNet components profit from the Security Component and how they use it to achieve their security goals.

2. Security for SpoVNet

In this chapter, we discuss the need for a Security Component in SpoVNet and also describe its role within the SpoVNet architecture. We identify security requirements and derive features the component must include.

2.1 Motivation

To prevent the security patchwork that has evolved in the past, SpoVNet pursues an approach which we have termed *integrated security*. Just like the Underlay Abstraction attempts to hide the properties of underlying network layers to SpoVNet applications, the Security Component provides a unified access to a wide range of security mechanisms. It abstracts from concrete implementations, allowing applications to define their security needs without requiring them to address the pecularities of security protocols. An integral part of the SpoVNet security concept are Policies. These will aid in the creation of secure and better-protected SpoVNets. The Security Component is accessible like a service, and all SpoVNet applications (and SpoVNet services as well) can draw on the features it offers.

Apart from this integration of security aspects into the SpoVNet philosophy, the abstraction has further advantages. The SpoVNet software remains extensible as new security features are added, and others, which might have become insecure, removed. All this can happen transparently to SpoVNet applications. On a technical level, it reduces the space for implementation errors.

In the following, we will outline the security requirements that we have identified for the Security Component. We discuss types of SpoVNet and explain the important role of the initiator.

2.2 Integration of Security in SpoVNet

Security in SpoVNet is implemented by the *Security Component* (SEC). This component integrates vertically into the SpoVNet architecture, as shown in Figure 2.1. Its interfaces can be accessed by all SpoVNet components. The SpoVNet Base, e. g., uses SEC for securing the Base Overlay and links in the Base Communication. SpoVNet services access the SEC to provide extended security functionality specific to the service: MCP-O, e. g., uses SEC to provide secure group communication in the Multicast-/Multipeer Overlay.

The SEC tries to utilize available security mechanisms of the underlay whenever possible. If available security mechanisms are not sufficient, e. g., to fulfil a security policy, SEC employs further security mechanisms. Finally applications and services may need security operations and therefore access SEC.

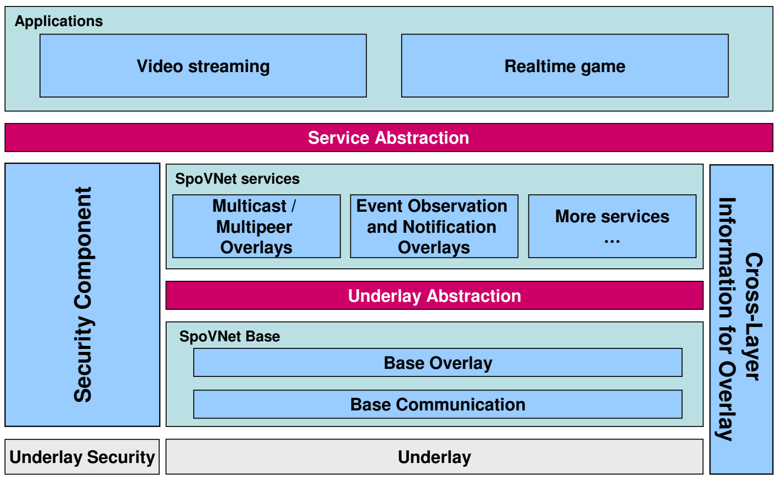


Figure 2.1: The Security Component in the SpoVNet architecture.

2.3 General Security Requirements

SpoVNets will be used on a wide range of devices, which have diverse access technologies. We can thus not assume that all underlays provide security functionality like confidentiality or integrity protection. For instance, it is reasonable to assume that stationary hosts might provide TLS or a comparable protocol, but other types of hosts, like small handheld devices, might not. Moreover, security in the underlay is often limited to security between two endpoints in the underlay, not the two endpoints of a SpoVNet communication. This is comparable to, e. g. , the use of SSL in Identity Federations [Libe07]: while SSL can be used between two TCP/IP endpoints, these endpoints are often not the same as the endpoints that run an Identity Federation protocol. Thus, the latter must employ security mechanisms of their own. The situation is similar with SpoVNet: if messages are routed through the Base Overlay, the endpoints of the communication are likely not the same as the TCP/IP endpoints that are involved in the underlay.

As a consequence, SpoVNet must feature a component that provides mechanisms to create secure communication links even if underlays do not support it. This component must provide cryptographic operations and be accessible for the SpoVNet Base, SpoVNet services and SpoVNet applications. This component will be the Cryptographic Engine.

2.4 SpoVNet-Specific Security Requirements

One important idea in the SpoVNet design is to liberate applications from the complexity of the underlying networks. Applications should be able to run even if mechanisms in the underlay change. The Security Component needs to support this as well. Applications should not have to define their security requirements in terms of the protocol and algorithms they wish to employ, but in a more abstract way. For example, they should be able to indicate that they need a secure connection of a certain (relative or absolute) strength, with added integrity protection. They should be able to rely on the Security Component for the verification of signatures and the like. The Security Component thus has to provide mechanisms for common cryptographic goals, like authentication or confidentiality; in addition to and building on the cryptographic operations that a Cryptographic Engine provides.

The complexity of SpoVNet security is increased by the decentralized and spontaneous nature of SpoVNets. As known from the work of Boyd [Boyd93], Trusted Third Parties (TTP) are a necessary ingredient in order to provide trust where two entities have no prior security context. In our SpoVNet settings, we will sometimes be able to provide such a TTP (also see 3.1.2). But in many others settings we will likely have to make do without it. Therefore this situation must be addressed as well.

One way to approach this is to rely to some degree on policies. These can define the actions that peers have to take in security-relevant situations (like admitting new nodes). Policies can be defined and signed by the initiator of a SpoVNet. The policies can define the type of the SpoVNet (see rest of the chapter) and additional information, e. g. in closed SpoVNets legitimate users must provide certain credentials.

2.5 SpoVNet Types

Different use-cases drive the need for differently protected SpoVNets. The distributed nature of a Peer-to-Peer approach requires us to support decentralized approaches when a server is not feasible. We summarize this by defining three dimensions according to which we categorize SpoVNets and their security. These dimensions are: a) the type of access control, b) the kind of administrative control, and c) the visibility of a SpoVNet.

2.5.1 Access Control

We distinguish the following types of SpoVNets:

Open SpoVNet: This type of SpoVNet does not enforce any access control at all. Any node can join. There is no verification of identity nor do nodes have to fulfil certain properties.

This type of SpoVNet is useful where security requirements are generally low or non-existent. A simple kind of application would be a look-up network, where keys are mapped to certain values (useful, e. g., for the look-up of bootstrap nodes). A more complex application would be swarm-distribution or streaming of videos. Note that while such a SpoVNet may not exercise access control, the distribution may still be protected by cryptographic means.

Closed SpoVNet: Only legitimate nodes may join this type of SpoVNet. Legitimacy must be proved by the joining node. The requirements may be quite variable. Closed SpoVNets would usually rely on authentication mechanisms (i. e. a node must prove its identity).

Closed SpoVNets are easy to realize if there is a central entity in the network that controls who joins the SpoVNet. It is much harder to achieve without such an entity; in the absence of a Trusted Third Party it becomes impossible. Policies are one way to improve this situation and add at least a certain level of security. Typically, applications with a closed user base would use Closed SpoVNets, e. g. messaging networks or gaming applications. Commercial applications are another use case.

We also define a special kind of SpoVNet where access may be restricted based on certain properties that nodes have to fulfil. We call this a Capability-restricted SpoVNet.

Capability-restricted SpoVNet: A capability-restricted SpoVNet is a SpoVNet that limits access to the SpoVNet by requiring nodes to fulfil certain properties, e. g. having enough bandwidth or belonging to a certain IP subnetwork. This is a form of authorization that does not require prior authentication. A typical example would be a streaming application that requires a certain bandwidth.

2.5.2 Administrative Control

This term refers to the mechanisms that a SpoVNet employs to achieve access control. This is most relevant for closed SpoVNets as open SpoVNets have no need for access control. We distinguish the following categories:

Centralized: In this type of SpoVNet, there exists a (single) central entity that controls access to the SpoVNet. This may be achieved either by requiring nodes to authenticate, or to fulfil certain properties, or both. The central entity may delegate its tasks to other trusted entities.

Decentralized: There is no single central entity in this SpoVNet that would be responsible for access control. Instead, access control is distributed on either selected nodes, or even all nodes in the network. Naturally, this has implications for the level of security that can be achieved.

Centralized networks are good solutions for commercial SpoVNets, where the administration needs to enforce restrictions and acts as a trusted party. Decentralized control is interesting for more spontaneous networks, like applications that use 'buddy lists' or dark nets [BEPW02] (friend-of-a-friend). In such cases, Web-of-Trust-like approaches (like PGP) to security become interesting.

2.5.3 Visibility

SpoVNets are not necessarily visible to everyone. Observers, unauthorized entities, or other SpoVNets should not always be able to detect and identify a SpoVNet instance. We distinguish the following two types:

Visible SpoVNet: This type of SpoVNet makes no effort to hide its presence on a given physical host. Any observer on the host can detect the SpoVNet and determine its ID.

Hidden SpoVNet: This type of SpoVNet remains hidden to an observer. This does not mean that an observer on the physical host cannot detect that there is a SpoVNet running. However, the observer cannot determine the ID of the SpoVNet and thus cannot decide which purpose it serves.

The visible SpoVNet is the standard. It also allows easier bootstrapping into the SpoVNet. An example of a hidden SpoVNet may be a private network over untrusted (e. g. wireless) links where an observer should not be able to determine the members and the identity of the SpoVNet. It can also be used as a weak form of access control when real access control is not (easily) applicable.

2.6 The Role of the Initiator

SpoVNets are spontaneous networks. An initiator has to start a SpoVNet at some point, others join later. The initiator thus has a special role – he defines the SpoVNet: its name, its ID, what type it is etc. We can thus consider the initiator the 'owner' of the SpoVNet.

Based on this distinct role, we can develop the concept that we call the *initiator-driven approach*. The idea is that, in many settings, SpoVNet security can rely on the initiator as a special entity. The initiator may act as a Trusted Third Party, or define policies and even enforce them. He can also delegate responsibilities to other nodes (e. g. further TTPs).

3. Security Concepts in SpoVNet

In this chapter, we will describe the key areas of our research, where we make contributions, and the solutions we propose. The Task Force has identified several research topics that are interesting and not covered by previous research done in the field, e. g., the integrated security approach described in 3.1.1.

3.1 Research Topics in the Security Task Force

The Security Task Force has identified four major research points to be investigated in the work of the Task Force. These points make the security approach in SpoVNet unique in comparison to other research that has been done in the field of security.

3.1.1 Integrated Security Approach

Most security designs either implement the security in the application itself or make use of mechanisms that are offered by the underlay. SSL/TLS is a popular example: An application may be designed to communicate via secure channels only and open a TLS connection on the host for this purpose. This design has a major drawback: The application is only aware of security that has been implemented in the application itself using a given library. It therefore will specifically and explicitly request functionality from the library which is a complex process and easily done wrongly.

Security in SpoVNet pursues a completely different approach, which we call the *Integrated Security Approach*: Security is an integral part of the SpoVNet architecture. Security is not left to applications but rather implemented transparently in the SpoVNet architecture. Applications and services do not need to be aware of specific security mechanisms but can operate using abstract, requirements-oriented primitives. The concept of Requirements and Policies provides this abstraction (see Section 3.1.3 below). The Security Component will translate these requirements into technical mechanisms and (if necessary) negotiate them with the Security Component of the other node that will provide appropriate security measures. This concept highly strengthens the overall security of SpoVNet Nodes and Instances.

The Integrated Security Approach has further advantages. Policies, e. g., can be enforced more easily and SpoVNet-device-wide. Whereas today, policies are enforced by the application and only application-wide (e. g. configuring a Web browser to use SSL).

3.1.2 Distributed Security Setting

SpoVNet instances are, by definition, created spontaneously. Scenarios where a central entity acts as a Trusted Third Party (TTP) are easy to address. However, we must also consider scenarios where we cannot rely on pre-defined TTPs, where global Trusted Third Parties like common Certification Authorities cannot help with issues like authentication or key distribution.

