UNIVERSITY OF TWENTE.

Thesis submitted to the
University of Twente
Faculty of Electrical Engineering, Mathematics and Computer Science
chair for Design and Analysis of Communication Systems
in partial fulfillment of the requirements for the degree of

Master of Science
in
Telematics

Secure Access Control to Personal Sensor Information in Federations of Personal Networks

J.W.C. Beusink
July 12, 2012

Supervising committee
Dr. Ir. G. Karagiannis*
Dr. Ir. G.J. Heijenk*
Dr. H. Benz**
Prof. Dr. Ir. S. Heemstra de Groot**

*
University of Twente,
Faculty of EEMCS,
DACS chair,
P.O. Box 217,
7500 AE Enschede,
The Netherlands.

**
Twente Institute for
Wireless and Mobile Communications B.V.,
Business & Science Park,
Institutenweg 30,
7521 PK Enschede,
The Netherlands.

Any intelligent fool can make things bigger and more complex... It takes a touch of genius - and a lot of courage to move in the opposite direction.

— Albert Einstein (1879-1955)

Abstract

This thesis provides a secure access control architecture for personal sensor information in Federated Personal Networks (FedNets) applied to the context of the VITRUVIUS project. To that end suitable authentication protocols, cipher suites, credential providers and policy languages are analyzed. We provide and test a prototype of our proposed architecture.

Security in this context entails more than the usual suspects being authentication, authorization, non-repudiation, data integrity and confidentiality. Due to the nature of a PN, confidentiality is notably complex. Privacy in this context consists of user and component identity confidentiality, user location confidentiality and user untraceability. Mobile devices are also susceptible to depletion attacks, aimed at draining the battery.

We found EAP-IKEv2 the best suitable authentication protocol based applicable security requirements we adopted from several fields of study.

We recommend a ciphersuite consisting of ECDH, ECDSA, AES and SHA-2 based upon keystrength, governmental and institutional recommendations and the wireless nature of PNs and FedNets.

We recommend WebDAV as credential provider as its usage allows for more efficient revocation checking.

We recommend PERMIS as reasoning engine along with its policy language.

Our prototype shows that the suggested security framework can be run on a resource constrained device though further performance improvements to the authentication and the authorization engine are needed.

Dedication

This thesis is dedicated, in loving memory, to my father Johan Willem (Joop) Beusink, who passed away on June 7, 2012, during the final stages of my thesis. I know you were worried about me graduating and wanted to be there at my graduation. Thank you for everything you have shown me, done for me, and inspiring me to be a better man.

Acknowledgements

I would like to express gratitude to my supervisors, Dr. Ir. G. Karagiannis, Dr. Ir. G.J. Heijenk, Dr. H. Benz and Prof. Dr. Ir. S. Heemstra de Groot, without whom this thesis would not have been made possible.

I would also like to thank experts in the field for creating and making the software available that is the basis on which our prototype was developed and for providing answers to questions on the inner workings of this software. For strongSwan they are Tobias Brunner; Martin Willi; and Prof. Andreas Steffen, HSR University of Applied Sciences Rapperswil, Switzerland. For PrivilEge and Role Management Infrastructure Standards (PERMIS) they are Prof. BSc. PhD. David W. Chadwick, University of Kent, United Kingdom; Dr. Stijn Lievens, University of Kent, United Kingdom.

For their patience and financial support I thank Everett NL B.V.

Special thanks to Dr. Ing. Bianca Beusink and BSc. Achiel van der Mandele for reviewing drafts and providing usefull feedback.

Last but not least I would like to thank my friends and family for their support and giving me the strength to finish this thesis.

Contents

Chapter 1

Introduction

This thesis provides a secure access control framework for personal sensor information in Federated Personal Networks (FedNets) applied in the context of the Versatile Interface for TRUstworthy VItal User (oriented) Services (VITRUVIUS) project. The VITRUVIUS projects takes a top-down approach. The work presented in this thesis is an elaboration of this approach.

1.1 Context/Motivation

The VITRUVIUS project [133] aims to provide a system architecture for a Body Sensor Network (BSN) that can be used simultaneously for entertainment, lifestyle and social services. The BSN consists of wireless sensors applied to a person and a body hub, a device — e.g., a Personal Digital Assistant (PDA) — that communicates with the sensors and the outside world. The body hub can process, store and relay sensor information. It may be configured to being reactive and proactive in relaying sensor information, depending on the available connection and the derived urgency determined by a reasoning engine. A large boost for acceptance of this concept can be achieved by standardizing the architecture so that service providers need not roll out their own body sensor hardware.

The concept of a BSN has striking similarities with the concept of a FedNet. A FedNet is a collaboration of two or more Personal Networks (PNs) sharing one or more (personal) resources to achieve a common goal. Examples of resources are files, such as pictures and services, such as printing [96].

A PN is an ubiquitous network of personal devices that communicate independently from their geographical location, is context aware, self-organizing and it provides secure connectivity [97]. In other words it is a set of devices owned by a particular user that communicates securely regardless of where the devices are located. The best way to communicate is automatically de-

termined based on the resources available at each device.

A BSN could be integrated in a PN, thereby the FedNet framework secures the communication of external entities to the BSN. Obviously sensitive medical data needs to be handled in a secure way.

The following story illustrates such a FedNet that incorporates a BSN. The case is taken and adapted from [61].

Harry is a sportive individual with a liking for triathlon. To improve his training results, he obtains a Sports Body Sensor Kit (BSK) for Runners consisting of a set of wireless sensors and a small box that connects wirelessly to his iPhone.[1] *Harry connects it with a few clicks and thereby integrates it into his Personal Network. With this kit, Harry buys some 'off-the-shelf' service from a service provider that provides him with various visualizations, evaluations and training suggestions based on his sensor data.*

Harry also discovers that a particular gymnasium supports the BSK. When he is training there now, the devices he works with provide additional sensor readings (speed of the treadmill etc.). For a small fee, the gymnasium offers additional training service software that is more refined and specialized for this type of training. When Harry signs up for the gymnasiums service, he authorizes them to access parts of his BSK. This federates his PN with the network of the gymnasium for these particular services during the time he is there. When he subsequently enters the gymnasium , their software automatically uploads to his computer and is flushed when he leaves. Thus the gymnasium can only access Harry's sensor data when he is present in the gymnasium. The training service of the gymnasium can provide Harry with useful hints using his iPhone and control training parameters of the devices he uses.

Later in his sports career, Harry moves to a professional triathlon coach, who provides him with much more sophisticated sensors (larger number of high-resolution high-sensitive accelerometers that can be worn under water etc.). His coach also has access to a suitable high-grade motion and training analysis services software. As before, a few clicks on Harry's phone federate his BSK with the service provider of his coach. In order to integrate Harry's general daily activity pattern into the training, he grants the coaching software access to a few types of sensor data throughout the day.

At some moment, the stress at his work gets to Harry and he develops a sleeping disorder. This brings him to a specialized medical center where they routinely monitor patients using the very same BSK, but some different sensor (sleep disorder patient monitoring). Monitoring a person's sleep at home allows the center to treat more patients faster because they do not need to be hospitalized and thus do not require the accommodation period to the

[1] Sensors come packaged in wireless, easy-to-wear fabrics with for instance accelerometers, skin sensors for temperature, resistance, heart rate, blood pressure, blood oxygenation. The small box also connects to other hype-products like iPod, PlayStation Portable (PSP), or a regular PDA. It may also come integrated in special PDAs.

strange environment. Doctors can directly express decision rules graphically using a decision support system, for instance based on real-time monitored parameters such as heart beat, leg motions, sound patterns (e.g., snoring). The service can control, for instance, lights and acoustic devices at Harry's home to guide his sleep. Within a few weeks and without any need for hospitalization Harry is cured.

1.2 Specific Problem

Information is, in the right context, very valuable. Consider Harry's heart rate and blood pressure monitor. In the gymnasium this provides useful information aiding Harry in his training. When applying for health insurance, Harry does not want the insurance company to pick up on his slightly higher blood pressure. More importantly, consider the case where a burglar were to be able to deduct from sensor readings when Harry is in the most deep phase of his sleep or even home at all. Security is needed to protect Harry from unwanted access to such information.

Access control is an important aspect in security. It specifies *who* is allowed to access *exactly what*. In computer security, access control relies on several other procedures such as authentication and policies. Which implementations of these procedures are required in the context of the VITRUVIUS project FedNets needs to be determined.

In this thesis, we provide a security architecture for a FedNet that can be applied to the VITRUVIUS project.

1.3 Research Questions

The main research goal of this thesis is:

- Specification, design and implementation of a scalable and refined access control architecture for a federation of personal networks that provides a high degree of security and privacy.

The main research objectives are:

- Refine a suitable authentication protocol to work in the suggested framework.
- Investigate and select suitable cipher suites to be used within the suggested framework.
- Refine a credential provider to work within the suggested framework.
- Refine a suitable policy language to work in the suggested framework.
- Develop an experimental system (prototype) that demonstrates the basic principles.

- Study the scalability of the solution taking into account:
  - the role of security and access control mechanisms
  - extensions of the platform with new components such as sensors, actuator and personal devices
  - new applications

1.4 Approach

We will first introduce the concepts of PNs, FedNets and Access Control Architectures (ACAs) to provide a clear context in which the research questions are to be answered. Next we will go into each of the research objectives. Then we will combine the outcomes of the research objectives to propose a architecture. We will then translate this architecture into a prototype and evaluate its performance.

1.5 Structure

The structure of this thesis is as follows. In Chapter 2 the concept of a PN will be explained and in 3 the concept of a FedNet. In Chapter 4 the concept of an ACA will be explained. In Chapter 5 a comparison of authentication protocols will be presented and a suitable one will be selected. In Chapter 6 keys are presented and a suitable ciphersuite will be selected. In Chapter 7 several credential providers will be presented and a suitable one will be selected. In Chapter 8 several policy languages will be presented and a suitable one will be selected. In Chapter 9 an architecture for use in the context of VITRUVIUS will be proposed along with a prototype. In Chapter 10 the prototype will be evaluated. In Chapter 11 we present our conclusions and discuss further work.

In Appendix A we present the acronyms used in this Thesis. In Appendix B we give instructions on how to reproduce our results. In Appendix C we give detailed specifications of the used hardware. In Appendix D we present the confidence intervals that we calculated in tabular form.

In Appendix E we present the java code that we have written. In Appendix F we present diff files, the modifications that we made to other programs.

Chapter 2

Personal Networks

Harry just obtained the BSK. The BSK needs to know that it is part of Harry's PN, i.e., the BSK needs to be personalized. He therefore initializes the BSK for use in his PN. Having obtained the right credentials along with configuration parameters, the BSK can set up security associations and communicate with other personal devices in the vicinity.