We address this distributed setting with two concepts: The first is our *initiator-driven approach*. In this scheme, the initiator of a SpoVNet instance has an important role. He determines the security policies and the type of SpoVNet that is to be established. He can, for example, determine that the SpoVNet is going to be a closed group (i. e. with access control) and that the initiator is going to be the sole TTP. Nodes might only be allowed to join via the initiator node, and only if they can present a certain credential. In other cases, the initiator might determine that the SpoVNet is a closed one, but that there are appointed delegates that can act as further TTPs and bootstrap nodes. Or the initiator might decide that the SpoVNet is open to anyone (no access control), but that public keys must be verified via a challenge-response upon joining of a node. The enforcement of these rules is then left to the nodes in the SpoVNet instance.

Clearly, these rules determine very different levels of security, and a SpoVNet Instance must be precise in the definition of the required security.

3.1.3 Requirements and Policies

Security in SpoVNet is directed by two fundamental schemes: Requirements and Policies.

Requirements

Requirements describe the security needs of a given application, service, or other SpoVNet component. They are passed to the Security Component, which interprets the description and translates it into appropriate security mechanisms. In this process the Security Component takes security mechanisms into account that are available on the host and its underlay. An application, e. g., may signal that it needs an authenticated and integrity-protected connection, but that confidentiality is not necessary. The Security Component translates this into appropriate mechanisms which implement the requested security requirements. The selection of security mechanisms relies, among others, on the availability of security mechanisms in the underlay. If the underlay does not offer the necessary mechanisms, the Security Component provides its own implementation of cryptographic routines (see Chapter 7.3).

Policies

Policies are a complementary concept. We distinguish two kinds of possible policies: Instance Policies and User-defined Policies, also called Device Policies.

Instance Policies define the overall security concept for a SpoVNet Instance and the measures a SpoVNet Node has to take to implement it. Each SpoVNet may have a different policy. For example, if a Trusted Third Party (TTP) is available for the SpoVNet Instance, the policy might indicate that this authority must determine whether a SpoVNet Node is allowed to join this SpoVNet Instance or not. Policies can, however, also be used to define security concepts for the case where a TTP is not available.

Apart from to SpoVNet Instance-specific policy definitions, policies can be defined SpoVNet-device-wide. In this case, the policy is predefined or given by the user. An example of such a SpoVNet Node-specific policy is that all connections that the SpoVNet Node takes part in must be encrypted. This policy can be enforced SpoVNet device-wide due to the Integrated Security Approach (see Chapter

3.1.1). Device-specific policies can interfere with requirements that an application or service define: an application requesting an unencrypted connection and a device-wide policy defining that only encrypted policies are allowed produces a clash of interest that must be resolved. Such clashes are always resolved in favor of stronger security. In the given example, the policy – defining that only encrypted connections are allowed – will overwrite the requirement for an unencrypted connection.

3.1.4 Extensibility and Exchangeability

Our integrated and requirements-oriented approach has further advantages: extensibility and exchangeability. Due to the fact that security is managed transparently—applications and services express their security requirements in abstract form—the actual mechanisms that are employed can easily be extended or exchanged inside the Security Component. This makes it possible to update weak mechanisms and replace them with stronger ones that provide the same functionality. Futhermore, adding new mechanisms with new functionality is possible and can be used to better represent the requirements given by the application.

3.2 End-to-End Security Concepts

SpoVNet knows three concepts for end-to-end security: authentication of nodes as well as confidentiality and integrity of data exchanged between two nodes. We use cryptographic canonical node names (CNNs)—with derived node identifiers (NodeIDs)—to implement those concepts (similar to [NiLD07, Aura05, MoNi06]). Each node chooses a key-pair and uses the public-key as its canonical node name. The key length and algorithm used is a SpoVNet specific system parameter (i. e., specified by the SpoVNet-Initiator). Using the generated key-pair nodes can prove identity, sign data, or use HMACs to ensure integrity as well as encrypt data by using a prior key exchange, e. g., by using an authenticated Diffie-Hellman key exchange algorithm. The parameters for those operations are specified in an algorithm independent manner. So, key lengths are specified as *absolute* values. For example, if an absolute key length of n-bits is specified, an attacker would need $2^{(n-1)}$ tries to break the algorithm. For translating between absolute and relative bit strengths we use mapping functions that need to be adapted over time when superior attacks become available.

We define the following parameters for a end-to-end link: for *authenticity* we use the established system security parameters for the canonical node name and use a challenge-response algorithm to ensure that a node is authentic. For *confidentiality* we exchange symmetric keys (e. g., with an authenticated Diffie-Hellman algorithm) with a given absolute key length. Security extensions (e. g., DTLS, TLS, IPsec) are used if they support encryption using the exchanged symmetric keys. For *integrity* keys are exchanged and used with an HMAC algorithm, if not already supported by the protocol. In this case the absolute bit length of the hash algorithm serves as security parameter.

4. Exemplary Scenarios

In this chapter, we describe several scenarios with different security requirements. We show the security requirements of these scenarios and which security concepts are used to achieve them.

4.1 Public TV Streaming

This example addresses public television streaming. Such an audio/video streaming scenario can be used to replace today's satellite and cable TV broadcasting systems and is related to the currently upcoming IPTV. In contrast to IPTV that uses IP Multicast, the SpoVNet scenario uses MCPO for streaming using *Application Layer Multicast* (ALM) mechanisms. In constrast to IPTV—that is only available intra ISP-wide due to deployment policies of IP Multicast—the ALM approach can be deployed Internet-wide.

Access Control: open A public television streaming is open for everyone and therefore does not employ access control[1]. Therefore, the access control type of the SpoVNet Instance is open.

Administrative Control: decentralized The public TV streaming scenario employs no access control and has no special policies that need to be enforced SpoVNet Instance-wide. Therefore, no centralized control instance is necessary. The administrative control of the scenario is therefore decentralized.

Visibility: visible As the streaming in this example is public, everyone is allowed to have knowledge of the existence of the SpoVNet Instance and is allowed to find the SpoVNet Instance. The discovery is therefore unrestricted and the SpoVNet Instance visible.

Although no access control and administrative control is employed, certain security mechanisms are necessary. The first is the TV consumers' necessity to verify that the TV stream is sent from the correct source. Due to the use of ALM, each receiver can possibly alter the video stream and forward it to its children in the ALM distribution structure. A receiver wants to be sure that the current stock market news that he is seeing is indeed sent from CNN and has not been tampered by a malicious node. As such integrity and source authentication must be employed that are implemented using the TESLA scheme [PSCT$^+$05].

[1]Please note that this does not apply to commercial Pay-TV streaming, which we are not discussing here.

4.2 Spontaneous Private Ad-hoc Network

In this example we sketch a network that is open and decentralized, yet provides a weak protection due to its hidden visibility. It additionally provides anonymous access with respect to user identity. The network could consist of users who jointly attend a meeting in a public place. Nonetheless, the event was initiated by someone who organized the event. The application could be data exchange and global chat.

The organizer serves as initiator and provides the policies that define how the SpoVNet operates and what security mechanism apply (open, decentral, hidden). To enable a hidden SpoVNet the organizer told the users the SpoVNetID and a secret out-of-band, e. g., orally. Other entities in the physical network can be found with lower layer mechanisms. They are contacted until a node of the corresponding SpoVNet is found. Their nodes then acquire the policies and give them to other new nodes that join later-on.

Access Control: open As noone knows all legitimate users and is able to provide a list of them, there is no individual access control. New users who want to join may get the information from other members, just as today in meetings one gets the WPA/WEP key for the wifi access.

Administrative Control: decentralized Nodes only locally enforce the policies. If a node is contacted via the underlay, it checks if the other node has the knowledge required by the hidden visibility. Otherwise it will not reply.

Visibility: hidden The nodes of the network only react to requests that include the correct SpoVNet ID and that encrypted this with the secret. The nodes encrypt at least SpoVNetID and NodeIDs.

As NodeIDs are cryptographic identifiers, nodes are authenticated and messages can be encrypted and integrity-protected. To realize the hidden SpoVNet, the Base Communication has to interact with the Security Component of each of its nodes to find the right node. The Security Component knows the SpoVNetID and secret and checks if the packet proves that the other knows it as well.

The network is protected from unwanted access on the basis of the hidden visibility. A passive attacker can not obtain the knowledge of the SpoVNetID and secret, but it needs them to interact and join. Once an attacker is in the network, it can easily add additional attacker nodes and run other attacks. Social engineering and side-channel attacks (attacker can hear human communication) are most dangerous.

Since the example scenario is only for short-lived spontaneous networks, the security may be sufficient and the attacker may join too late to profit.

4.3 Gaming Application

In this example, we describe a SpoVNet for a gaming application. The SpoVNet is closed and decentralized, but also capability-restricted. The users in this example are a – potentially large – group of players of a popular online game. The game is not open to everyone and access control is enforced. More importantly, all participating nodes must have an acceptable roundtrip-time (this is the *capability*).

The initiator is the player who initially started the game. He has specified a policy (possibly with the game's GUI) that defines the rules to join the game.

Access Control: closed The initiator wishes to restrict access to the game to persons he 'knows' – 'friends' – and transitively to such persons that they 'know'. Transitivity is limited to three hops, i.e. a player must be (at most) a friend-of-a-friend-of-a-friend-of-a-friend. To this end, the initiator has defined a policy that public keys must be used for authentication. The degree of transitivity is determined by verifying signatures chains (e. g. as in PGP/GPG). If a signature chain with less than three hops can be found for the new would-be player, he will be admitted. Otherwise, this new player will be rejected (and must first obtain such a signature). Being able to provide a ‘good’ signature chain alone, however, will not enable a new player to participate. The policy also requires that each node must have an acceptable latency (ping time) of less than 150ms. This will be checked by CLIO/UNISONO every time a new node wishes to join.

Administrative Control: decentralized The initiator does not operate as a Trusted Third Party. However, he has specified in his policy that only he and two other nodes may make additions to the policy (two friends he knows really well). The two other nodes cannot completely modify the policy, e. g. by replacing the Initiator as the trust anchor.

Visibility: visible As the initiator usually likes to play the game in his free time and at home, there is no need to hide the SpoVNet.

4.4 News Syndication

In recent years, the RSS based publication and aggregation of news stories has gained immense popularity. In such a scenario, consumers subscribe to certain categories of news (e. g sports, business, etc.) and producers publish stories in the form of RSS feeds. The published stories are then delivered by the infrastructure to consumers with matching interests. Internet-wide news dissemination imposes certain security requirements on the infrastructure, e. g., only authentic producers should publish news stories and only authorized consumers should receive them. The *Event Service* developed as a part of SpoVNet provides an ideal communication paradigm for such a service[2].

Access control: closed The set of allowed publications (from producers) and/or subscriptions (from consumers) depend on certain credentials. Therefore, a new publisher should provide a proof for his identity and his entitlement to participation, before publishing news stories. Similarly access control should determine if a new consumer is allowed to subscribe.

Administrative control: centralized with delegation Many-to-many communication in an event service mandates a scalable access control mechanism, however the anonymity and loose coupling between the publishers and consumers as well as dynamically changing subscriptions make it difficult to use a completely decentralized administrative control. Therefore, the centralized scenario with delegation is desirable. The initiator of the event service can delegate the trust to other trustworthy nodes. These trustworthy nodes can help in balancing network load by forming a network of brokers. Brokers are responsible to enforce access control policies as well as route information content from the publishers to the relevant consumers.

Visibility: visible News syndication is a public and unrestricted service. Every consumer should be able to find the SpoVNet Instance and join it by providing the necessary credentials.

Another issue related to the security in the Event Service is the confidentiality of the published information. In a news syndication service, this confidentiality may be achieved by encrypting the confidential part of the information content so that it is not visible to the brokers.

[2]All current realizations of the RSS protocol periodically poll the source irrespectively of whether there are any new updates and hence waste network resources. Alternatively a more complex push based model like that of the Event Service is more efficient and future implementations will probably shift to it.