This chapter explains the concept and working of a PN. A PN is a ubiquitous network of devices all having the same owner. More and more devices come equipped with communication capabilities. Letting these devices communicate with each other and making their services available independently of their geographical location can be of use to the owner. By providing ubiquitous connectivity and mobility support, a user can access all content and services regardless of the device or location.

PN-communication-capable devices, called nodes, together create a network. Much like the typical Local Area Network (LAN) found in many homes. Like a Personal Area Network (PAN), the concept of a PN allows for an ad-hoc network consisting of wired and wireless nodes around you. For security reasons a PN, unlike a PAN, only allows devices owned by you to communicate directly to each other. Thus creating a Private PAN (P-PAN), acting as a LAN being shielded from the internet by a firewall.

All nodes that are able to communicate with each other without the need of a supportive third party infrastructure together are called a cluster. By definition a PAN moves around with you, as it is a network in your vicinity. The cluster that in effect is a P-PAN is called the local cluster. A cluster in general, unlike a PAN, is not restricted to being in your vicinity. Your nodes at home can also combine in a cluster even if you are not there. This cluster is called a home cluster, as it resides in your home. A PN can consist of many clusters, such as local cluster, a home cluster, a car cluster and an office cluster. Each cluster should at least contain one node with

communication capabilities to other clusters and nodes that are not part of its PN. All clusters owned by a single person combined are called together a PN. Figure 2.1 depicts an example of a PN containing several clusters.

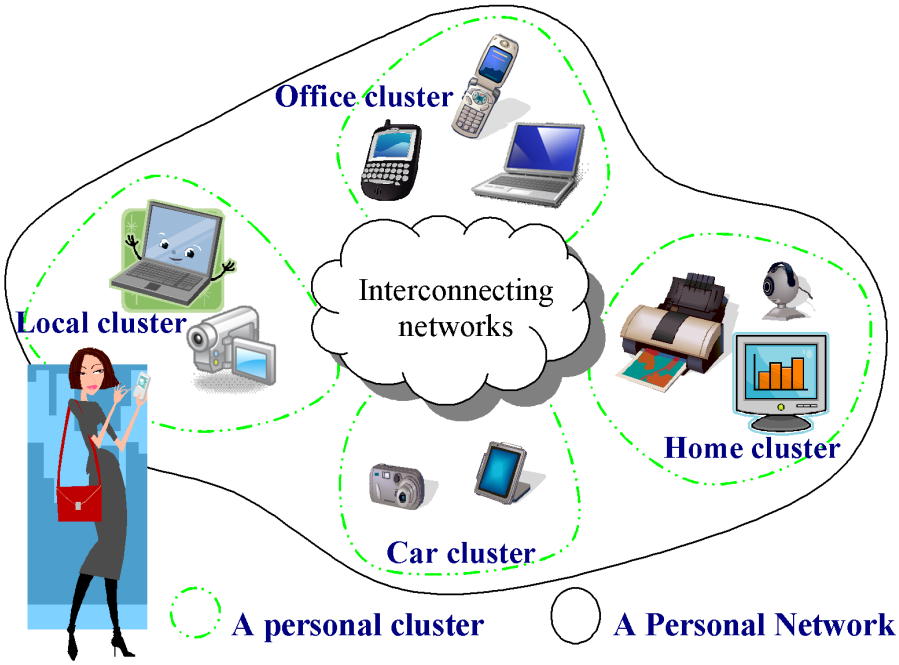

Figure 2.1: Example of a PN. Figure reproduced from [55] with permission from the authors.

The concept of a PN has been extensively studied by the Personal Network Pilot 2008 (PNP2008) project [102] and the My personal Adaptive Global NET (MAGNET)[82] and MAGNET Beyond[83] projects. This chapter is largely based on PNP2008 and references [44, 52, 54, 55, 68, 69, 96, 97, 139].

2.1 Requirements

Functionality can be specified in terms of requirements. In order to promote PN implementation instead of stating requirements PN developers have used concerns. Concerns are issues that should be addressed in a full fledged PN. This allows for a gradual increasing maturity of the implementation. A developer may choose to implement, thereby only a particular concern, thus not having to address all concerns simultaneously, in which it has expertise. The concerns [65] are stated below.

1. **Ubiquitous Connectivity with Mobility Support**
   A PN should be able to connect via an infrastructure-based network as

well as an ad-hoc network, keeping connectivity whilst being mobile. This means universally identifiable and reachable devices. Addressing support is therefore needed. In light of future technologies it should support IPv6 as well as IPv4.

2. **Trust Relationships and Authentication**
   Trust relationships must be supported in order to have authentication and authorization. A universal trust model is needed, because users need to use different services and connections provided by a multitude of parties, such as their own nodes, foreign nodes, a service provider, other third parties, etc. This model should be efficient and work well within a PN as well as between PNs. Login procedures should be easy to use from a users point of view. For instance, instead of entering a personal key a personal digital assistant might use its camera to identify the user using facial recognition. Obviously the false acceptance and rejectance rate issues should be addressed when using a technique such as facial recognition.

3. **Context Awareness**
   [65] uses the definition stated in [7]: "Context is any information that can be used to characterize the situation of an entity". Using this information a PN is capable of offering better services. For example, when you are driving in your car, you might be interested in traffic jams, weather conditions, etc. PN Nodes should gather and efficiently distribute relevant context information.

4. **Content Management**
   PNs need to support smart content management. Storing a file remotely in a PN should work, from the users point of view, the same as storing it locally. Content should be accessible from all relevant devices in a PN at any given time. Automatic data backup could assist in data integrity and data availability in case of a unreachable cluster. A PN should provide unified data management, P2P communication, transcoding (to deliver content in an optimal way) and DRM.

5. **Service and Application Concerns**
   A PN should improve the mobile experience of the user in order to provide added value. It can do so by offering services such as, automatic backing up of data, or unrestricted ocation / time accessibility.

6. **Auto Configuration and Self-organization**
   A PN should improve the mobile experience of the user in order to provide auto configuration and self-organization at both the network and service levels. Clusters may be geographically dispersed and devices should be able to enter and leave clusters at any moment. The user should not be bothered with this. For example, a PN user comes

home from work and his local cluster merges with his home cluster, which has a faster internet connection.

7. **Quality of Service and Reliability**
   A PN should be supportive of QoS to enable a user to robustly use multimedia applications such as VoIP and streaming video. As different network technologies may use different metrics in routing and handover — such as bandwidth, bit error rate and acceptable latency — these metrics should be taken into account in a PN. Other reliabilities should also be taken into account in routing and handover. Such reliabilities include the fact that mobile devices could join or leave a cluster at any time, because a mobile device moves out of range, a device may run out of batteries or the device might be switched on or off by the user.

8. **Seamless integration of Heterogeneous Devices, Services and Technologies**
   Devices in a PN may have different communication capabilities, such as Ethernet, Zigbee, WLAN, Bluetooth and UMTS and must be able to communicate with each other. Possibly via other devices, as is the case with multi-hop ad-hoc networking. Capacities for different devices can vary significantly, such as processing power, battery life and storage capacities. The needs of services can also differ greatly, a file sharing program needs lots of cheap bandwidth, a medical application needs reliable communication and a VoIP call has QoS needs. All these different aspects need to be supported.

9. **PN Management**
   [65] defines the concern of PN management as:

   > Network management functionality typically deals with non-real-time management functions, such as fault-, configuration-, performance- and security management. Actions that people and/or organizations are required to do for PN management should be brought to a minimal level. However the user must still remain in control of his PN and may therefore need to be able to perform certain management tasks. The user must be able to manage his devices, services, content, settings (connections, security, ...) anywhere and anytime trough intuitive user interfaces on all relevant personal devices. At the same time, the PN must not be intrusive and instead support certain tasks instead of taking tasks over.

   It should be possible to outsource PN management to a third party, called a *PN provider*. The provider facilitates PN management for

multiple users and optimizes the availability of the resources to the managed PNs.

10. **Usable Security and Privacy**
    A PN should ensure that only with consent of the owner its data, services, PN-related content and devices in a PN can be accessed. A PN must also preserve user privacy and interwork with other security solutions. In order to protect against corruption of, or unauthorized access to, traffic and control, a PN should include expanded encryption techniques and user authentication.

11. **Scalability**
    A PN should therefore be scalable, as a PN might need to handle a large number of devices, services and applications. These may be numerous, affecting addressing schemes, service discovery and resource discovery. Some of these devices will be running on battery power and must be able to cope with this load for a reasonable amount of time.

2.2 Overall Architecture

A PN can be seen from different views, the service level, networking level and the connectivity level, each performing its own tasks. This section goes into more detail on what each level does. Figure 2.2 depicts the abstraction levels.

At the connectivity abstraction level, nodes are organized in (radio) domains with communication links between them. This is discussed in Section 2.2.1. At the networking level, Personal Nodes form Clusters and Clusters form PNs enabling user-centric networking. This is discussed in Section 2.2.2. On the service level, the applications, content and services are made available to the nodes in the PN. This is discussed in Section 2.2.3.

2.2.1 Connectivity Level Abstraction

The connectivity-level abstraction consists of the basic wired and wireless networks and protocols. It addresses concerns such as link levels, Medium Access Control, the physical link and their interrelationships. Because a lot of technologies, such as WLAN and Bluetooth, share the same radio frequency — e.g., the 2.4GHz ISM band — resource sharing and management should be handled in this level. As a PN is built upon such connections, the exact workings of the connectivity level are out of scope of this document.

2.2.2 Network Level Abstraction

The network-level-abstraction deals with how the nodes are distributed and how they communicate with each other. The network level has to operate

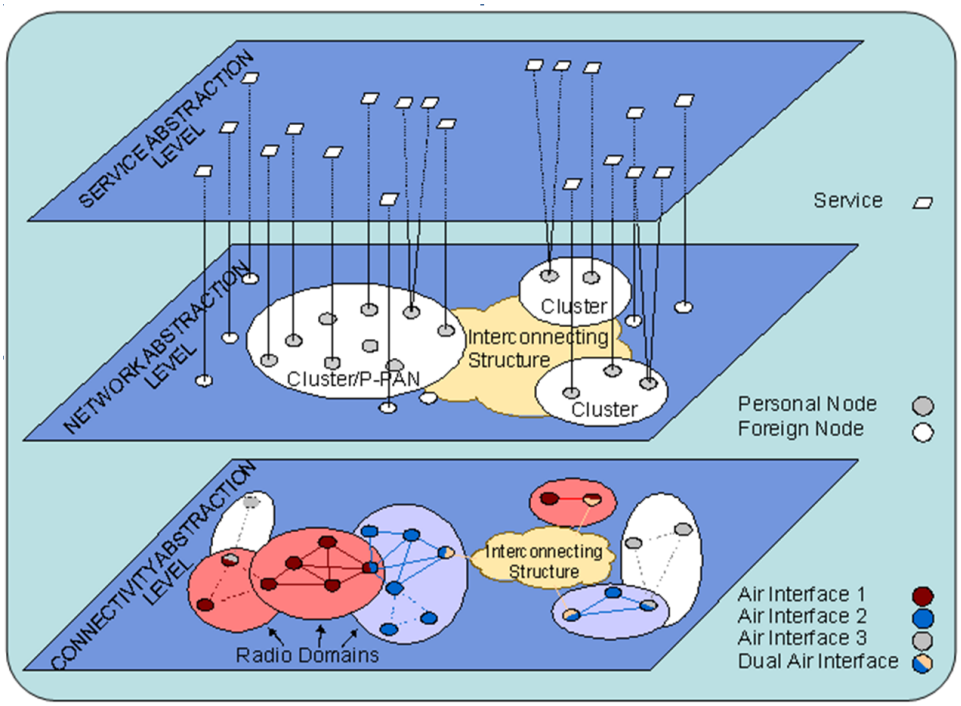

Figure 2.2: Architectural artifact concerning PNs. Figure reproduced from [65].

as independent as possible from the connectivity level so that current and future (wireless) communication technologies can be supported.

The network level addresses the network-related aspects of the following concerns:

- Ubiquitous connectivity and mobility support (see Point 1 in Section 2.1)
- Trust relationships and authentication (see Point 2 in Section 2.1)
- Auto configuration and self-organization (see Point 6 in Section 2.1)
- Seamless integration of heterogeneous devices and technologies (see Point 8 in Section 2.1)
- Security and privacy (see Point 10 in Section 2.1)

Cluster formation is purely based on physical connectivity and trust, being a local process this formation process does not need support from an infrastructure. As clusters work as LANs they need their own local routing, addressing, self-configuration and other internal mechanisms. The network level should separate communication of PN-nodes from other (foreign) nodes.

This allows for protection on a local scale for communication routing and other self-organizing mechanisms.

Cluster management in such as routing, gateway node selection and local Service Discovery (SD) is restricted to actors inside a cluster. The gateway node can connect to another cluster via an interconnecting structure, such as foreign nodes. It is the task of the gateway node to look for opportunities to establish and create a interconnecting communication, such as a tunnel. Full intra-PN communication can take place after a tunnel has been set up.

To ensure security, intra-cluster communication should be encrypted and have an addressing scheme in such a way that foreign communication can be easily detected and discarded without wasting resources on trying to process the package. To this end, a device needs to be configured to be part of a particular PN. This process, called personalization, provides the device with an attribute for a long-term trust relationship. Personalization will be explained in more detail in Section 2.3.1.

Security is present in each of the abstraction levels. On the network level it needs secure cluster formation and secure inter-cluster communication. Security mechanisms are also needed to access services outside the PN and for providing access to foreign nodes and services. A device has a single owner, however might have multiple users that each have their own PN. Through personalization a device gets (long-term) credentials by which it can authenticate itself and establish security associations with other personal nodes belonging to the owner. An example of a device that might have multiple users is a photo camera used by members of a family. This device then needs to be able to get security credentials for each of these PNs. In order to provide protection against unwanted adoption of a device into a PN the owner must authenticate itself before new security credentials of other users are to be accepted. An example of a device that is part of two PNs is depicted in Figure 2.3.

Devices that implement the PN networking level, i.e., run software implementing a PN Node, are called PN-capable devices. Devices that can be part of multiple PNs are called multi-PN-capable devices. This device must ensure that the several PN nodes running on the device are securely separated, both service- and content-wise.

2.2.3 Service Abstraction Level

The service abstraction level deals with the availability and quality of services and content. It also provides management support of the PN.

This level addresses the service-related aspects of the following concerns:

- Context awareness (see Point 3 in Section 2.1)
- Content management (see Point 4 in Section 2.1)

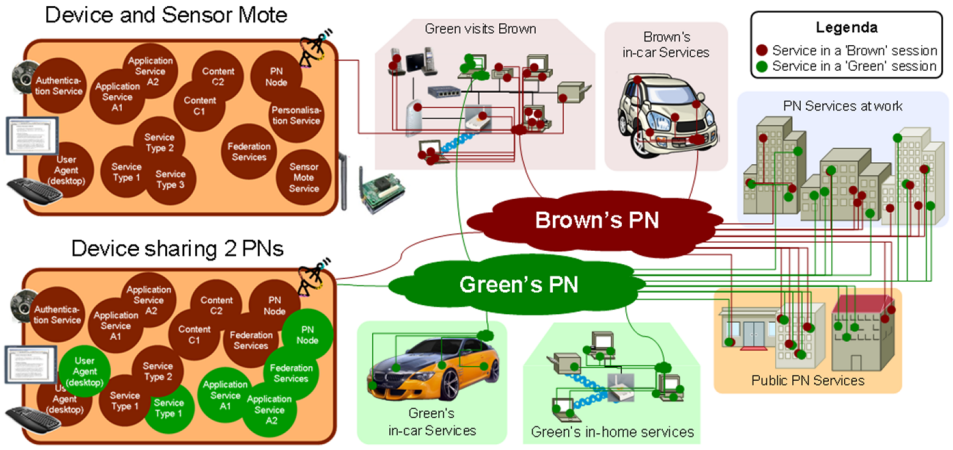

Figure 2.3: Services connected by PNs. Figure reproduced from [65].

- Service and application concerns (see Point 5 in Section 2.1)
- Auto configuration and self-organization (see Point 6 in Section 2.1)
- QoS and reliability (see Point 7 in Section 2.1)
- Seamless integration of heterogeneous devices and technologies (see Point 8 in Section 2.1)
- PN management (see Point 9 in Section 2.1)

A device can have a User Agent (UA) service, which recognizes the user and acts on its behalf. Allowing for multiple persons to use a single device to access their own PNs whilst maintaining separate security domains, is described in [65] as follows.

> The UA communicates with a physical person that it can identify. To this person, the UA is a private 'window to the PN' from which Applications can be started and used. To the PN, a UA is the service that 'electronically impersonates' its PN User. All context information that the UA keeps track of, is called a PN User Session, which includes an identifier for the party fulfilling the role of User and Customer, a link to the UA Service instance it is associated with and much more. UAs may have multiple PN User Sessions, which they may switch between depending on whom they identify as being their current user.

All applications are initialized by the UA (on behalf of a person) or by another application. The UA itself is the only exception as this application is initialized by a person.

Managing all the settings could become cumbersome if the network consists of a lot of entities. The responsibility of the management and administration tasks can be outsourced to a third party, which is called a PN Provisioning Party (PNPP). The PNPP maintains an administration called a PN Provisioning Administration (PNPA). This administration contains registrations of all Device, Nodes, PNs, FedNets, Services, Applications and Content that the PNPP provisions. All of these entities will list an owner and entity-related attributes. The administration also contains all constraints that apply.

The PNPP can provide management consoles to different parties playing different roles. In essence supplying them with party-centric views. A PNPA is functionally centralized. A PN, however, is dynamic, so the realization is likely to be distributed. A means of local availability and synchronization is therefore necessary.

2.3 Network Components

The network abstraction level (discussed in the previous section) is formed by the networking components. The network components are depicted in Figure 2.4 and will be individually discussed in this section. Service components are discussed in Section 2.4.

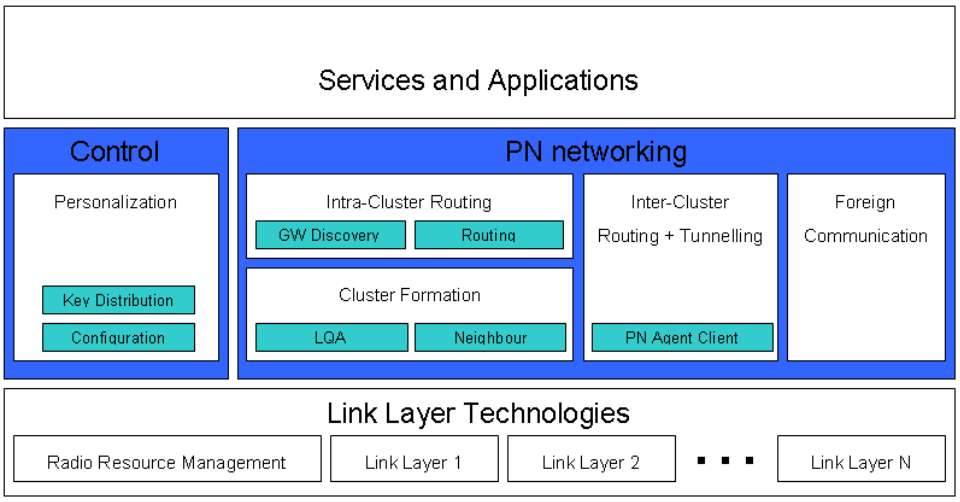

Figure 2.4: Functional decomposition of the Network Abstraction Level. Figure reproduced from [65].

2.3.1 Personalization

The purpose of personalization is to initialize a node for use in a PN. Called imprinting, the new device gets security credentials, such as keys and certifi-

cates, which it needs to establish a security association with the PN. This is needed to be able to get incorporated in (get access to) the PN. A device can be part of more than one PN, imprinting then has to be repeated for each new PN. By default the first imprinter is determined as the owner. For every next imprint the owner must be authenticated. If so set up by the first imprint, this authentication might be done by specified other imprints on the device.

2.3.2 Cluster Formation

A cluster is a set of personal nodes within communication reach of each other without the need of an infrastructure. The task of the cluster formation component is to detect neighboring personal nodes, authenticate them and establish secure connections between them. Not all nodes necessarily have a direct connection to each other, thus a cluster might be a multi-hop network. To optimize routing, this component also handles Link Quality Assessment (LQA).

2.3.3 Intra-Cluster Routing

A cluster might be a multi-hop mesh network consisting of several different communication capabilities. Due to the dynamic nature of a PN, nodes should cooperate in an ad hoc manner to provide the network, security, routing and gateway functionality. A PN network is multi-hop IP based, providing addressing and routing functionality for efficient and secure intra-cluster communication. This component also takes care of gateway node discovery, cluster wide and PN-wide broadcasting [65].

[65] proposes Personal Network Clustering Protocol (PNCP) [50] to be used for clustering the nodes in a PN.

2.3.4 Inter-Cluster Routing and Tunneling

The inter-cluster routing and tunneling component connects disperse clusters into one PN. This enables transparent and secure communication between all the personal nodes of a PN.

Nodes in a cluster need to be able to access nodes in other clusters and perhaps even nodes that are not part of the PN. A node therefore need a way to communicate outside its own cluster. Gateway nodes provide this functionality. These gateway nodes can communicate with foreign nodes. Therefore, gateway nodes must be PN-internally addressable as well as external (world wide addressable). This allows for the gateway node to differentiate between internal and external traffic and for external nodes to connect to the gateway node.