5. Threat Models

Security can only be discussed in the context of threat models. These models describe the capabilities of attackers. We use them to address two aspects:

First, evaluating the security of cryptographic protocols. A prominent (albeit simple) example of such a protocol is a challenge-response protocol to verify a node's ownership of its cryptographic ID (which is a self-certifying identifier). We use the Dolev-Yao model for this purpose.

The second aspect is to accurately describe the dynamics within a SpoVNet Instance. We need models that describe the behavior and capabilities of malicious nodes within a SpoVNet Instance, and the impact they can have on the network. The Dolev-Yao model would be too strong here. Hence, we use models that are based on Byzantine attackers.

5.1 Dolev-Yao Model for Protocol Evaluation

We use the Dolev-Yao model [DoYa83] to formally evaluate the cryptographic protocols that we define. First proposed by D. Dolev and A. Yao in 1983, this model is generally regarded to give a formal definition of the strongest possible attacker. It is often described as 'the network is the attacker', thus implying that every channel in the network is controlled by the attacker. In particular, the attacker may do the following.

Eavesdropping The attacker may eavesdrop on any communication in the network. We also assume that he has recorded all prior communication between any two nodes. Thus, if security goals like confidentiality are to be ensured, a protocol must take care that the respective communication is encrypted and the attacker cannot derive the keys used for the encryption.

Message Insertion We assume the attacker knows the protocol specifications. He can thus create and insert new messages at will. A protocol must ensure that such messages do not lead to protocol compromise.

Message Delay and Deletion The attacker may delay any message in the network, or even delete it. A protocol must ensure that this cannot lead to the compromise of security goals.

Message Modification The attacker can modify any intercepted messages. A protocol must thus ensure that a receiver can detect such tampering.

Replay and Out-of-order Messages The attacker can replay any previously sent messages, and he can also preplay such messages (i.e. forward them out of order). A protocol must thus ensure that such re-ordering and replaying can be detected and does not lead to compromise.

Attempt at Impersonation The attacker may attempt to act under the identity of another principal. A protocol must ensure that the sender of a message can be clearly identified.

Old Session Keys We assume that the attacker is in possession of session keys from older protocol sessions. A protocol must thus take care that such session keys are not used in a manner that leads to protocol goals being compromised.

Disabling the network The attacker may at any time flood the network with messages, or partially or completely disable it (‘cut the wire’). This is a capability against which a protocol cannot take measures. Protocols can only be designed to raise the barriers for the attacker as high as possible (e. g. make Denial-of-Service attacks as difficult as possible). However, for our purposes, this capability plays little or no role in the evaluation of a protocol.

Note that the attacker is a legitimate user of the network (i.e. he may act under his own identity). The attacker is, however, bounded by the strength of the cryptographic primitives, encryption and signatures. He cannot encrypt or decrypt messages without knowledge of the corresponding key and he cannot forge signatures or message authentication codes without knowledge of the corresponding key. This effectively results in secure channels on which the attacker cannot modify messages without such tampering being detected. It does not prevent him from suppressing such messages and interacting with the normal protocol flow.

When we evaluate the security of SpoVNet protocols, we usually do this against the background of the Dolev-Yao model. The rationale is as follows. First, it is one of the standard models in the security community, and is well understood. Second, it is reasonable to design a protocol that is secure against the strongest possible attacker because in many cases we cannot be entirely sure of the capabilities of an attacker. There are only few cases where we can be sure that certain options are *not* available to the attacker. Third, current model checkers for cryptographic protocols, like AVISPA [AVI07] or Scyther [Crem06] use the Dolev-Yao model internally. Model Checkers have become an important method for evaluating the security of protocols and finding attack vectors.

It is important to observe here that the Dolev-Yao model is a model for the formal analysis of a given system. It restricts the attacker to attack vectors within the system. Therefore, it does not cover attacks outside the system, like social engineering (e. g. users giving away passwords) or errors in an implementation that lead to the compromise of secret information.

5.2 Describing Network Dynamics with Byzantine Attackers

Although the Dolev-Yao model is useful for protocol evaluation, it is rather unlikely in practice that an attacker has the full potential of a Dolev-Yao intruder – at least, at the outset. It is more likely that only very few nodes are under the control of an attacker, and that attacks will be staged from this position. This leads to the question how a network is going to be affected, particularly how it evolves if the attacker brings more nodes under his direct control. The Dolev-Yao model is too strong here as it is not concerned with such issues: the attacker can always disrupt network functionality anyway. Thus, we need to relax the assumptions and assume a weaker attacker when we discuss the ‘dynamics’ of SpoVNet Instances and the impact of a few compromised nodes.

Our choice is to model this with so-called Byzantine Attacks. The attacker in this model is derived from the adversaries in the Byzantine Generals Problem. Described by Lamport et al. [LaSP82], this problem describes a setting where several entities need to reach an agreement in the presence of malicious entities, which they do not know to act maliciously.

Applied to networks, the attacker is assumed to have partial or full control over a number of entities. He may behave arbitrarily to disrupt the network, acquire secret knowledge or otherwise gain an unfair advantage. In contrast to a Dolev-Yao attacker, however, he does not have control over all communication channels. Rather, such control depends on the capabilities of the nodes that he has under control, and the extent to which he can exercise this control. For example, if a node is under full control, the attacker can use the identity and keying material of this node, and he can also insert or delete messages from the network, as described above. His control does not extend to the communication channels of other nodes, however, and he may not be able to delete or modify a message between two other entities in the network.

Clearly, such models are more difficult to describe formally, and they span a wide range of possible attacker behaviour. We do not intend to give a precise and formal description here but restrict ourselves to categories that are likely to apply in the context of SpoVNet.

Nodes We assume that there are two types of nodes: honest ones and adversaries. Honest nodes follow the protocols without deviation. They are not compromised. Without loss of generality, we can assume that they also have a correct internal state: faulty nodes, which may play into the hands of an adversary, can be modeled as (partially) compromised nodes.

Corruption of nodes Depending on the use case, an attacker may have the knowledge which node it would be opportune to attack next. In other use cases, the attacker may not have this knowledge. We always assume that the attacker knows the SpoVNet protocols. First, because the SpoVNet software is open source; second because inside attackers (see below) will always have this knowledge. An *adaptive attacker* will adapt to SpoVNet protocols and security measures. He can change his strategy depending on the security measures that honest nodes take. An example would be an attacker who has gained some information and uses it to target 'weakly' protected nodes. This is in contrast to an attacker who, e. g. , is eavesdropping and will not change his strategy, like a scripted attack.

Corrupted nodes Corrupted nodes can be co-ordinated by the attacker to co-operate in attacks. In particular, they can exchange data by using out-of-band communication. We assume that at any time only a subset of nodes is corrupt. This set is not static, it may change as the attacker gains control over more nodes (or loses it again as, e.g., administrators bring a node back under their own control again). Note that it has been shown in [LaSP82] that agreement (on some set of information) between multiple parties, and for any kind of protocol, becomes impossible once the number of corrupted and maliciously acting nodes reaches one third of all nodes in the network.

Passive and active attackers We distinguish between passive and active attackers. These terms describe the capabilities of an attacker that has already been successful in compromising one or more nodes.

A *passive attacker* is said to have obtained the complete (secret) information held by a corrupted node. However, the node will still correctly execute the SpoVNet protocols. The attacker may now be able to eavesdrop on and interfere with conversations that are protected with the compromised secret information. In this situation, one must investigate how this can lead to further nodes being compromised.

An *active attacker* is an attacker who has obtained complete control of a corrupted node and can use both the information obtained plus the node's SpoVNet functionality in order to further attack the network. Cryptographically, this attacker is not stronger than the passive attacker because he is in possession of the same knowledge. However, as he has the control over a node, he can truly act in the name of that node. This corresponds to a Dolev-Yao attacker whose range of action is limited to the channels originating from the local node. He can insert, modify, delete messages etc., but he does not have control over other communication channels between other nodes.

The attacker may use the thus corrupted nodes to stage certain attacks. A Denial-of-Service attack, for example, could now be staged from a node within a SpoVNet, possibly selectively targetting other important nodes in the same SpoVNet.

Outside and inside attackers We say that an outside attacker is an attacker who is not a part of the SpoVNet Base Overlay and does not belong to the SpoVNet Instance. However, the attacker may still know the SpoVNet protocols and will attempt to compromise nodes once he knows about their existence. Inside attackers, on the other hand, participate in the SpoVNet Base Overlay.

Limitations As in the Dolev-Yao model, we generally assume that the attacker cannot break cryptographic primitives efficiently, i.e. within a time span that would allow him to achieve his goals. This is argued to be due to computational limitations.

These categories need not be distinct; attackers may also have the combined possibilities from several categories. We will precisely state the capabilities of the attacker with respect to these categories whenever we evaluate the security of a SpoVNet concept.

6. Requirements for the Security Component

Due to the integrated nature of the Security Component, requirements are two-fold: First, the Security Component must be able to transparently support certain security goals (e. g. authentication). At the same time, it must also enable SpoVNet applications and services to design their own security measures. To this end, it must provide basic cryptographic functionality (e. g. encryption).

In the following, we identify these requirements. We begin with the latter aspect, the cryptographic functionality.

6.1 Cryptographic Primitives and Operations

Applications and services must be able to design their own security concepts without having to supply their own implementation of cryptographic operations. The Security Component thus incorporates a cryptographic library that can be accessed by all applications and services. The library provides cryptographic primitives, which can be used to achieve one or more of the classic goals of cryptography.

Confidentiality Intruders must not be able to learn the content of confidentiality-protected messages.

Authenticity The receiver of a message must be able to clearly identify the sender of the message.

Integrity An intruder must not be able to change the content of an integrity-protected message without the receiver noticing that the message has been altered on the way.

The library also supports **data origin authentication**, which is a weak form of non-repudiation. We understand this to be a receiver's capability to verify that certain data have been signed with the respective sender's key (without further assertions being assumed)[1].

We can now identify the following cryptographic operations that need to be provided by SEC:

Symmetric encryption The SEC provides a set of algorithms where sender and receive share a key, which is used for both encryption and decryption. Examples are AES and 3DES. The SEC will focus on the current state-of-the-art algorithm, which is currently AES-128.

[1]It is important to know that the term non-repudiation is used differently in literature. In cryptography, it most commonly refers to the property that a sender cannot deny that he has created a certain message. The assumption is that the sender's keys cannot be compromised. The meaning of the term is different when used in the context of network protocols. The assumption there is that keys can become compromised, and non-repudiation needs to take this into account. Therefore, non-repudiation becomes much harder to achieve in such systems, and includes the use of timestamps. In the context of SpoVNet, however, non-repudiation is only provided as it is understood in cryptography. If keys become compromised, this is an issue that must be addressed by the respective application or service.

Asymmetric encryption: The SEC must provide algorithms where sender and receiver have different keys for encryption and decryption. Examples are RSA, ElGamal, or Elliptic Curve Cryptography (ECC). The SEC will focus on one of the standard algorithms, which are either RSA or ElGamal.

Message Authentication Codes (MACs) The SEC must provide MACs as means of integrity protection for messages. MACs are generally fast to compute. However, since they rely on symmetric cryptography, a third party cannot decide whether a message has been created by the sender or the receiver, or anyone else in possession of the key. HMAC, which combines hashing algorithms with MAC, is a well-known standard.

Digital signatures The SEC must provide means to create digital signatures. These use asymmetric cryptography for integrity protection, and therefore also achieve the goal of data origin authentication. Digital signatures are expensive, and therefore usually combined with a hashing scheme. Popular examples are RSA signatures and the Digital Signature Standard (DSA). Digital signatures may be needed by certain applications and services. CLIO is expected to use digital signatures, for example.

Cryptographic hashing Hash functions are reduction functions that map a value of arbitrary length to a value in the co-domain with a fixed length that is in the majority of cases shorter than the original length. Therefore hash functions are generally not injective. Cryptographically secure hash functions are hash functions that distribute the hash values uniformly over the co-domain. Moreover, they guarantee that the complexity of finding two values that map to the same hash value is about the same as that of a brute-force search. It is still unknown whether secure hash functions exist. However, there are several proposals that are generally considered to approximate them well, e. g., SHA1.