To let clusters find each other, thus forming a PN, the PN-agent is introduced. Much like the Home Location Register (HLR) in GSM and the

home agent in mobile IP. This agent keeps track of where the clusters are, or in other words: how to reach them. The clusters keep the Agent registry up to date with the clusters' connections to the interconnecting structures. The PN agent can therefore be consulted to determine the gateway-node address of another cluster. Figure 2.5 provides an example for such a situation. The PN-agent may also aid in the inter-cluster communication when both clusters are behind a NAT using techniques such as STUN [113] or TURN [111].

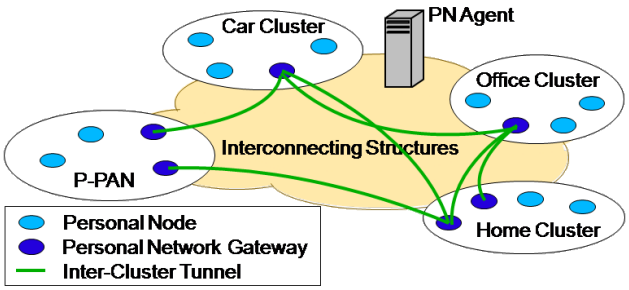

Figure 2.5: Intra-cluster communication with use of a PN Agent. Figure reproduced from [65].

2.3.5 Foreign Communication

This component enables communication with other nodes than those within the PN. These nodes may be part of another PN or be non PN-capable devices. Communication with a foreign node must go via the gateway node.

2.3.6 Radio Resource Management and Link Layers

Although the PN does not really concern itself with the connectivity abstraction level, A PN should support as much link layer technologies as possible and be able to adapt new technologies quickly. Coordination between link layers, such as radio resource management should increase cooperation and performance.

2.4 Service Components

The service abstraction level (discussed in Section 2.2) is formed by the service components. The service components are depicted in Figure 2.6 and will be individually discussed in this section.

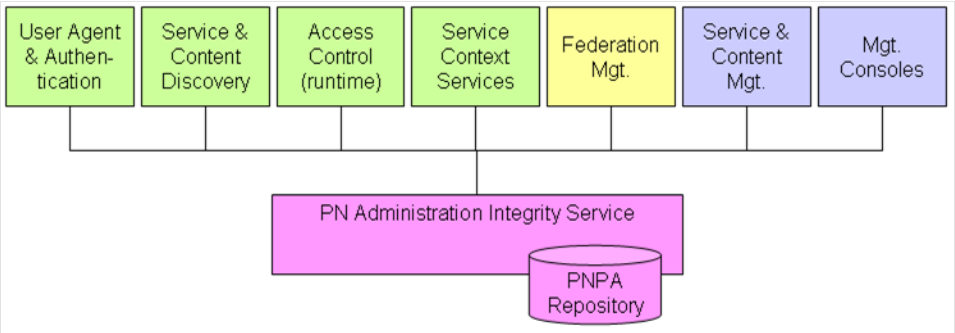

Figure 2.6: Functional decomposition of the service abstraction level. Figure reproduced from [65].

2.4.1 PN Administration Integrity Service

The PN administration integrity service enables PN services to rely on the availability and integrity of the data maintained in the, possibly unreachable, PNPA. In the case where the PNPA resides in a distributed database, the PN administration integrity sees to the distribution and integrity of the data.

2.4.2 User Agent & Authentication

The user agent & authentication functional component performs the following five tasks:

1. Interact with the PN User
2. Provide and maintain the context within which applications can run
3. Provide messaging authentication services
4. Spawn applications upon request of the PN user
5. Create multiple PN users sessions and switch between them

These tasks ensure that the user of a device is the PN owner and that only those PN services are accessed by those that are allowed to.

2.4.3 Service & Content Discovery

This component facilitates “the dynamic (runtime) construction of service collaborations that provide the functionality of a given application in a specific context, by locating appropriate services an content and selecting the ones to be called" [65].

Once it has found applicable services or content it hands them over to the access control component which determines authorization. This component may be absent in a very simple PN-configuration.

2.4.4 Access Control

This component prevents illegitimate use of (application) services and content in the PN. It retrieves a set of services and or content from the service & content discovery components and determines which subset is allowed in the context of the current application session.

Simple forms are Access Control Lists (ACLs) and DAC. More mature implementations are based on RBAC.

2.4.5 Service Context Service

The Service Context Service (SCS) component enables services to perform better or more accurate by providing information based on the context in which the service is running. Three purposes of the SCS can be defined:

1. Generic enhancement of services
2. Enhancement of PN operations
3. Enhancement of application operations

This component may be absent in a very simple PN-configuration.

2.4.6 Federation Management

The concept of FedNets is discussed in whole in the next chapter, including its management component. From a PNs perspective this component takes care of all connection and security issues to participate in and secure a FedNet.

2.4.7 Service & Content Management

According to [65], the purpose and task of the Service & Content Management functional component is "to optimize the availability and usability of Service instances that can be called as well as Content that can be used seamlessly from everywhere within a PN. Since configuration data can also be considered Content, the Service & Content Management component can also be used for the optimization of the availability and usability of configuration data, including the PNPA".

This component may be absent in a very simple PN-configuration. Content is then located where it is stored. Very matured PNs may have their service instances and content moved/cached as to enhance their availability. Service instances and content may also be converted allowing services and/or content to be used in a different environment than originally intended.

2.4.8 Management Consoles

The management-consoles component provides user interfaces to authorized parties and allows them to manage the PN Entities that they have dealings with, from a variety of perspectives, or views and such that they will only see whatever is relevant to them [65].

This component may be absent in very simple PN-configurations.

2.5 Summary

A PN is an ubiquitous network of devices that have the same owner.

Rather than basing, on a strict requirement analysis, [65] based a PN on concerns. These concerns may lead to multiple sets of requirements which can in turn be validated. By defining a PN in concerns gradual development is made easier.

The PN architecture can be viewed on three levels: the connectivity, network and service level. The functionality in each level is divided into components that take care of a specific functionality.

The exact working of the connectivity level is considered to be out of scope for the functionality of a PN and is therefore not addressed in detail.

The network level takes care of the formation of clusters and the PN, providing secure connectivity. The functional components of the network level are depicted in Figure 2.4 .

The service level takes care of the interactions with the user providing authentication and management functionality. The functional components of the service level are depicted in Figure 2.6.

More information on PNs can be found in [65, 66].

Chapter 3

FedNets

Harry signs up at the gymnasium and wishes to use the additional training service software. He needs to authorize the gymnasium to use the sensor data. He takes out his iPhone and opens the FedNet discovery program. The program allows him to search for available FedNets. He selects that he wishes to search for a FedNet in his proximity. His phone offers several options, as the gymnasium clerk created one for Harry that is easily identifiable (e.g., contains Harry's name) and Harry chooses the right one. By accepting the proposed configuration Harry is all set to use the software.

In the previous chapter we introduced the concept of a PN. A Federated Personal Network (FedNet) is a federation of PNs in which content and/or services are shared. Consider a picture taken with a camera belonging to a PN that is to be shared with (the user of) another PN. Temporarily allowing access to a node from the outside the PN imposes certain security risks. For example, the picture is to be shared with a specific other and not everyone connected to the internet. Also the other allowed to access the picture might not be allowed to access other pictures on the camera. A FedNet therefore needs an ACA to ensure that only those resources and services that are allowed to be used can be accessed by the appropriate user and no one else. Figure 3.1 gives an example of a FedNet. ACAs are discussed in the next chapter.

In this document we use the term FedNet. The terms 'PN Federation' and 'PN-F' used elsewhere in literature mean the same and are kept when quoted.

The remainder of this chapter is organized as follows. First types of FedNets are given by example. Next the lifecycle of a FedNet is given, followed by the functional components. Finally interactions during the lifecycle are given.

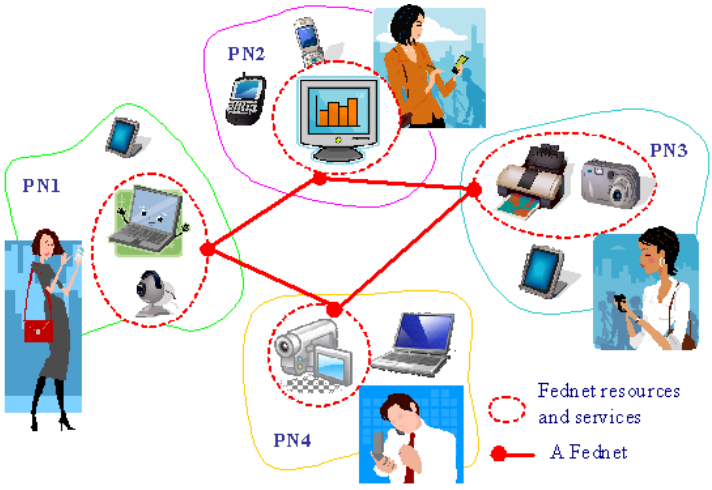

Figure 3.1: Example of a FedNet. Figure Reproduced from [55] with Permission from the Authors.

3.1 FedNet Types

There are many types of FedNets. Type-naming is based on the aspects and properties of the FedNet. Table 3.1 gives a few examples.

Table 3.1: Examples of FedNets.

Criteria	Type of FedNet	Explanation
Composition	Ad hoc	The FedNet is established without the use of an supportive infrastructure, e.g., car-to-car communication.
	Infrastructure-supported	The FedNet is established trough an interconnecting network, e.g., via an access point of the office.
Membership	Static	FedNets that rarely change, e.g., a family network.
	Dynamic	FedNets that have changing members, e.g., wireless community networks and emergency networks.
Distribution	Localized	When federating clusters are close together, e.g., in the same room.
	Distributed	When the participating clusters are far apart, e.g., one is at home and the other resides at the office.
Initiation	Provider-enabled	The FedNet is created by a third party such as a provider or game host.
	User-enabled	The FedNet is (spontaneously) created on the initiative of a PN.
Continued on Next Page...		

Table 3.1: (continued)

Criteria	Type of FedNet	Explanation
Changeability	Static	A FedNet that has requirements and characteristics that remain the same throughout its functioning.
	Dynamic	A FedNet that has requirements and characteristics that change throughout its functioning.
Creation timing	Proactive	A FedNet that is created on forehand.
	Reactive	A FedNet that is created as need arises.
Accessibility	Public	A FedNet that is open for everyone to join.
	Private	A FedNet that is restricted to certain members.
Duration of FedNet functioning	Long-term	A FedNet that exists for a long time before being disbanded. Not to be mistaken for a permanent FedNet
	Short-term	A FedNet that is around for a relative short amount of time.
Duration of member's cooperation	Temporal	A FedNet that only have one evolution cycle during its lifecycle. The lifecycle is explained in the next section. An example is an car-to-car network.
	Permanent	A FedNet that has members that last for more than one lifecycle.
On-purpose or on-opportunity formation	Purpose-driven	A FedNet created to achieve a certain goal.
	Opportunity-driven	A FedNet created when interesting scenario arose.
Visibility of member list	Anonymous	A FedNet in which members are unaware of each others identity.
	Transparent	A FedNet in which members are, or can be, aware of each others participation and identity and contributions.
Keeping history of previous events	Stateful	A FedNet that keep track of previous activities, experiences of the members, trust and reputation ratings.
	Stateless	A FedNet that does not keep state information.
Application type	Entertainment	A FedNet created to support some form of entertainment.
	Educational	A FedNet created to support education.
	Business	A FedNet created to support a business.
	Scientific	A FedNet created to support scientific work.