6.2 Security Goals

In addition to basic cryptographic operations, the Security Component must also support a number of security concepts. We have identified the following security goals that need to be supported in SpoVNet:

Authentication Principals must be able to ensure that they interact with other legitimate principals. To this end, the Security Component provides mechanisms to ensure the authenticity of a certain principal's identity. Authentication in SpoVNet is of particular interest because nodes in the Base Overlay use self-certifying identifiers. This makes it possible to establish the ownership of a certain identifier, and therefore a public key, through a simple challenge-response protocol.

In certain scenarios authentication should be done mutually. The Security Component will also notify the requesting node of the authentication mechanisms being used in order for the node to be able to decide whether the strength of the authentication is sufficient for its purposes. This is also helpful in avoiding downgrading attacks, where the attacker tempts to lure the victim into using a weaker cipher that he can break.

It should be noted that authentication is a term that must be clearly defined in order for security evaluations to provide meaningful insight. A well-established definition is *Authentication as Injective Agreement* [Lowe97]. Whenever we analyse authentication in the context of SpoVNet-specific protocols, we will clearly indicate which definition of authentication we use.

Key Establishment and Key Agreement Principals must be able to establish cryptographic keys between each other. Key Establishment is the general term for this operation. Key Agreement means that the key must be derived through an interaction of both parties, in which both parties provide material that is used in key creation. Note that authentication and key establishment are usually goals that are pursued together, as there are only very few use cases where authentication makes sense without encrypting the ensuing communication. Since the authenticity of a public key can be easily established with the help of the self-certifying identifiers (see above), it is also easy to add a key exchange after authentication. One possibility would be a Diffie-Hellman Key Exchange, which has the added benefit of providing *Perfect Forward Security*, i. e., a communication that was protected with such a derived key remains confidential even when the long-term keys become compromised. Therefore an attacker cannot decrypt recorded communication sessions.

Encrypted Connections and Payloads Principals must be able to request encrypted connections. This is usually achieved through the Base Communication by transparently establishing secured connections (e. g., TLS), but keys are provided by the Security Component. Where the underlay does not provide such mechanisms, the Security Component can be used by the Base Communication to transparently encrypt/decrypt messages on the insecure channels. Finally, applications and services can also directly access the Security Component and request that messages are encrypted with a given key.

Authorization and Access Control The SEC provides mechanisms that enable SpoVNet to provide authorization and access control. These can be realized via policies, access control lists, etc. The Base Communication can use these mechanisms to decide whether a node is allowed to join the SpoVNet or not. The Security Component can also determine which components and which users may access which data. This is important for CLIO, for example, as one SpoVNet Instance must not be able to access data of another SpoVNet instance on the same host. Similarly, the Security Component stores user data safely and makes it accessible only to the legitimate user.

It should be noted that there are other security goals, like reliability or availability. These are not directly supported by the Security Component as they are really properties of a distributed system. SpoVNet applications or services can make use of the functionality of the Security Component to achieve such goals.

7. Architecture of the Security Component

In this chapter, we will describe the architecture that we have designed for our Security Component. The architecture supports the security goals and requirements that we have laid out in Chapter 2 and Chatpter 6. We introduce our terminology and discuss sub-components and modules.

7.1 Terminology

We follow a modular approach in the architecture of the Security Component. Several definitions are necessary for the architectural design:

Subcomponent: This term indicates a logical grouping of functionality. An example would be the Cryptographic Engine, which provides access to a number of cryptopgraphic operations (see below).

Module: Each subcomponent consists of one or more modules. A module is a grouping of mechanisms that are available to provide the functionality. E. g., the Cryptographic Engine contains modules for symmetric encryption.

Mechanism: A mechanism is a concrete implementation of a specific functionality in a module. Mechanisms can therefore be interchangeable. For example, mechanisms for symmetric encryption could be algorithms like 3DES or AES.

Several or even all subcomponents might be involved when a security goal is to be achieved. For example, an authentication protocol may have to draw on the security storage to store its state; on the cryptographic engine to create and verify signatures or encrypt messages; and on policy management to determine whether a given node can be authenticated with the credentials it provides or not.

7.2 Subcomponents

Four subcomponents exist in the Security Component, as shown in Figure 7.1.

Indispensibe for security are cryptographic operations and therefore the security component needs a **Cryptographic Engine**. This includes modules to provide encryption, decryption, message authentication codes, signatures, etc. The Cryptographic Engine does not to implement cryptographic mechanisms by itself. Rather, it provides an interface to operations based on well-established cryptographic libraries.

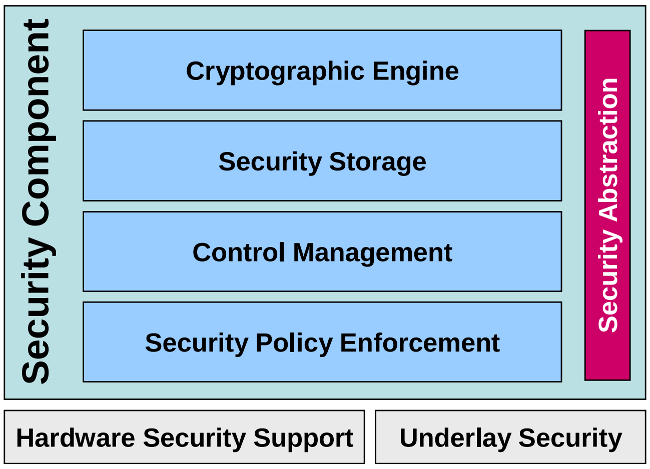


Figure 7.1: Subcomponents of the Security Component.

Security protocols and other methods need to hold state. A simple form of state is the knowledge of a key or certificate. For protocols, this state represents the current step and the information acquired. So, the Security Component needs to include a **Security Storage** for the storage of security-related state. If available, hardware-side security components (e. g., Trusted Platform Modules) can also be used.

Control Management is important for the embedding of security in SpoVNet. A normal peer may only know the policies of the SpoVNet. But a trusted node, primarily the initiator, has to operate as a TTP in the SpoVNet, e. g., for join and bootstrap operations. It needs to know who is allowed to enter and requires means to check the identity. As a result, it may issue certificates once a node authenticated itself. To be more resilient, TTP operation can be delegated, so the delegation needs to be defined and the delegate needs to be able prove this fact. Furthermore, TTP operation may also include the management of trust and evaluate reports on fraud and attacks. In decentral networks, TTP operation is fully distributed and basic functionality for joint decisions is needed in all nodes.

Besides the management tasks and all the basic operations, the security component also needs **Security Policy Enforcement**. Of course, the component is limited to directly enforce policies within its own operations. However, it can provide decisions for other components based on the policies and information it received. This includes access control decisions for other SpoVNet components.

7.3 Modules

The subcomponents introduced in Section 7.2 represent logical units that make up the Security Component. We now refine this presentation and introduce *modules* as self-contained functional blocks. Figure 7.2 shows the associated modules for each subcomponent.

Cryptographic Engine The Cryptographic Engine contains modules for single cryptographic primitives like ciphers and message authentication codes, as well as more complex cryptographic protocols like key agreement schemes. Further more, SpoVNet-specific NodeID primitives, protocols, and protocol adaptations are provided in the Cryptographic Engine. NodeID protocol adaptations represent, e. g., new authentication schemes for existing protocols like TLS.

Security Storage Security related state and context information is stored in the Security Storage. We can differentiate between three modules in the Security Storage:

- Security Context and State Information

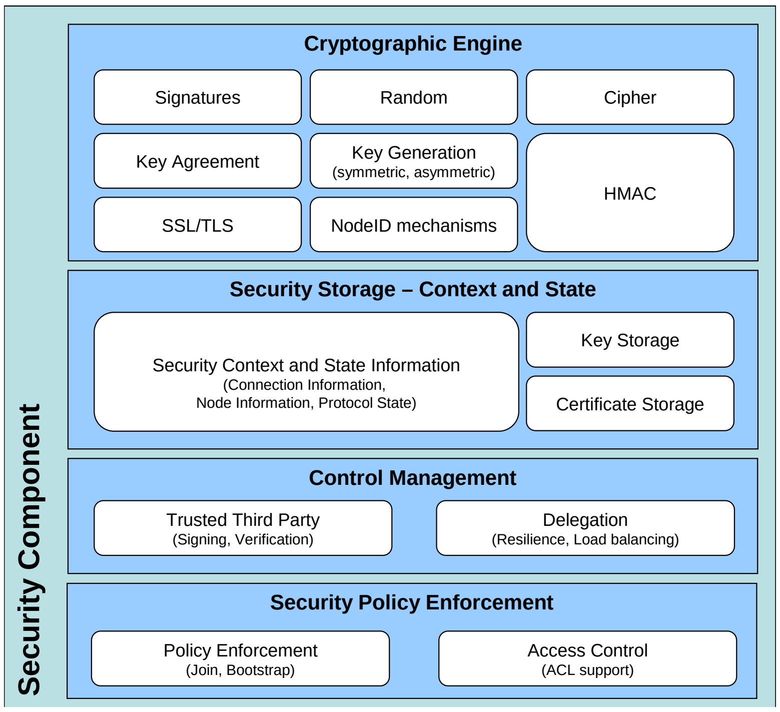

Security Component
Cryptographic Engine
Signatures
Random
Cipher
Key Agreement
Key Generation (symmetric, asymmetric)
HMAC
SSL/TLS
NodeID mechanisms
Security Storage – Context and State
Security Context and State Information (Connection Information, Node Information, Protocol State)
Key Storage
Certificate Storage
Control Management
Trusted Third Party (Signing, Verification)
Delegation (Resilience, Load balancing)
Security Policy Enforcement
Policy Enforcement (Join, Bootstrap)
Access Control (ACL support)

Figure 7.2: Subcomponents and their modules in the Security Component.

- Key Storage
- Certificate Storage

The module for Security Context and State Information stores relevant information about connections, nodes, and protocol state. One example are security associations for a connection with security properties like integrity or encryption.

Control Management A further subcomponent in the Security Component is Control Management. This includes a Trusted Third Party module for signing and verification scenarios. The Trusted Third Party is, in the majority of cases, the SpoVNet Initiator. The purpose of the Delegation module is to transfer trust and policy information to other SpoVNet nodes. This way, e. g., policy enforcement can be delegated to other SpoVNet nodes and therefore resilience and load balancing achieved.

Security Policy Enforcement The SpoVNet Initiator defines the security policy for the SpoVNet Instance. The policy is enforced by the subcomponent for Security Policy Enforcement. This has implications on, e. g., join and bootstrap operations for nodes.

8. Subproject Work

This chapter will detail on the TP-specific work that has been performed in order to extend the STF work. TP-specific security work has been done in the context of TP1 SpoVNet Base 8.1, TP1 MCPO 8.2, TP2 8.3, and TP3 8.4.

8.1 TP1 - SpoVNet Base

In the following we detail on the security mechanisms in the SpoVNet Base, mainly from a conceptual perspective. This includes how cryptographic identifiers are used to inhibit eavesdropping, ensure integrity and authenticity of messages sent over SpoVNet links. This is done by concentrating on how existing protocols can be used or adapted for used with the SpoVNet base. As an additional feature, a mechanism is introduced to obfuscate SpoVNet traffic in a way that it is difficult to detect specific traffic patterns in order to discover a node that is running the SpoVNet software. Parts of the concepts presented here have been developed in [Toll09, BaMi07].

8.1.1 Cryptographic identifiers

Cryptographic identifiers have been proven to be a very easy way to overcome difficulties in typical end-to-end security scenarios. They allow the verification of the ownership of a node identifier, signing and verification of messages. Furthermore key-exchange mechanisms can exploit cryptographic identifiers to create secure end-to-end links. In order to take advantage of cryptographic identifiers, a node must generate a pair of keys, a private- and public-key respectively. To generate a pair of keys we consider RSA, DSA, and ECDSA as valid algorithms, since they are widely used and tested. However, different algorithms can be used in the future. Performance issues with those algorithms are discussed in Table 8.1. When a node chooses a pair of keys, we generate the cryptographic identifiers as follows: first, an arbitrary value $modifier$ is chosen. This modifier enables the node to change its identifier value without changing the associated public key. In the next step public-key and modifier are concatenated and hashed with a cryptographic hash function (e.g., SHA1) which should result in a bit-string long enough to be used in the SpoVNet Base Overlay. If the hash is longer than needed, only the n leftmost bits are used. This scheme follows the NIST recommendations. Figure 8.1 shows the exemplary generation of a cryptographic node identifier (*NodeID*) and its storage on a UNIX-based system. In order to prove the identity of a node, an additional data structure is needed which includes the type of the cryptographic algorithm, the modifier and the public-key of a node. This structure is called *CryptoID*. To verify the ownership of an identifier a node must provide its *CryptoID* to the

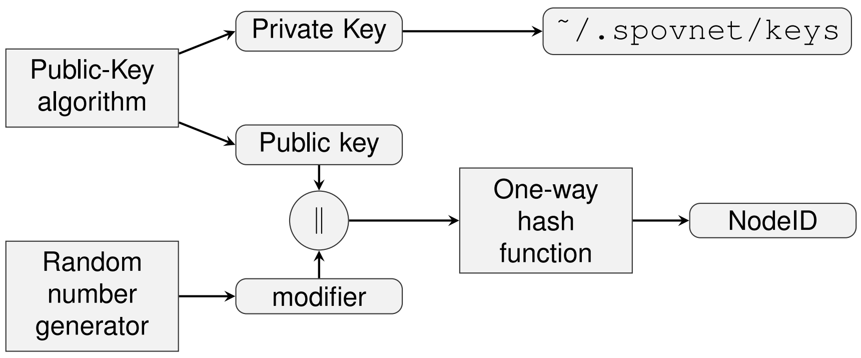
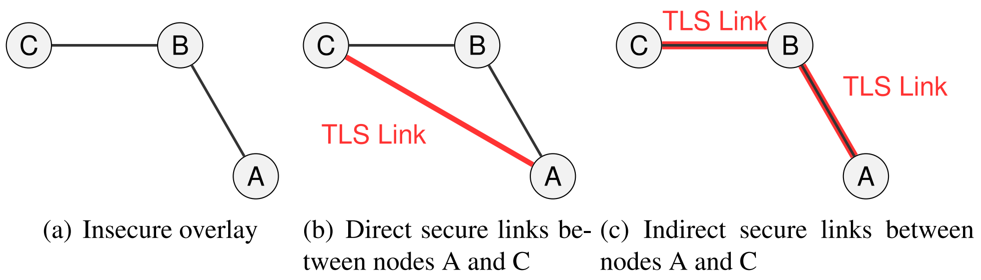


Figure 8.1: Generation of cryptographic identifiers



(a) Insecure overlay (b) Direct secure links between nodes A and C (c) Indirect secure links between nodes A and C

Figure 8.2: Topology of a SpoVNet network

verifying node signed with the according private key and algorithm. A recipient is now able to verify the signature and validate the NodeID by checking the calculations done by the author. Signing and verifying the authenticity and integrity of arbitrary messages is possible in a similar way when the CryptoID of a node is known.

8.1.2 Link security

The SpoVNet base provides secure end-to-end links to SpoVNet applications. Using the cryptographic identifiers together with the CryptoID structure, the SpoVNet base can exchange symmetric keys to secure links between arbitrary nodes. Due to the objective of SpoVNet to reuse as many existing protocols as possible, first, the option of reusing protocols like IPsec, TLS, and DTLS is evaluated. Second, a solution to provide secure links for SpoVNet is presented. When considering existing solutions, TLS or IPSec would be a desired protocol to replace or extend the transport mechanisms in SpoVNet. However, SpoVNet has one significant feature: it allows relay paths the be created over different underlay protocols, i.e., Bluetooth, or mixed IPv4 and IPv6 networks. In this situation existing protocols cannot be used in SpoVNet. Consider the scenario shown in Figure 8.2: node A and C can reach each other directly and can therefore establish a TLS session. However, if A and C are using different protocols and could not communicate directly, a relay path would be necessary which would result in two concatenated TLS links with node B in the middle. To route messages, node B would likely have to decrypt the data sent over this relayed link and would become a potential eavesdropper. This makes the transparent use of existing protocols complicated and is the reason why SpoVNet pursues its own solution that borrows concepts from existing protocols to provide link security. In order to develop a solution for SpoVNet link security, key-exchange protocols that are currently in use were analysed whether they are suitable for SpoVNet. In addition to the overall performance of the protocols, two further criteria were taken under consideration: the overall computational cost of each key exchange to support devices like handhelds, and the need for a reliable transmission channel. Further analysis in [Toll09] shows that using JFKi for key-exchange in conjunction with the DTLS payload transfer protocol is best suited for SpoVNet links. To use those protocols, the JFKi protocol was adapted for the use with SpoVNet by replacing the IP-Addresses

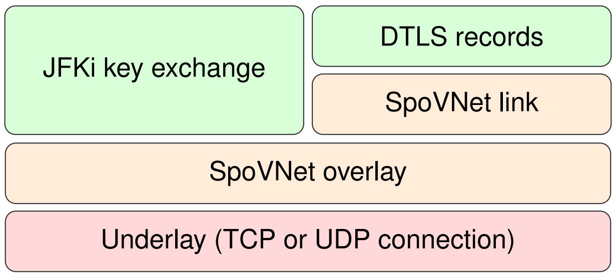


Figure 8.3: Architecture for the link security

with node identifiers and certificates with the CryptoID structure. After exchanging symmetric keys with JFKi the DTLS record protocol is used to protect the data transfer. The fact that the open-source implementation OpenSSL implements DTLS since Version 0.9.8 in 2005 and is actively maintained this makes it less likely to introduce additional security flaws with our protocol.

8.1.3 Security for Key-based Routing (DHT)

Since the base overlay uses a key-based routing protocol to map node identities to their respective underlay addresses, the question arises whether the overlay is resistant against adversarial nodes. Concerning our attacker model in Chapter 5 there are a lot of possibilities to inhibit communication or degrade the performance inside the key-based routing overlay. Basically there are two classes of attacks possible in such overlay networks: (1) Eclipse-, and Sybil-attacks where adversaries try to choose their logical position inside the overlay in a way they can harm certain nodes or the network or by flooding the network. (2) attacks that try to use, modify, or simply drop signaling traffic. While (1) can be inhibited by using crypto-puzzles, (2) have to be addressed with typical mechanisms from fault-tolerant systems: introducing redundancy. This allows finding paths inside the overlay network that are free of adversarial nodes. An example of such a secure key-based routing has been proposed in [BaMi07] which can be used in SpoVNet as a base overlay. It extends the Kademlia key-based routing protocol to provide security by using a loosely coupled routing scheme and impedes Sybil- and Eclipse-attacks. Since it also uses cryptographic identifiers, it can be easily adapted for use in SpoVNet if needed.

8.2 TP1 - Multicast/Multipeer Overlay

In the following we detail on the conceptual security mechanisms for the MCPO service that differ heavily from the end-to-end security approach the SpoVNet Base provides (see Section 8.1), as group security mechanisms are required. In the following we detail on security mechanisms that have been selected and integrated into the MCPO service.

The Multicast/Multipeer Overlay (MCPO) is an application-layer multicast service, based on the NICE protocol [BaBK02]. It runs over the SpoVNet Base and therewith leverages security properties provided by the SpoVNet Base. As MCPO provides group communication mechanisms, group security mechanisms must be deployed. In the following we give an overview on the security mechanisms for MCPO.

Group security mechanisms can be divided into the three parts *confidentiality*, *authenticity*, and *authorization* that we detail on in the following sections.

8.2.1 Confidentiality

Confidentiality in group communication systems uses *group keys* to provide confidentiality of messages inside a specific group. Mechanisms developed for end-to-end confidentiality cannot be deployed. As the SpoVNet Base provides end-to-end confidentiality, these mechanisms are not applicable, but can be leveraged for other parts of the MCPO security system. An example here is initial

	DTLS	HIP	IKEv2	JFKi	JFKr
Round trips	4 or 6	4	4 or 6	4	4
Number of private key operations (initiator)	2	1	1	1	1
Number of private key operations (responder)	1	1	1	1	1
Number of public key operations (initiator)	2	2	1	2	1
Number of public key operations (responder)	1	2	1	1	1
PFS	Ephemeral RSA or Diffie-Hellman	Diffie-Hellman[a]	Diffie-Hellman	Diffie-Hellman[a]	Diffie-Hellman[a]
Computation DoS	Reachability test	Cryptographic puzzle	Reachability test with additional round trip	Reachability test	Reachability test
Memory DoS	Reachability test	Pre-computed first message	Reachability test	Cookie	Cookie
Privacy of initiator's identity	no	no	yes (against passive attack only)	yes	yes (against passive attack only)
Privacy of responder's identity	no	no	yes	no	yes
Keys renewal	yes	yes	yes	no	no
Repudiability	yes	no	no	no	yes
Formal proof	TLS ([HSDD$^+$05]). No proof for DTLS available	Authenticity, confidentiality ([CaKr02])	Authenticity, confidentiality ([CaKr02])	Authenticity, confidentiality and DOS resistance ([AbBF07])	Authenticity, confidentiality and DOS resistance ([AbBF07])

Table 8.1: Summary of key exchange protocols properties [Toll09]

[a]Memory DoS can only be prevented with an interval forward secrecy

authorization during the group joining process where communication between two parties – the joining node and the rendezvous point – is required. In such cases end-to-end confidentiality is required and can directly benefit from the security properties provided by the SpoVNet Base.

We define a group key GK that is used for symmetric encryption of messages inside the group. This GK is calculated by a group controller GC and distributed to the members of the group in a secure way. Refreshing the keying material of the GK and distributing the key to the group is the process of rekeying. To provide forward and backward secrecy, rekeying must be performed whenever a node joins or leaves the group that is secured using the GK. Whether all or only part of the nodes need to be rekeyed depends on the scheme. Schemes where only one GK is employed need rekeying whenever any node joins or leaves. Other schemes where multiple GKs are employed and used in a cluster-based scheme only need to rekey the respective cluster, but have more overhead as messages need to be re-encrypted while traversing clusters.

Complexity in group security systems is measured in distribution cost of a new GK. In a naive system the GC has pairwise keys with each node and can securely transmit information to each node. For rekeying, the GC calculates a new random GK and sends this key to each node using the pairwise key. Therefore, the GC has to make n cryptographic operations and n sending operations for rekeying. This naive scheme has therewith a complexity of $\mathcal{O}(n)$. Reducing this complexity is possible using a $keytree$, as first proposed in [WoGL98]. In such a scheme, the nodes are leafs of a tree whose inner nodes are keys. Each node knows all keys from its leaf towards the root. The root node is the GK. In case a node leaves the system, all keys from the node's leaf including the root become invalid and must be refreshed. Now in this scheme, the GC no longer needs to provide each node with the new GK individually but can rather use the highest common leaf of a subtree to rekey parts of the nodes together. Using such a scheme the rekeying complexity can be reduced from $\mathcal{O}(n)$ to $\mathcal{O}(log_2(n))$.

For group security in MCPO we use a two-tiered system: (1) to provide group keys inside each cluster, and (2) for inter-cluster communication. Each MCPO cluster is secured with a cluster-specific GK. Inside a cluster, the MCPO cluster leader acts as group controller to provide each node with the cluster-specific GK. A Logical-Key Hierarchy scheme is used in each cluster, independently of the other MCPO clusters. Respectively, in each higher layer one Logical-Key Hierarchy is employed to secure this respective cluster. Each cluster leader respectively manages its cluster to make sure that forward- and backward-secrecy is guaranteed. The MCPO overall cluster leader then manages one cluster, too.

For data dissemination, we employ a scheme similar to [Mitt97]. As each cluster operates its own GK, data that is distributed through the system must be de- and encrypted at each point where data moves between clusters. This is possible as the cluster leaders that forward the data always have identities in two or more layers. Therefore, cluster leaders can de- and encrypt traversing multicast data.