3.2 The FedNet Lifecycle

Harry's coach has some time between training sessions and turns on her laptop. She connects to Harry's body hub to get information on his training activities, while he is at work. Having looked at the data, she writes Harry a message about his improvements. She also updates Harries training exercises to better suit his training improvements. The next time Harry looks at the scheduled exercises he sees that the adjustments have already been Incorporated. Just as the coach stated in her message.

According to [56] a FedNet lifecycle consists of four phases: the initial, formation, operation and dissolution phase. These phases and its transitions are depicted in Figure 3.2.

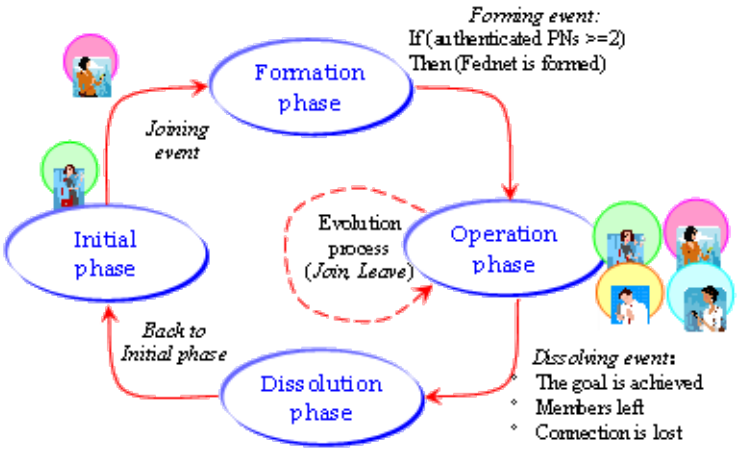

Figure 3.2: The FedNet Lifecycle. Figure Reproduced from [56].

The basic entities that take part in forming a FedNet are the FedNet Agent (FA) and FedNet Manager (FM), which we will explain in more detail in the next section. In this section we go into detail on the interactions on the formation, evolution, service access and dissolution phases.

3.2.1 Initial Phase

In the initial phase, a PN creates membership profile containing all relevant data. The PN preforms a discovery process to find at least one other FedNet to join whereas from a FedNet point of view discovery is the detection of a new member for the FedNet. After discovery, a joining event on one of the found FMs triggers the formation phase.

3.2.2 Formation Phase

In the formation phase first-level authentication and authorization processes take place. When the members (at least two) are authenticated and the necessary credentials and tokens are exchanged a FedNet enters its operation phase.

The initialization phase and the formation phase of the lifecycle are combined into interactions, shown in Figure 3.3. All steps and substeps must be performed, except for Finding step in which only one substep needs to be successful. Though all three means of finding must be supported.

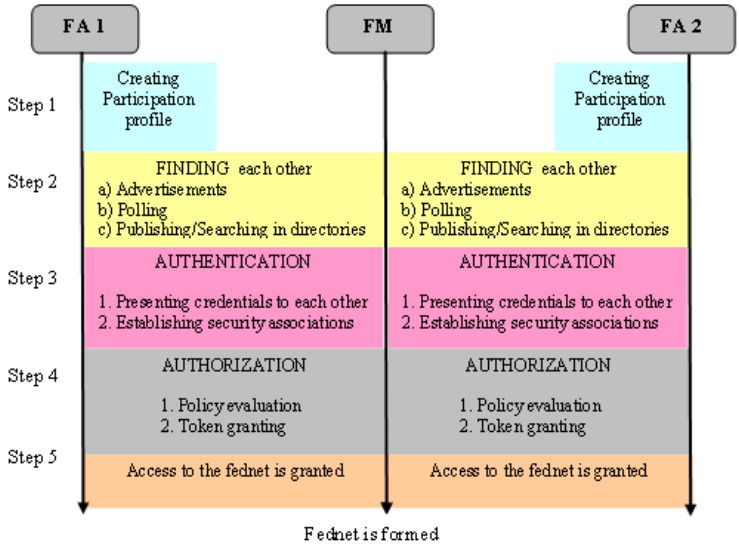

Figure 3.3: Interaction Steps at FedNet Formation. Figure Reproduced from [56].

3.2.3 Operation Phase

In the operation phase a FedNet is in use. This means that a connection has been established and that resources and services are shared and can be used between the FedNet members. The process of members joining and leaving is called evolution and makes a FedNet dynamic.

The operation phase is divided in two interaction overviews: the evolution and service access, this corresponds to first level and second level access control respectively.

3.2.3.1 Evolution

In the evolution stage a member can be added to the FedNet by the steps depicted in Figure 3.4. Except from the Finding step, all steps and substeps

must be performed. In the Finding step only one means of successfully finding each other will suffice, although all three means must be supported.

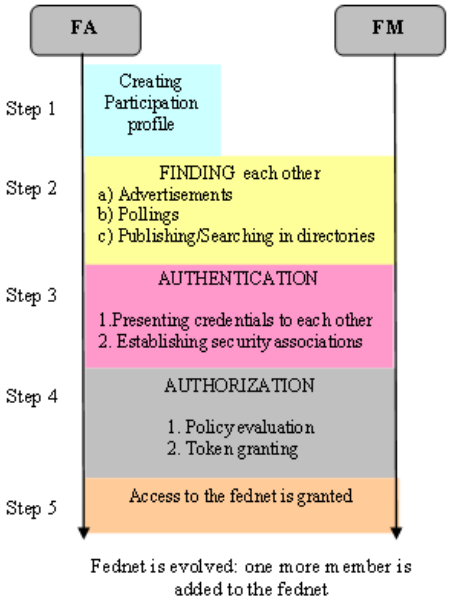

Figure 3.4: Interaction Steps at FedNet Evolution. First Level Access Control Steps. Figure Reproduced from [56].

It is also possible for a member of a FedNet to disband from the FedNet it is taking part in by taking the steps depicted in Figure 3.5.

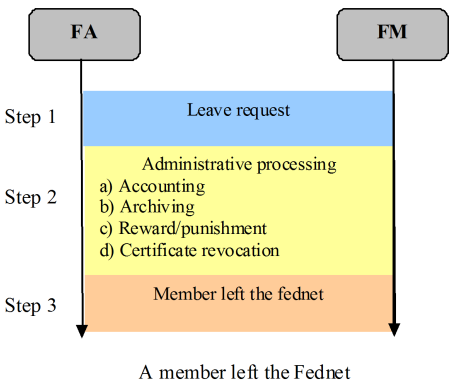

Figure 3.5: Interaction Steps at FedNet Evolution. Figure Reproduced from [56].

The administrative processing by the FM and FA can consist of accounting, archiving and reward/punishment. After these, if applicable, have been handled, all concerning security associations must be flushed. If a member

leaving a FedNet causes the FedNet to fail one of its constraints, such as consisting of at least two PNs a dissolution process takes place. The dissolution interactions will be discussed after the next interaction overview.

3.2.3.2 Service Access

When the FedNet is operational the services that are offered in the FedNet are searchable. The FM relays information on the offered services such as realtime availability and relevant connection settings. With this information the shared service can be used. At which point the requestor enters the second level of access control ensuring that the relevant policies are applied.

These steps are depicted in Figure 3.6. All steps and substeps must be performed.

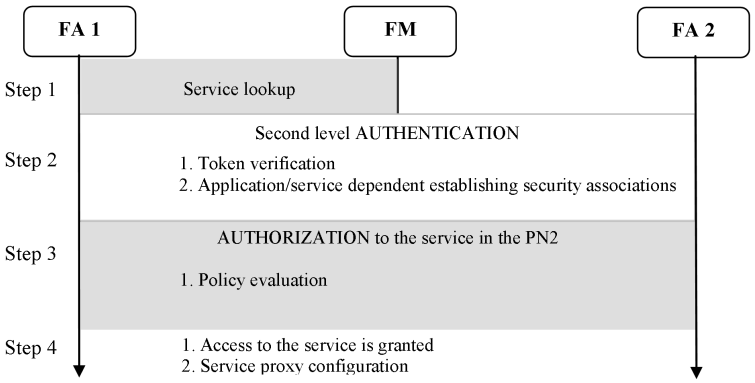

Figure 3.6: Second Level Access Control Steps. Figure Reproduced from [55] with Permission of the Authors.

3.2.4 Dissolution Phase

In the dissolution phase the FedNet is terminated. Triggered by a dissolving event, there are three types of dissolution: graceful, forced and abrupt. The first takes place when the objective of the FedNet has been met. The second takes place when requirements are no longer met —e.g., not enough members, impossible to reach objective, insufficient resources. The last one, abrupt dissolution, is the only one that happens without prior notification and can occur when, for example, the network connectivity is lost and the policy is such that it does not allow for an evolution iteration.

The steps are depicted in Figure 3.7. The administrative processing by the FM and FA can consist of accounting, archiving and reward/punishment. After these have been handled, all concerning security associations must be flushed.

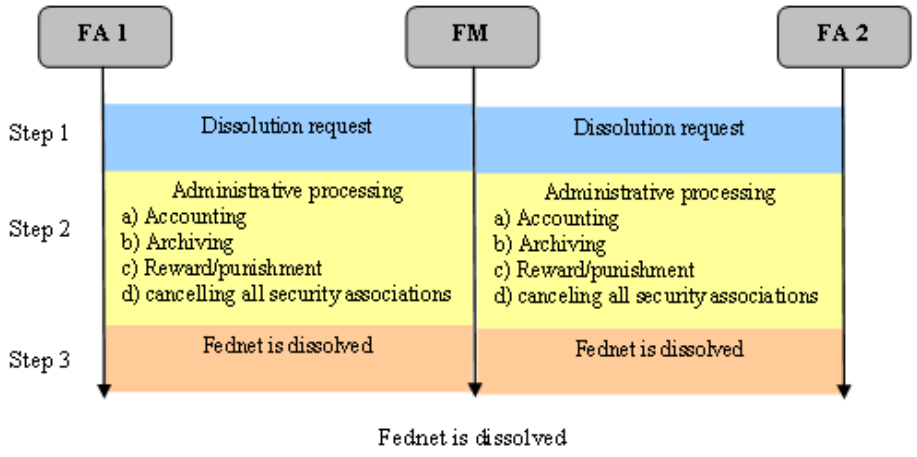

Figure 3.7: Interaction Steps at FedNet Dissolution. Figure Reproduced from [56].

3.3 Architecture

This section states the architectural components and what they do for a proxy based FedNet as defined by [55] and the the PNP2008 project [56].

Subsection 3.3.1 provides an overview of the components and data sets. The following subsections go into detail on each of these.

3.3.1 Architectural Components

To be able to federate, PNs need to have the following functional components:

- FedNet Manager (FM, one per FedNet);
- FedNet Agent (FA, one per PN);
- Gateway (GW, one or more per PN);
- Service Proxy;
- Service Management Node (SMN, one per cluster);
- Service Discovery (SD, one per FedNet);
- FedNet services (a set of PN services);

Also the following data sets needs to be stored:

- FedNet Access Policys (FAP, one per FedNet);
- Service Access Policys (SAP, one per PN);

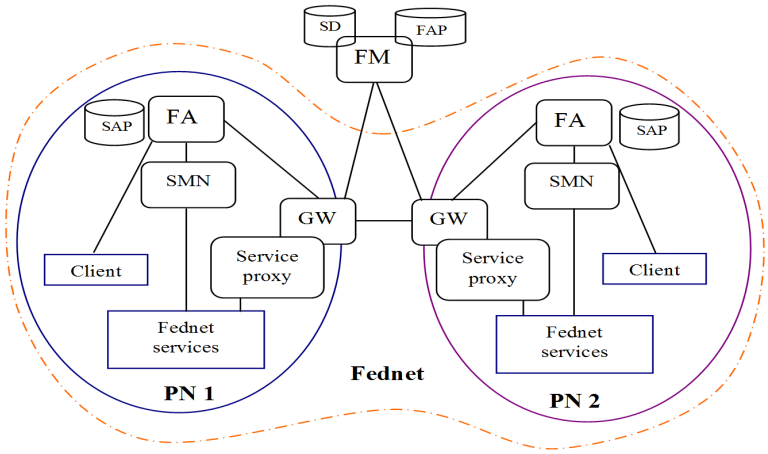

Figure 3.8: Basic Proxy-Based-Architecture of a FedNet with an External FM. Figure Reproduced from [55] with Permission from the Authors.

The functional components listed above are depicted in Figure 3.8) in a FedNet using an external FM.

In principle FedNet members are equal peers, however, the amount of shared data and services may be asymmetrical. It stands to reason to group above components and data sets that have similar occurrence. Such as the FM, SD (including FedNet services) and FedNet Access Policy (FAP). As well as the FA, GW (including service proxy) and Service Access Policy (SAP).

3.3.2 FedNet Manager

We use the definition of a FM as defined by [65]:

> "The FM is a central functionality in authentication and access control process within the PN-F. Its main tasks are PN-F management, authentication and access control within the PN-F domain. It manages access control decisions to the PN-F the registering new members, managing the member lists, provides the directory service to the PN-F members. This includes also maintaining the list of services in the PN-F. The FM component is located in the Owner's PN. As an Owner, this functionality is required to initially authenticate prospective members and in their admittance to the federation. Another role of the FM is to construct and manage the profile of the PN-F."

The FM performs the following three functions:

1. Management function
   Which function is "to create FedNet profile and advertisements; to

manage the formation, evolution and dissolution of a FedNet, to maintain the list of FedNet members and their contributions, experiences during the FedNet operation" [55].

2. Service directory look-up function
   Which function is “to create FedNet profile and advertisements; to manage the formation, evolution and dissolution of a FedNet, to maintain the list of FedNet members and their contributions, experiences during the FedNet operation" [55].

3. FedNet Access control function
   Which function is “to produce decisions on the access control to the FedNet; to issue membership credentials for FedNet members" [55].

The FM can either be located outside the architecture of the PN (external FM, see Figure 3.8) at a third party such as an PN provider or inside a PN (internal FM, see Figure 3.9). Useful in cases such as on-line gaming or very large FedNets requiring dedicated managing.

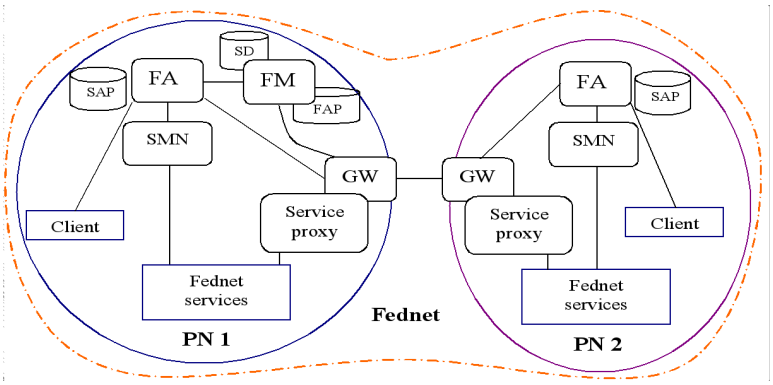

Figure 3.9: Basic Proxy-Based Architecture of a FedNet with an Internal FM. Figure Reproduced from [55] with Permission from the Authors.

3.3.3 FedNet Agent

We use the defintion for FA stated in [65]:

> “Attached to the Cluster Gateway Node is the FedNet Agent (FA), which is used for all the management needs of the PN while registering with and participating in, a federation. The predominant role of the FA is in authentication and access control to the Services and Content of the PN. As a PN, the FA’s functionality is also used to authenticate the owner of the federation and during the member association procedure is used in the authentication of the other federation members and in the configuration

> of its Service Proxy for that particular member. Another role of the FA is to construct and manage the PN-F membership profile for the member used in the federation." [65]

The FA resides inside each individual PN and performs the following three functions [55]:

1. Management function
   Which function is "coordinating PN services to which a PN can grant a temporal access within a FedNet, managing the participation of a PN in a FedNet, creating a participation profile, joining and leaving a FedNet".

2. Service access control function
   Which function is "to produce access control decisions to PN services".

3. Service proxy configuration function
   Which function is "to configure a requested service proxy to a particular member in a FedNet".

3.3.4 Gateway

The gateway node is a device inside a PN capable of communicating with foreign nodes and performs translating services to its internal nodes for external communication. Therefore it has a publicly addressable interface besides its internal address.

3.3.5 Service Proxy

"The PNs also need a way to offer and use Services and thereby also a mechanism to enforce the authentication and access control. For this usage, they rely on a Service Proxy (SP), which is a functional component located at the Gateway Node of a Cluster of a PN participating in a federation. Its role is to restrict access to other federation members who have properly associated themselves to the owner of the SP. They also maintain the perception to the PN that it is only handling Intra-PN services while in fact offering services to and utilizing the services of, other PNs in the federation. This is achieved by using service handlers located at the SP. These service handlers are applications that mimic the client/server functionality of the required service. At minimal functionality, the handlers would just forward the received message, however the handlers could be made more complex to overcome possible interoperability issues between certain client and servers. The main advantage of the service proxy though, is the fact that the security of the PN Clusters is located at the border of the Clusters, allowing the internal Nodes of the PN to operate in a federation without the need for any additional functionality." [65]

The service proxy functional component mimics the behavior of a service and acts as an intermediary between client and service. This enables a separate security domain within a FedNet security domain, preventing alien traffic inside the PN. Using a service proxy enables more lightweight PN nodes, because only gateway nodes need extra access control functionality [55].

3.3.6 Service Management Node

The Service Management Node (SMN) is a centralized service discovery and management entity providing a specific cluster with information on the available services in the PN combined with the location of the handling service proxy. The SMNs communicate on a P2P basis to the SMNs of other clusters in the PN. [55]

3.3.7 A FedNet Service

This component contains the (subsets) of services offered to share with other PNs. There is a distinction between a common and a specific service. The common service can be accessed by any member of the FedNet upon providing membership credentials. The specific service requires a specific access control procedure [55].

3.3.8 A FedNet Client

A client is a process, application or personal node requesting a FedNet service [55].

3.3.9 Service Discovery

Information regarding the participating FedNet members and their offered services are listed in the Service Discovery [55].

3.3.10 FedNet Access Control Policies

Information on the network-level (first-level) access control is stored in the FedNet access policies data set. The FM evaluates these when a new member wishes to join the FedNet. These policies contain the rules, constraints and statements on the access to the FedNet and are defined by the initiator of the FedNet [55].

3.3.11 Service Access Control Policies

Information on the service-level (second-level) access control is stored in the service access control data set. The FA evaluates these when a service is

accessed. These policies contain the rules, constraints and statements on the access to PN services and are defined by the owner of the service [55].

3.3.12 FedNet Services

FedNet services are the subset of all PN services that are shared in a FedNet. They are divided in two classifications: common and specific services. A *common* service needs only FedNet membership credentials, whereas a *specific* service needs additional (second level) access control. Thus, both type of services need FedNet access control and only specific services need service access control. The owner defines to which type a service belongs. Examples of common services are a forwarding service, a display service, a printing service, internet access, storage facilities, etc. An example of a specific service is file sharing [55].

3.4 Summary

Having a lifecycle of four phases — the initial, formation, operation and dissolution phase — a FedNet is a collaboration of PNs that share content and/or services in order to achieve a common goal. In Section 3.3.1 the following functional components needed to form a FedNet are introduced: the FM, FA, GW, SP, SMN, SD and FedNet services. Datasets containing the FAP and SAP are also needed.

Chapter 4

Access Control Architectures

At the medical center, Harry is getting his new sensor. Sleep deprived as he is, he worries about his privacy. He informs the doctor that he would not like his boss to know he is having trouble sleeping. The doctor kindly explains that the sensor is quite sophisticated and has all the security features to ensure privacy. Harry personalizes the sensor and joins the FedNet, thereby allowing the medical staff to monitor his sleep.

"In the context of network security, access control is the ability to limit and control the access to host systems and applications via communication links. To achieve this, each entity trying to gain access must first be identified, or authenticated, so that access rights can be tailored to the individual". [121]

In order to compare ACAs it helps to understand why there is need for them. To that end, Section 4.1 is dedicated on security threats and Section 4.2 on security definitions. The reader is assumed to be familiar with the basic concepts of access control. Access control itself will not be covered in depth, the aim of the next section is to provide a short outline. Next a study of available security ACAs that might be applied in FedNets is provided in Section 4.3. Section 4.4 states selection criteria for selection of a suitable architecture in FedNets. Section 4.5 evaluates the different architectures. Section 4.6 discusses which of the evaluated architectures is most suitable. And finally Section 4.7 summarizes this chapter.

4.1 Security Threats

Data being sent over any communication means are susceptible to a number of threats. In the previous chapters we discussed several security requirements that relate to the PN and FedNet concept. In this section we provide

an overview of the security threats that apply to data communications in general.

Some of these attacks have been applied to computer networks since the time of ARPANET — one of the first well documented attacks is described in [127]). Over the years, more and more attacks occurred, this eventually led to well documented specifications of security. For instance [62, 6].