The scheme has the benefit that it prevents the *one-affects-all* problem, as normally, every single join and leave operation triggers a complete rekeying of the system. As each cluster operates its own GK in our scheme, only the respective cluster(s) need rekeying, therewith reducing the rekeying overhead. A disadvantage of the scheme is the de- and encryptions necessary to forward the multicast data. We argue that this overhead highly depends on the application type and the capabilities of the leader nodes. Symmetric cryptography can be implemented efficiently in hardware modules. We see the reduced instability that is triggered by joins and leaves in a system where only one GK is used as highly undesirable and therewith opt for a scheme with multiple GKs, especially in the face of lifetime-distributions and inter-arrival times of peer-to-peer systems [StRe06].

8.2.2 Authenticity

In the following we look into authenticity of message sources and authenticity of MCPO groups. For source authenticity we present an adapted version of the TESLA scheme in Section 8.2.2.1. This scheme provides the security that data has been sent by the actual node that claims to be the sender. To prevent spoofing of group controller roles, we present a scheme for group authentication using cryptographic identifiers in Section 8.2.2.2.

8.2.2.1 Source Authenticity

Authenticity provides guarantees that the stated sender of a message is the actual sender. In end-to-end systems this property can be achieved through (1) integrity protection using a shared secret key like in HMAC schemes, or (2) using asymmetric cryptography through signatures. As signatures have high computational cost and large size, they are not useful for continuous message streams like video-streaming or gaming protocols. Therefore, shared symmetric keys are used in end-to-end systems to provide source authenticity. The security of this scheme stems from the fact that the receiver of a message can be sure that either he or the sender created the message through the secret shared key. As he knows that he did not create the message, source authenticity is guaranteed.

Such a scheme is not applicable for group communication systems to provide source authentication. Sharing a secret key in the complete group only provides guarantees that the sender of a message is a member of the group, but no further source authentication means are provided. Such means are necessary, e. g. in a multicast video streaming application. In such a system the receiver of video data needs to be sure that the sender of this data is the actual sender. Exemplarily, a receiver of CNN multicast video stream wants to be sure that the data is actually sent from CNN, and not some rogue node inside the group. As group keys only provide confidentiality inside the group, special mechanisms for source authentication are necessary.

A well-known scheme for source authentication in broadcast and multicast systems is TESLA [PCST01]. Using an initial bootstrapping of the source authenticity-trust by means of asymmetric cryptography, TESLA then chains this trust further in time using pure symmetric cryptography to keep the trust up in the following messages. This way, TESLA provide good performance, as only initially signatures are used. We use the TESLA scheme in conjunction with the cryptographic node identifiers for the initial signature-based part of TESLA. As the cryptographic node identifiers are self-certifying, no public key infrastructure (PKI) is required.

8.2.2.2 Group Authenticity

Nodes can use the multicast mechanisms of the MCPO control structure to find embedded groups inside MCPO. This is done by using the multicast mechanisms and relying on the specific group controller to answer the request. The node can then contact the group controller to join the embedded group. Such group advertisements can be spoofed by other nodes that are in the MCPO structure, therewith making the joining node believe he is joining a confidential group where he is really joining a spoofed group.

To prevent such spoofing attacks, we use cryptographic identifiers for group identification. A group controller creates a cryptographic identifier using a public/private keypair. The resulting identifier is used as the group name. A group controller can now prove that he created the respective group. This way, rogue group advertisements can be detected and group controller spoofing prevented.

As the group name is the generated from the public key, the group name cannot be chosen freely. For arbitrary group name selection, this scheme can possibly be extended using Identity-based Encryption [BoFr01].

8.2.3 Authorization

To provide user-restricted embedded groups in MCPO we present an authorization scheme that is based on the mechanism presented in 8.2.2.2 and similar to the scheme presented in [CaMo02]. It uses signatures to provide security that a node is authorized to join a specific embedded group. The scheme is split into two parts: First, a node acquires an authorization token. Second, the token is used for a proof of membership to show authorization to a group during the join process of this embedded group.

Authorization Aquisition

Acquisition of an authorization token is an out-of-band mechanism that is not coupled with MCPO or SpoVNet. The token is a signature on the cryptographic identifier of the requesting node using the private key of the group cryptographic identifier, owned by the group controller.

Issuance of the authorization token is an out-of-band mechanism not defined in SpoVNet. Exemplarily, such a mechanism can be implemented using password-based authorization or by means of access-control lists. For more fine-grained access control timestamps can be included in the signature to provide time-limited authorization that can be refreshed.

Proof of Membership

After a node has detected a group by means of multicasting, he will contact the responding group controller to join the embedded group. To prove authorization to join the group, he uses the authorization signature he acquired, as detailed in the previous paragraph.

Now he can perform a proof of membership by first prooving his identity through his cryptographic identifier. After successful authentication the node will provide the token – the signature acquired in the above paragraph – to the group controller. The signature can be validated by the public key of the group – information that is accessible to the group controller. This way the node can prove that he has earlier acquired a credential that allows him to join the group.

8.3 TP2 - Cross-Layer Information for Overlays

CLIO/UNISONO is responsible for conducting cross-layer measurements in a SpoVNet. The system makes use of the Security Component, but also adds its own security mechanisms. In the following, we show the interactions between CLIO/UNISONO and the Security Component.

8.3.1 Overview of Security Issues

CLIO/UNISONO is a split component: CLIO and UNISONO are separate processes on a node. They communicate via XML-RPC. Security in CLIO/UNISONO thus focuses on two aspects.

The first is communication between CLIO/UNISONO instances. Our priority here is on securing the data exchange between two CLIO instances. We discuss this in section 8.3.2.

The second aspect is securing CLIO/UNISONO against misuse or malicious attacks by applications, services or remote nodes. CLIO/UNISONO is an open system. It is possible to induce load on a node just by sending messages. Remote orders open a further attack vector: an attacker can attempt to request a high number of measurements to increase load either on the local node or on a remote system. He can also attempt to request a high number of measurements from a large number of different remote nodes, but always with the same measurement destination. Our priority is thus to make CLIO/UNISONO as robust as possible against attack and exploit. To this end, we need to provide access control to measurement modules with respect to identifiers and requested resource usage. We also allow to define rate limits. We discuss this in sections 8.3.3 and 8.3.4.

8.3.2 Securing the Communication between Instances of CLIO/UNISONO

UNISONO instances only exchange data packets in order to conduct measurements. Control messages are sent via CLIO. Measurement packets have randomized content and do not need confidentiality and integrity protection. The UNISONO instances involved only accept packets on a certain port and only during the time indicated by the control messages. Further authentication is not used to avoid overhead. UNISONO uses timeouts to cover the case of lost control messages.

CLIO instances may communicate in two cases. First, if remote orders are exchanged. Second, where measurements need to be coordinated between the node conducting the measurement and the node being measured. Bandwidth measurements, for example, use CLIO messages to signal the start and end of the measurement.

Such communication between CLIO instances must be protected for several reasons. First, in order to implement proper access control, we must ensure that we can authenticate the sender of a message. Second, an attacker should not be able to alter the content of a message without the receiver noticing. Otherwise an attacker would be able to change the message type or measurement values, which can, e. g., lead to degradation of optimization algorithms which rely on this information. Third, it is desirable that message content remain confidential. It could, e.g., be information that could be abused for attacks (like open ports). It is also a privacy issue: invisible SpoVNets (see section 2.5.3) should not leak information. Thus, CLIO messages are protected as shown in the following.

Confidentiality

CLIO messages are encrypted using one of two methods. One is to use Ariba to request end-to-end encrypted connections. This is possible if the respective protocols are available on a node. The second method is to use the Cryptographic Engine to directly encrypt a CLIO message.

Authentication

If it were possible to forge a receiver's ID, this could be abused to redirect CLIO messages to other CLIO instances. Being able to forge the sender of a remote order would allow an attacker to redirect the result to a different node as part of a Denial-of-Service attack. CLIO uses Ariba and the Security Component to authenticate sender and receiver. As described in section 8.1.1, cryptographic IDs are used for this. Ariba delivers incoming messages to CLIO and passes the sender as a parameter. CLIO uses only this information to determine to which node a reply must be sent.

Data Integrity

CLIO does not use Ariba to provide data integrity as it requires digital signatures instead of Message Authentication Codes. This makes it possible that a third node, to whom a measurement result may be sent later, is able to verify data integrity as well. CLIO uses the Cryptographic Engine to sign each payload with the private key of the node it is running on.

8.3.3 Using Policies in UNISONO

In the following, we describe a central feature in UNISONO: the use of policies to control access to data items and protect against misuse.

Requesting data items is subject to access control. This is flexibly adjustable by a user or administrator in order to block undesired measurement requests and maintain a certain privacy. We only briefly summarize the most important concepts here.

UNISONO uses policies that define how a data item can be accessed, and by whom. Before an order is executed, it is checked against the current policy. UNISONO uses its own Policy Engine, for three reasons: First, only UNISONO is aware of which orders are currently being executed and thus can determine whether a rate limit would be kept or not. Second, policies may contain rules both in the semantics of underlay and overlay. UNISONO can maintain an agnostic position here and interprete identifiers as simple strings. Third, UNISONO runs as a process of its own and is intended to be also accessible by non-SpoVNet applications (if they are registered).

Policies can be defined to provide the following functionality:

- **(Role-based) access control:** UNISONO uses ACLs. Roles can be assigned to Node IDs, IP addresses or Service IDs. It is also possible to add these directly to an ACL, without a role.
- **Control over resource usage:** Measurements have costs, represented in UNISONO as cost values. Access control can also be defined in terms of maximal allowed resource usage.
- **Rate Limiting:** Rate limits can be assigned to data items. It is possible to distinguish between incoming and outgoing remote orders. So-called destination limits can also be set: these define the maximal rate at which measurements are allowed to any single node.

8.3.4 Protecting CLIO/UNISONO Against Misuse

In the following, we show how CLIO/UNISONO are protected against misuse.

8.3.4.1 Protecting Communication With UNISONO

Apart from the above-mentioned measurements, UNISONO instances do not communicate directly with each other. UNISONO communicates with CLIO via XML-RPC and accepts RPC calls only on the local host interface.

To protect against malicious applications running on the same host, UNISONO allows access only after authentication. CLIO must present a credential that has been pre-registered with UNISONO. The credential is stored in the Security Component's keystore, to which both UNISONO and CLIO have access.

Communication between UNISONO and CLIO is optionally encrypted. To this end, CLIO and UNISONO use keying material from the keystore. CLIO uses the Cryptographic Engine to encrypt and decrypt the communication; UNISONO uses a library of its own as it runs in a different process.

8.3.4.2 Protecting Against Misuse Using CLIO

SpoVNet applications can use an authorized CLIO to access UNISONO. An attacker could thus write a malicious application in an attempt to abuse the system for his own purposes. It is therefore important to add a protection against undesired orders.

State Information in CLIO

CLIO will not reply to messages that are not expected. For example, CLIO does not accept measurement replies or remote orders that are not addressed to the node it is running on.

Avoiding Overload

An attacker can write a SpoVNet application that issues valid measurement orders, but in such a way that they can harm the local system (or a remote one). For example, an application may attempt to request several hundred or thousands of bandwidth measurements. UNISONO must not execute these orders to avoid local overload. With remote orders, there is an even greater danger. An application can send the same remote order, e.g. to measure bandwidth to a certain node, to a large number of other nodes. This would resemble the consequences of a Reflected Denial-of-Service Attack (as described, e.g. in [Paxs01]). There is no perfect protection against such attacks; our goal is thus to raise the barrier for an attacker. We use policies in UNISONO to this end.

The definition of policies to prevent overload by orders from local applications is straight-forward. In some cases, role-based access control to measurement modules may be enough. In others, this must be enhanced with rate limits and cost restrictions.

Policies gain more importance in the context of remote orders. The most restrictive rule would be to disallow all or any but the least expensive data items for remote orders. However, rate limits and especially destination limits are a more flexible solution. For example, one can define that there may be no more than 3 measurements (of any kind) per minute to the same destination node. This is shown in figure 8.4: orders exceeding the rate limit are denied; the destination node receives a certain degree of protection. However, for such a policy to be effective, it is necessary that all or nearly all nodes of a SpoVNet employ the same or at least very similar policy settings. This can be achieved with the initiator concept from section 2. The initiator can pre-define policies for his network, which are then distributed when a new node joins. Nodes should follow these policies, if only acting out of self-interest.