1. **Passive attack**
   The passive attack, also known as the eavesdropping or sniffing, is an attack where the adversary learns the contents of the message someone (specific) is sending. Confidential and or personal information might fall into the wrong hands. Depending on the contents, all kinds of malicious events might happen, such as an adversary learning your username and password that you sent over the network to log in somewhere. Data confidentiality protects against a passive attack.

2. **Active attack**
   By altering the data being sent (modification), applications might perform a different behavior then intended. A bank transaction suddenly being sent to the different account number, probably one controlled by the adversary, is also clearly unwanted. An adversary capturing your bank transfer assignment and offering it again to the bank, a so-called replay attack, ending up in a lower outstanding balance is clearly unwanted. Data integrity takes care of active attacks.

3. **Identity theft**
   The adversary could perhaps lie to the bank, claiming to be you and issue a bank transfer, effectively giving him access to your funds. Using authentication this can be avoided. By proving to have knowledge of some secret that is only shared between you and the bank, the bank can determine whether it is communicating with the person he claims to be. However, if the authentication mechanism is compromised, because of a sniffed username/password combination this will not help. Authentication is therefore often combined with data confidentiality.

4. **Traffic analysis**
   Traffic analysis is a kind of passive attack where not the contents of the message itself is compromised, but the routing information in the message. This can be viewed as the mail man needing to read the destination address in order to deliver it, however perhaps learning that the package was sent from a particular sender or deducting what the contents of the package are (e.g., by size and weight). One can see how this is a problem if one is burglarized the next day and the shipped goods are gone. One can generally trust the mailman. The risk

increases however, when others can also look at the package and the data associated with it — as is the case with wireless communications.

For IP these include source and destination address and other headers, such as — in the case of TCP or UDP — source and destination port [104].

5. **Privacy**
When carrying a device capable of wireless communication, one might not want one's personal identity unknowingly being revealed to other parties. Many devices are equipped with a unique identifier, such as a MAC address, which is needed in order to address messages to them. This identifier might be correlated to an identity. [143] proposes the use of random addresses by devices. This of course generates more overhead. It is clear however, that authentication protocols need to be in place that do not leak identity of the user to an eavesdropper. Nor should the identity be provided to an anonymous service.

Identification privacy and location privacy are usually not an issue in computer security, as the connection is static and an adversary needs to put significant effort in obtaining ones exact location or monitor traffic. For example a court order is usually needed to obtain the actual identity/location information from an ISP and one has to dig up a cable in order to start sniffing. In any kind of mobile communications however, everybody that obtains the right equipment can sniff data that is being transmitted nearby. Combine this with the fact that a PN is user centric and it is clear that identification and location privacy is an issue.

[143] derives four requirements for identification and location privacy:

(a) **User identity confidentiality:**
- the property that the user identity of a user to whom a service is delivered cannot be eavesdropped on the radio access link.
- the property that the user identity of a user to whom a service is delivered will not be given to the PAN [Personal Area Network] component acting as server for the user requests in the case of anonymous service access.

(b) **Component identity confidentiality:**
In the context of a PAN, many components are expected to have integrated a hardware identification... By means of correlation between user and PAN components, such a component identity could be used to identify its user

if not protected appropriately. Component identity confidentiality includes:

- the property that the identity of a PAN component to whom a service is delivered cannot be eavesdropped on the radio access link
- the property that the identity of a PAN component to whom a service is delivered will not be given to the PAN component acting as server for the user requests in the case of anonymous service access.

(c) **User location confidentiality:**

- the property that the presence or the arrival of a particular user in a certain area cannot be determined by eavesdropping the communication on the radio access link. However, sometimes a user may wish to inform the other components of her arrival. In such a case, user location confidentiality might not be required and user or component identity information might be sent on the user's request.

(d) **User untraceability:**

- the property that an intruder cannot deduce whether different services are delivered to the same user by eavesdropping on the radio access link.

6. **Depletion attack**
   A depletion attack is a kind of denial-of-service attack that aims to drains the battery of a mobile device. While this is certainly important in military sensor networks, operation of most devices used by the public are not as critical. Some however are critical — e.g., a pacemaker — and need to be withstand depletion attacks.

7. **Costs**
   There is always a tradeoff between (usable) security and cost. In general, costs increase with better security. A user naturally wishes to have (good) security, however in some cases the cost of the added security makes a product to costly to purchase. The balance depends on the context and the preferences of the user (e.g., [140] shows a significant percentage of unencrypted and poorly encrypted WEP WLAN access point). For recommendations on key sizes and their strength see [94, 34].

4.2 Security Definitions

In the previous section, several security threats have been introduced. In this section we will discuss general techniques to counter these threats.

Security encompasses more than verifying ones identity stating what access rights that identity has. For example, for a spy it is important that others do not learn his identity when he communicates with his agency, also he may wish that the contents of the communication are not to be learned by others as it is sensitive information and the content might lead to exposure of his identity. These are forms of *confidentiality*, both on identity and data. However, the agency needs to verify that it is communicating with its spy and the spy needs to be able to authenticate himself without leaking its identity.

For the bank it is important that the customer cannot claim not to have send a transfer order executed on his behalf. *Non-repudiation* guarantees that the order is sent and/or received by the claimed identity.

The following subsections will go into more details on the techniques used for the above.

1. **Keys**
   Keys are used for the techniques listed below. That makes them an important part in security.

   There are many encryption methods, roughly dividable in two groups. The first are based on a symmetric key, i.e., the key to encrypt the data is the same as the key to decrypt the data. The second group is based on asymmetric keys, whereby two keys reverse each others encryption. Usually one of these keys is made public, the other is kept private.

   We will discuss keys, key distribution and key derivation in more detail in Chapters 6 and 7.

2. **Authentication**
   Authentication is defined by [6] as “the act of verifying a claimed identity, in the form of a pre-existing label from a mutually known name space, as the originator of a message (message authentication) or as the end-point of a channel (entity authentication)".

3. **Access control**
   Access control — also referred to as authorization — is defined by [6] as “the act of determining if a particular right, such as access to some resource, can be granted to the presenter of a particular credential".

4. **Confidentiality**
   Confidentiality provides privacy. Data confidentiality is defined by [62] as “the property that information is not made available or disclosed to unauthorized individuals, entities, or processes". User identity confidentiality is defined by [143] as “the property that the user identity of a user to whom a service is delivered cannot be eavesdropped on the radio access link". Confidentiality is sometimes referred to as privacy.

To ensure confidentiality communications are encrypted. Encryption, or encipherment, is the cryptographic transformation of data to produce ciphertext [62] that is not readable by anyone except those possessing a key. This key is used to decipher or decrypt the ciphertext to the original data.

A *secret-handshake-mechanism* can be used to establish a connection without yielding ones identity. Proof of knowledge of a secret key is given before the actual identities are exchanged and thus revealed. Secret-handshake-mechanisms are provided in [11, 12]. Component-identity-confidentiality might be achieved by using temporary identifiers such as a MAC address.

5. **Data integrity**
   Data integrity assures that messages are received as sent, with no duplication, insertion, modification, reordering, or replays [121, Section 1.4]. In connectionless systems only modification can be guaranteed. Integrity is provided by adding a digest to the data that is send. The digest is the result of a cipher that takes the data and a key as input and calculates a x-byte value, x depending on the cipher used. The cipher is sometimes referred to as a hash function.

   The receiving party can check for modification by performing the same calculation over the received data and compare his results with the digest send with the data. An adversary does not have the required key to calculate the corresponding digest to go with the altered data. A thorough analysis of integrity ciphers, fall out of scope of this Thesis. However, we state several requirements in Section 6.2.2.

6. **Non-repudiation**
   Non-repudiation ensures proof of delivery and sending. Neither sending or receiving party can deny the transmission of the message. Non-repudiation is based on asymmetric cryptography. The private key, which is believed to be only known by one user, is used to digitally sign the message. Like a normal signed document, the text itself is readable, but proof of identity is added. Therefore a hash generated over the data (document) is encrypted using the private key. Note that the hash function itself does not rely on a key [121, Section 1.4].

4.3 Security Access Control Architectures That Can Be Applied in FedNets

In this section we will discuss several ACAs that might be suitable for use in the context of PNs and FedNets.

4.3.1 AAA

In computer science AAA stands for Authentication, Authorization and Accounting. Authentication, Authorization and Accounting (AAA) is sometimes combined with Auditing and is then accordingly named AAAA. The AAA architecture provides a generic framework for authentication authorization and accounting. An AAA server evaluates and where necessary propagates access control requests, using policy and event repositories, Application Specific Modules (ASMs) and other AAA servers [29].

The AAA authorization framework discussed in [134] focuses solely on the authorization (access control) in AAA. It defines message sequences for use in a single domain case, roaming and distributed services. There are three different sequences: the agent, pull and push sequence. Each shall be discussed shortly. When the organization that authorizes and authenticates the user is different from the organization providing the service — i.e., roaming. — the home user organization needs to be consulted. In distributed systems, multiple organizations provide a part of the functionality (e.g., QoS over a multi-organizational connection). In a distributed system each participating organization must properly set up their own service equipment by using its own message sequence (one organization might use a pull sequence whilst another uses an agent sequence).

In the *agent sequence* the AAA server functions as an agent between the user and the service (see Figure 4.1 and 4.2).

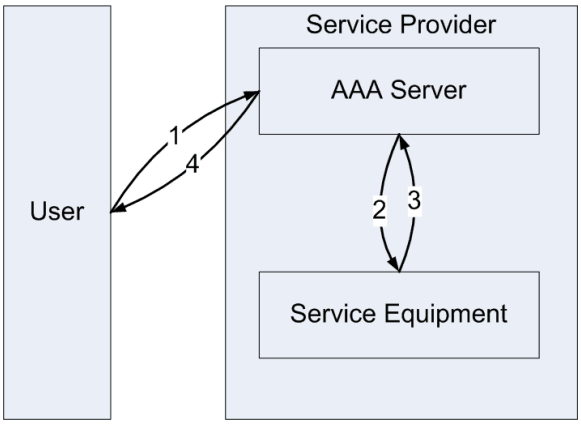

Figure 4.1: Agent Sequence in Single Domain, Figure Reproduced from [134]

In the *pull sequence* the service equipment acts as agent and forwards the request to the AAA server which replies the appropriate response to the service equipment (see Figure 4.3 and 4.4).