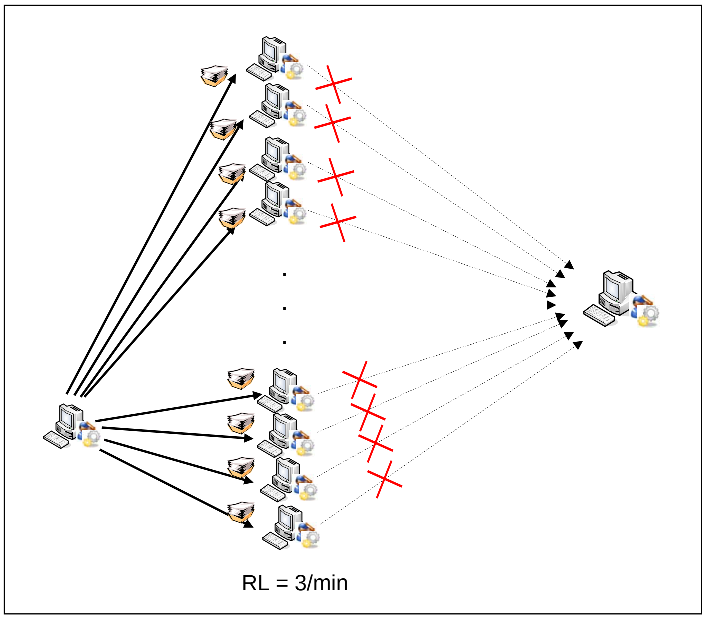


Figure 8.4: Preventing exploitation of CLIO/UNISONO by using rate limits.

Limits of Protection via Policies

To get an idea how much protection can be offered by policies, we evaluated the concept for a few network settings. We only briefly present some results here.

The effect of the protection depends mainly on the size of the SpoVNet and the number of nodes that are known to the attacker (i.e. to whom he can send remote orders). For example, assume a network of $1,000$ nodes, of which 10 are mal-configured and 100 (randomly chosen) are known to the attacker. The attacker may issue 50 measurement orders per minute[1]. If the network is unprotected, an attacker can thus achieve that $5,000$ measurements to a single node are executed per minute. If a destination limit of 3 orders per minute to any given node is set, however, this is reduced to about 53 measurements per minute. In a network with 100 nodes, of which 1 is mal-configured and 10 are known to the attacker, the attacker could cause up to 500 measurements per minute to a victim node (unprotected case). With the same policy as above, this can be limited to less than 10 measurements. However, policies work less well in larger networks. Assuming 10^6 nodes, of which 10^4 are mal-configured and 10^5 known to the attacker, an attacker can still cause $50,000$ measurements even in the protected case.

Our conclusion is thus that policies are very helpful for smaller networks, but lose much of their effect in very large ones. The question is which SpoVNets would reach such large dimensions. SpoVNets are by definition spontaneously created networks, with a limited number of users. It is thus plausible that many SpoVNets will have only a limited number of participants, and policies remain effective. The critical point will nonetheless be that all nodes employ the same policy. This can be enforced to a good degree thanks to the special role of the initiator. Where networks do reach very large sizes, the best protection would probably be very prohibitive policies that allow remote orders either to be executed only extremely rarely, or possibly only from a few trusted nodes – or else to disable remote orders entirely.

Finally, it should be mentioned that it is of course also possible to define rate limits for outgoing remote orders. However, this is only effective against attackers that have no control over the local CLIO and UNISONO. Other attackers are able to circumvent the rate limit by using a patched version of the software.

8.4 TP3 - Event Service

The Event service provides a progressive communication paradigm - event-based communication enhanced by in-network detection of correlated events.

A common way to implement event-based communication is the publish/subscribe paradigm. Publish/subscribe supports asynchronous interactions between communication entities through space, time and synchronization decoupling. Publishers inject information into the publish/subscribe system, and subscribers specify the events of interest by means of subscriptions. The publish/subscribe routing infrastructure is responsible for receiving the events from the publishers and notifying all the subscribers with matching subscriptions.

8.4.1 Security in publish/subscribe system

The publish/subscribe system developed as a part of the Event Service focuses on two aspects: access control and confidentiality. Access control in the context of publish/subscribe means that only authenticated publishers should be allowed to disseminate events in the network and those events should only be delivered to the authorized subscribers. Similarly, events should not be visible to the routing infrastructure and a subscriber should receive all relevant events without revealing its interests/subscription to the system.

Traditional mechanisms to provide access control and confidentiality cannot be directly applied in a content based publish/subscribe system. For example, end-to-end authentication is difficult to achieve

[1]This is a reasonable assumption as the attacker has to choose different data items each time. Otherwise, UNISONO would only return results from the cache.

due to the decoupled nature of interactions between publishers and subscribers. Similarly, traditional mechanisms to provide confidentiality by encrypting the whole event message conflicts with content based routing. Therefore, the Event Service employs security mechanisms of its own, which are built on top of basic security mechanisms provided by the SpoVNet Security framework. The following requirements are placed on the SpoVNet security framework.

- Publishers and subscribers have cryptographic NodeIDs i.e. they can prove that they are authentic SpoVNet Nodes.
- The SpoVNet base (Ariba) provides secure end-to-end connections between publishers and subscribers.
- The Cryptographic Engine in the SpoVNet Security Component provides a module with simple pairing based cryptographic primitives to realize identity based encryption (IBE). Furthermore, secure storage is available to store the cryptographic keys and state information.
- The SpoVNet initiator acts as a trusted authority. It manages the master keys and issues keys to publishers and subscribers. The initiator can however delegate responsibilities to other SpoVNet nodes.

The security is provided in the presence of the following threats. No SpoVNet nodes are entitled to publish or subscribe to every information. Publishers can try to fabricate fake events. Similarly subscribers can subscribe to more events than they are authorized. Furthermore, publishers and subscribers are curious to obtain information about other events (to which they have not subscribed) and subscriptions in the system.

8.4.2 Identity based encryption

In 1984 Shamir [Sham84] introduced the concept of identity-based encryption (IBE) where a public key can be any binary string identifying its owner non-ambiguously. IBE enables any pair of users to securely communicate and to verify each others signatures without the need for public-key infrastructure (PKI).

In a traditional public-key infrastructure a public-key certificate is required to bind the key to its user. Senders and receivers are strongly coupled, i.e. before a sender can encrypt a message, the receiver must generate a public/private key pair, sign its public key by a certificate authority and communicate it to the sender.

Instead, IBE requires a trusted authority for the generation and distribution of private keys to the users. It removes several troubles associated with traditional PKIs such as certificate management and provides decoupling between the communication entities. IBE allows a sender to encrypt and send the message at any time. Receivers, on the other hand, should contact a trusted authority to obtain the corresponding private key. The authenticity of the public keys is guaranteed implicitly as long as the transport of the private keys to the corresponding users is kept secure.

Figure 8.5 shows the basic communication steps for identity based encryption.

1. The trusted authority generates a master public/private key pair. The master public key is known to all users in the system, whereas the private key is kept secret.
2. Alice wants to send a message to Bob using Bob's email address as public key. It encrypts the message using Bob's email address and the master public key.

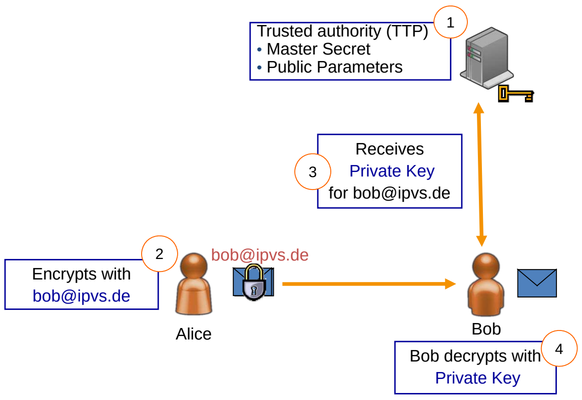


Figure 8.5: Identity based encryption

3. Bob contacts the trusted authority to obtain the private key that corresponds to its email address. The authority authenticate Bob's identity (ensures he is a valid user) and establish any other policy element. It generates the Bob's private key using Bob's email address and master private key.

4. The authority returns Bob's private key, so Bob can decrypt any message encrypted with its email address as public key.

8.4.3 Applicability to the Publish/Subscribe System

Identity based encryption provides the desired decoupling between the communication partners and is therefore a suitable paradigm to build secure a publish/subscribe system. The following sections will give an overview of different mechanisms needed to provided security on top of the spatial indexing based content routing approach developed as a part of WP3.1 and WP3.2.

8.4.3.1 Spatial indexing overview

For completeness, we will first briefly explain the spatial indexing approach. The event space, composed of d distinct attributes $\Omega = \{A_1, A_2, ..., A_d\}$ can be geometrically modelled as a d-dimensional space so that each dimension represents an attribute. Subscriptions and advertisements are represented by hyperrectangles in Ω, whereas published events represents a point.

With the spatial indexing approach, the event space is hierarchically decomposed into regular sub-spaces which serve as enclosing approximations for the subscriptions and advertisements. The decomposition procedure divides the domain of one dimension after the other and recursively starts over in the created sub-spaces. For example, a 2-dimensional event space is first divided in dimension d_1 into two subspaces, then in dimension d_2 into four sub-spaces, then again in dimension d_1 into eight sub-spaces and so on. Each sub-space and hence the subscription/advertisement approximated by it can be identified by a dz-expression. A dz-expression is a bit string of "0"s and "1"s. A containment relationship can be easily defined between the sub-spaces and their dz-expressions. In general, a sub-space represented by a dz-expression dz_1 is covered by the sub-space represented by dz_2, iff dz_2 is a prefix of dz_1, e.g. 00 is covered by 0.

An event can be approximated by the smallest (finest granularity) sub-space that encloses the point represented by it. An event dz_e matches a sub-space dz_s if dz_e is covered by dz_s. In general, the number of sub-spaces matched by an event is logarithmic with respect to the number of sub-spaces at the finest level and is directly proportional to the length of its dz-expression. For example, an event 0010 is covered/matched by the sub-spaces 0010, 001, 00, 0 and Ω.

The publisher/subscriber overlay is maintained strictly according to the containment relationship between the subscriptions, i.e. subscribers whose subscriptions map to bigger sub-spaces are placed higher in the dissemination tree. Furthermore, subscribers receive and forward only those events which match the sub-spaces approximating their subscriptions. In order to maintain such a topology each subscriber should know the subscription of its parent and child peers. When a new subscriber arrives, it sends the connection request along with its subscription to a random peer in the overlay network. The connection request may be forwarded by many peers in the overlay network until it reaches the right peer to connect. Each forwarding peer matches the subscription in the request with the subscription of its parent and child peers to decide the forwarding direction.

A subscription/advertisement can be composed of several sub-spaces and is therefore represented by a set of dz-expressions. However for brevity, security mechanisms in this report are explained with the assumption that each subscription and advertisement is mapped to a single dz-expression. Details about the extension of the mechanisms to the set of dz-expressions can be found in the milestone report for WP3.4.

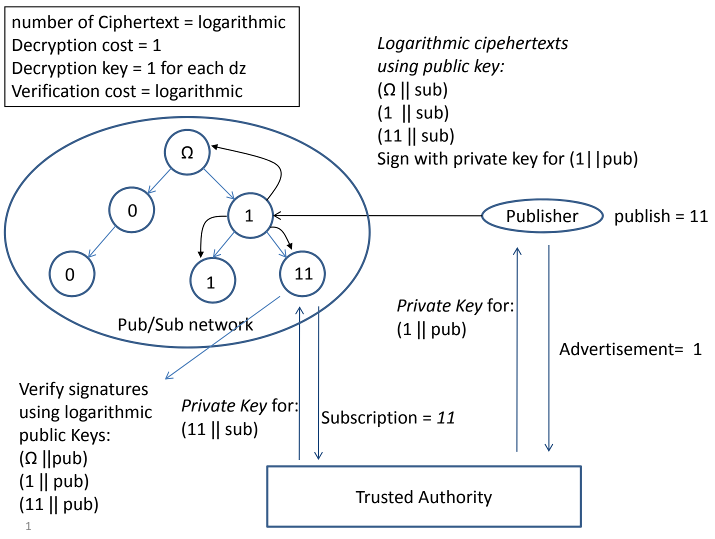

number of Ciphertext = logarithmic
Decryption cost = 1
Decryption key = 1 for each dz
Verification cost = logarithmic
Logarithmic cipehertexts using public key:
(Ω || sub)
(1 || sub)
(11 || sub)
Sign with private key for (1||pub)
Ω
0
1
0
1
11
Pub/Sub network
Publisher
publish = 11
Private Key for: (1 || pub)
Advertisement= 1
Verify signatures using logarithmic public Keys:
(Ω ||pub)
(1 || pub)
(11 || pub)
Private Key for: (11 || sub)
Subscription = 11
Trusted Authority

Figure 8.6: IBE to provide publisher/subscriber authentication and event confidentiality.

8.4.3.2 Authentication of publishers and subscribers

Before starting to publish events, a publisher should contact the trusted authority along with its advertisement. If the publisher is allowed to publish events within the sub-space represented by its advertisement, the authority uses the master private key to generate and return the private sign key $Private_{SignKey}$ that corresponds to $Public_{SignKey} = (Advertisement||pub)$. For example, a publisher with advertisement represented by a dz-expression 1 will receive private sign key for $(1||pub)$. Publishers use $Private_{SignKey}$ to sign each published event so that the subscribers can verify the validity of an event using $Public_{SignKey}$

Similarly, to receive events within a sub-space, a subscriber should contact the trusted authority and authenticate itself to the authority. Once the authority authenticates the subscriber's identity and its subscription, it returns the private key $Private_{DecKey}$ that corresponds to the public key $Public_{DecKey} = (sub - space||sub)$. For example, a subscriber authorized to receive events within the sub-space 11 will receive the private key for $(11||sub)$. The $DecKey$ pair is used to provide event confidentiality.

Clearly, pub and sub strings are used to differentiate the keys used for the verification of valid events from the ones used to provide event confidentiality. Furthermore, any peer (subscriber or publisher) in the system can generate the public keys without contacting the trusted authority.

8.4.3.3 Confidentiality

Event confidentiality

When a publisher wants to send an event in the system, it generates a ciphertext for each sub-space that encloses the event, the reason being that the subscribers of any of these sub-spaces should be able to successfully receive the event. To generate a ciphertext for a sub-space represented by dz, $Public_{DecKey} = (dz||sub)$ along with the master public key is used to encrypt the event. Finally, the ciphertexts are signed using the $Private_{SignKey}$ of the publisher and disseminated in the system. For example, in figure 8.6, to publish an event 11, ciphertexts are generated for sub-spaces 11, 1 and Ω. Furthermore, generated ciphertexts are placed in the ciphertext-message strictly according to the containment relationship between the sub-spaces as shown in figure 8.6.

On receiving the ciphertexts, a subscriber tries to decrypt them using his $Private_{DecKey}$. The ciphertexts are strictly ordered according to the containment relation, therefore a subscriber only tries to decrypt the ciphertext whose position coincides with the position of its sub-space in the containment hierarchy of sub-spaces. For example, in figure 8.6, the subscriber with subscription 11 can only decrypt the last ciphertext. Successful decryption can be easily determined by either including the hash of the plaintext-message in the ciphertext or ending the message with a predefined sequence of zeros.

Once the decryption is successful, the subscriber tries to verify the publisher's signature using $Public_{SignKey}$. However, due to the decoupling, a subscriber does not know the advertisement of the publisher. For example, an event 11 can be originated by the publisher with advertisement 11, 1 or Ω. Therefore, the signature verification cost is logarithmic. A subscriber tries to verify a signature by using $Public_{SignKey}$ for each possible sub-space that encloses the event.

Subscription confidentiality

As mentioned earlier, the publish/subscribe overlay is maintained according to the containment relationship between the subscriptions of the subscribers. In order to maintain such a topology, each peer should know the subscription of its parent as well as the child peers, which makes providing subscription confidentiality very hard. An algorithm is developed to provide weak confidentiality while maintaining the desired overlay topology. The weak notion of confidentiality means that the following information can be leaked/inferred about the subscription of a subscriber:

- *Parent peer*: Subscription of a child peer is either the same or smaller than mine
- *Child peer*: Subscription of the parent is either the same or bigger than mine
- *Arbitrary peer*: Subscription of the other peer is neither the same nor smaller than mine.

The algorithm is based on the idea that a subscriber can only connect to those peers in the overlay topology, whose subscriptions are either the same or bigger than its own subscription. For example, a subscriber with subscription 00 can only connect to parents with subscriptions 00, 0 or Ω. When a new subscriber arrives in the system, it makes a secret connection request. The secret connection request contains a ciphertext for each sub-space to which the new subscriber (requester) is allowed to connect. In the worst case, the number of allowed sub-spaces can be logarithmic (in the number of sub-spaces at the finest level). Any peer who can successfully decrypt any of the ciphertexts is a potential parent of the requester. However, other peers cannot guess anything about the subscription of the requester. The details of the algorithm can be found in milestone report for WP3.4.

8.4.3.4 Event dissemination

The overlay maintenance guarantees that the subscription of a parent peer covers the subscription of its child peers. This property makes provision of subscription confidentiality in the presence of content based filtering very difficult. The parent peer can decrypt every event, which he forwards to his child peers, by maintaining histories he can eventually discover the subscription of its child peers. Two mechanisms can be used to mitigate this problem:

- *One hop flooding*: The weak notion of subscription confidentiality allows a parent to infer that the subscriptions of its child peers are smaller or equal to its own. Therefore, in one hop flooding, a peer assumes that the child peers have the same subscription and forwards each successfully decrypted event to all child peers. However, the subscription of a child peer may be smaller and thus may results in false positives.

  The one hop flooding mechanism works as follows. The publisher forwards the ciphertexts of an event to a randomly selected subscriber in the dissemination tree. Once the event is received, the subscriber tries to decrypt and verify it. In case of failure, the event is only forwarded to the parent (unless the event is received by the parent). However in case of success, the event is forwarded to all the child peers without considering their subscriptions.

- k *subscription* : In this mechanism, a subscribers divide its subscription into k sub-spaces and connects to a separate parent for each sub-space. The complete subscription of a subscriber cannot be determined unless k subscribers collude with each other. However, this mechanism has the added cost of maintaining more parent connections and managing more $Private_{DecKey}$ keys.

9. Conclusion

This report has presented the results of the Security Task Force in SpoVNet, to which members from TP1, TP2 and TP3 have contributed.

We have shown why there is a distinct need for security in SpoVNets and how SpoVNet differs from the conventional approach on the Internet by pursuing *integrated security* from the start. Applications and services are freed from the details of implementing security by defining their *requirements* in an abstracted way. *Policies* are used to define how the security of a SpoVNet network and a SpoVNet node is to be achieved. *End-to-end security* is provided with the help of cryptographic identifiers. We have also shown which different *types of SpoVNets* there can be – open ones, access or capability-restricted and hidden ones. We have given example scenarios of SpoVNets and have shown what their security requirements are and how they can be achieved with SpoVNet security concepts. Furthermore, we have discussed security models for the evaluation of our concepts. These efforts have resulted in the design of the *Security Component*, whose requirements and architecture have been presented here. The Security Component is used by services from different TPs to achieve security goals.

References

[AbBF07] M. Abadi, B. Blanchet und C. Fournet. Just fast keying in the pi calculus. *ACM Transactions on Information and System Security* 10(3), July 2007, p. 9–68.

[Aura05] T. Aura. Cryptographically Generated Addresses (CGA). RFC 3972 (Proposed Standard), March 2005. Updated by RFCs 4581, 4982.

[AVI07] AVISPA: Automated Validation of Internet Security Protocols and Applications. http://www.avispa-project.org/, July 2007.

[BaBK02] S. Banerjee, B. Bhattacharjee und C. Kommareddy. Scalable Application Layer Multicast. In *Proc. of SIGCOMM'02*, Pittsburgh, Pennsylvania, USA, October 2002. p. 205–217.

[BaMi07] I. Baumgart und S. Mies. S/Kademlia: A Practicable Approach Towards Secure Key-Based Routing. In *Proc. Int. Workshop on Peer-to-Peer Networked Virtual Environments 2007 (P2P-NVE 2007) in conjunction with ICPADS 2007*, Volume 2, Hsinchu, Taiwan, December 2007.

[BEPW02] P. Biddle, P. England, M. Peinado und B. Willman. The Darknet and the Future of Content Distribution. In *Proceedings of the 2002 ACM Workshop on Digital Rights Management*, 2002.

[BoFr01] D. Boneh und M. K. Franklin. Identity-Based Encryption from the Weil Pairing. In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, London, UK, 2001. Springer-Verlag, p. 213–229.

[Boyd93] C. Boyd. Security architecture using formal methods. *IEEE Journal on Selected Topics in Communications* Band 11, 1993, p. 694–701.

[CaKr02] R. Canetti und H. Krawczyk. Security Analysis of IKE's Signature-Based Key-Exchange Protocol. In *Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '02)*, August 2002, p. 143–161.

[CaMo02] C. Castelluccia und G. Montenegro. Securing Group Management in IPv6 with CGA. Presentation at 53th IETF Meeting, February 2002.

[Crem06] C. Cremers. *Scyther - Semantics and Verification of Security Protocols*. Dissertation, University Press Eindhoven, 2006.

[DoYa83] D. Dolev und A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory* 29(2), 1983, p. 198–208.

[Gutm02] P. Gutmann. PKI: It's Not Dead, Just Resting. *IEEE Computer* 35(8), August 2002, p. 41–49.

[HSDD+05] C. He, M. Sundararajan, A. Datta, A. Derek und J. C. Mitchell. A modular correctness proof of IEEE 802.11i and TLS. In *Proceedings of the 12th ACM conference on Computer and communications security (CCS '05)*, November 2005, p. 2–15.

[LaSP82] L. Lamport, R. Shostak und M. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems* 4(3), 1982, p. 382–401.

[Libe07] Liberty Alliance. Identity Federation. http://www.projectliberty.org, July 2007.

[Lowe97] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop, Rockport, MA, USA (CSFW '97)*, 1997.

[Mitt97] S. Mittra. Iolus: A Framework for Scalable Secure Multicasting. In *SIGCOMM*, 1997, p. 277–288.

[MoNi06] R. Moskowitz und P. Nikander. Host Identity Protocol (HIP) Architecture. RFC 4423 (Informational), May 2006.

[NiLD07] P. Nikander, J. Laganier und F. Dupont. An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers (ORCHID). RFC 4843 (Experimental), April 2007.

[Paxs01] V. Paxson. An analysis of using reflectors for distributed denial-of-service attacks. *ACM SIGCOMM Comput. Commun. Rev.* 31(3), 2001, p. 38–47.

[PCST01] A. Perrig, R. Canetti, D. Song und D. Tygar. Efficient and Secure Source Authentication for Multicast. In *Network and Distributed System Security Symp.*, February 2001, p. 35–46.

[PSCT+05] A. Perrig, D. Song, R. Canetti, J. D. Tygar und B. Briscoe. Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction. RFC 4082 (Informational), June 2005.

[Sham84] A. Shamir. Identity-Based Cryptosystems and Signature Schemes. In *Proceedings of CRYPTO, Lecture Notes in Computer Science*, Volume 7, 1984, p. 47–53.

[StRe06] D. Stutzbach und R. Rejaie. Understanding Churn in Peer-to-Peer Networks. In *Proceedings IMC, ACM SIGCOMM*, Rio de Janeriro, Brazil, 2006. p. 189–202.

[Toll09] A. Tollenaere. Security for Overlay-Based Middleware. Diplomarbeit, Universität Karlsruhe (TH), July 2009.

[WoGL98] C. K. Wong, M. G. Gouda und S. S. Lam. Secure Group Communications Using Key Graphs. In *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, 1998, p. 68–79.