In the *push sequence* a ticket or certificate verifying access to a particular service is obtained by the user directly from an AAA server (see Figure 4.5). The ticket or certificate gets relayed by the user to the service equipment

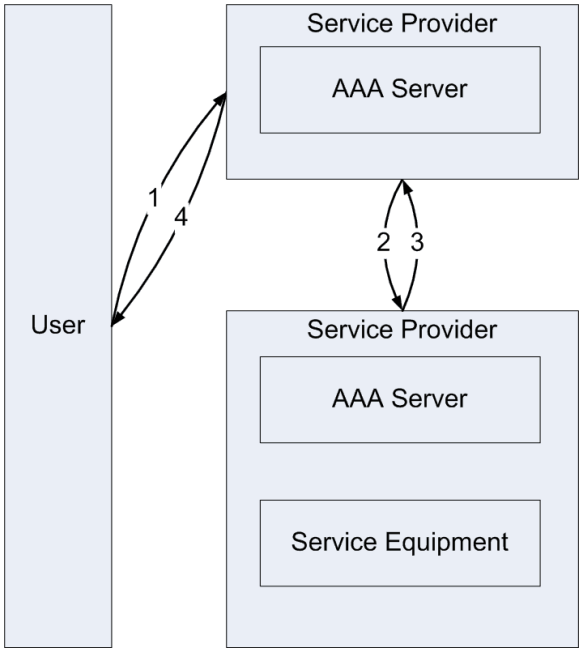

Figure 4.2: Agent Sequence with Roaming, Figure Reproduced from [134]

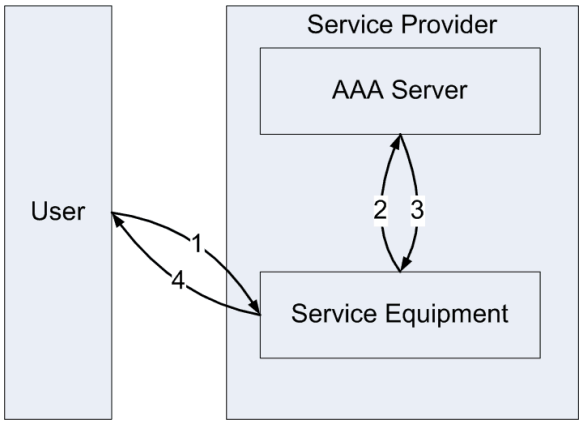

Figure 4.3: Pull Sequence in Single Domain, Figure Reproduced from [134]

that can now grant access based on the provided ticket or certificate.

According to [134], one view of an authorization is that it is the result of evaluating policies of each organization that has an interest in the authorization decision. Each organization may have his own administration style. Each independent policy must be retrieved, evaluated and enforced.

Policies are stored in a policy repository by or on behalf of the organization that requires them. The retrieval is typically done by the administration that defines the policy of an agent acting on its behalf. In order to evalu-

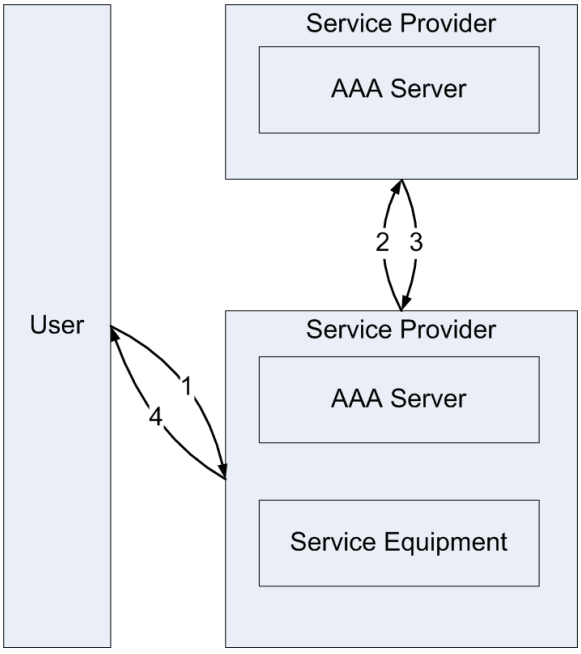

Figure 4.4: Pull Sequence when Roaming, Figure Reproduced from [134]

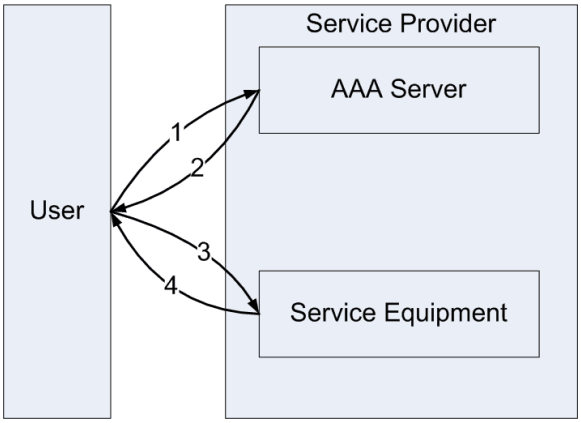

Figure 4.5: Push Sequence in Single Domain, Figure Reproduced from [134]

ate the policy, information referred to in the policy must be available. If the needed information does not reside locally in the organization evaluating the policy, a remote administration must be queried for the needed information or the policy has to be forwarded and the remote administration must perform the actual evaluation.

Enforcement is typically done by the service provider on the service equipment (being the target of the policy). The service equipment that gets a policy from an AAA server should be set up to "enforce" the policy or, if

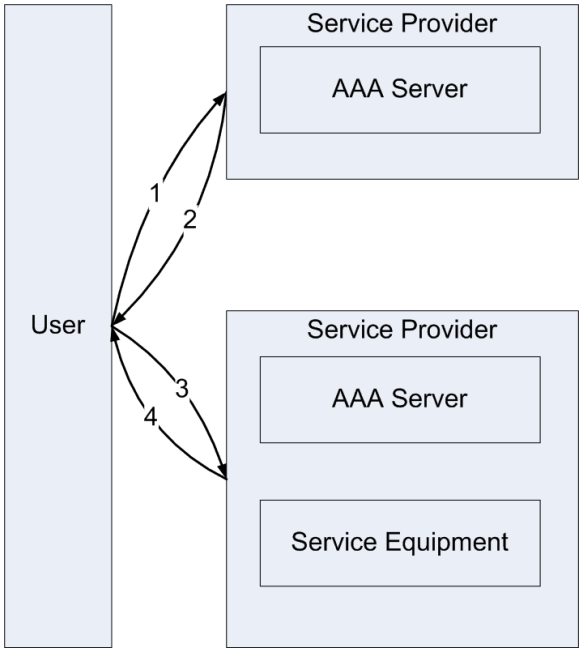

Figure 4.6: Push Sequence when Roaming, Figure Reproduced from [134]

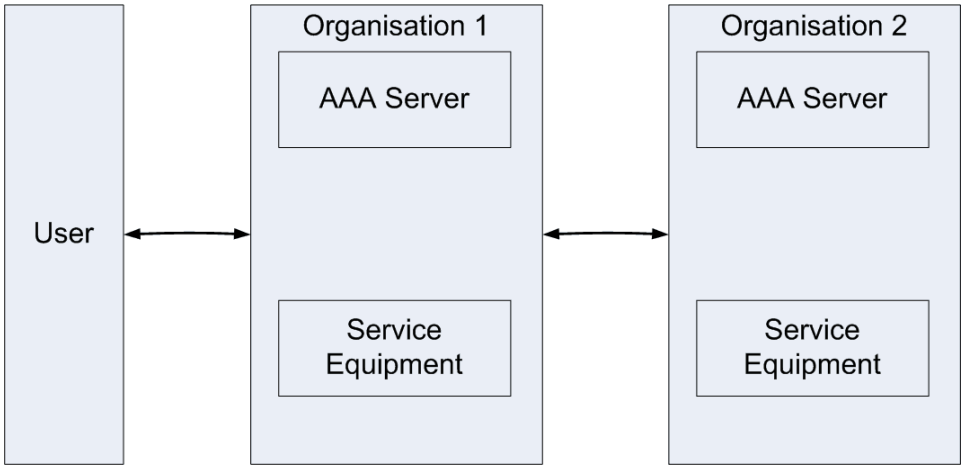

Figure 4.7: Distributed Message Sequence, Figure Reproduced from [134]

not able to enforce, reject the request. A policy is retrieved at a Policy Retrieval Point (PRP) from a Policy Repository, evaluated at a Policy Decision Point (PDP) or Policy Consumer and enforced at a Policy Enforcement Point (PEP) or Policy Target. Information against which policy conditions are evaluated are accessible at Policy Information Points (PIPs) [134].

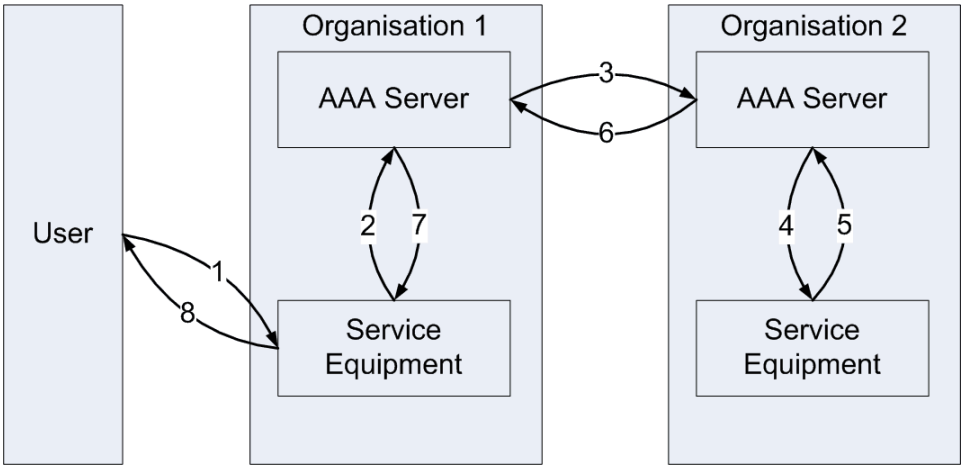

Figure 4.8: Possible Distributed Message Sequence, Figure Reproduced from [134]

4.3.1.1 RADIUS

Remote Authentication Dial In User Service (RADIUS) is a AAA protocol for carrying authentication, authorization and configuration information between a Network Access Server (NAS) that wishes to authenticate its links and a shared authentication server. RADIUS incorporates the client-server model in which a NAS operates as a client of RADIUS. The RADIUS server authenticates the user and returns all configuration information required. A RADIUS server can act as a proxy-server to other RADIUS servers, enabling roaming capability (see Figure 4.10). A variety of user authentication methods are supported by the RADIUS server (e.g., provided with the user name and original password given by the user, it can support PPP Password Authentication Protocol (PAP) or Challenge-Handshake Authentication Protocol (CHAP), UNIX login and other authentication mechanisms) [108]. Figure 4.9 provides an example of a RADIUS server.

Although transactions between a client and RADIUS server are authenticated using a shared secret that is never send over the network [108], the user credentials is send using an MD5 hashing algorithm to obfuscate passwords. As MD5 is no longer considered secure [105] it is wise to send RADIUS data over an encrypted channel, such as a tunnel that utilizes a more secure encryption technique.

RADIUS supports the Extensible Authentication Protocol (EAP) (see [5]), allowing several authentication protocols to be used in conjunction with RADIUS. RADIUS also support accounting, which is explained in [107]. RADIUS runs in the application layer basing its connectivity on UDP.

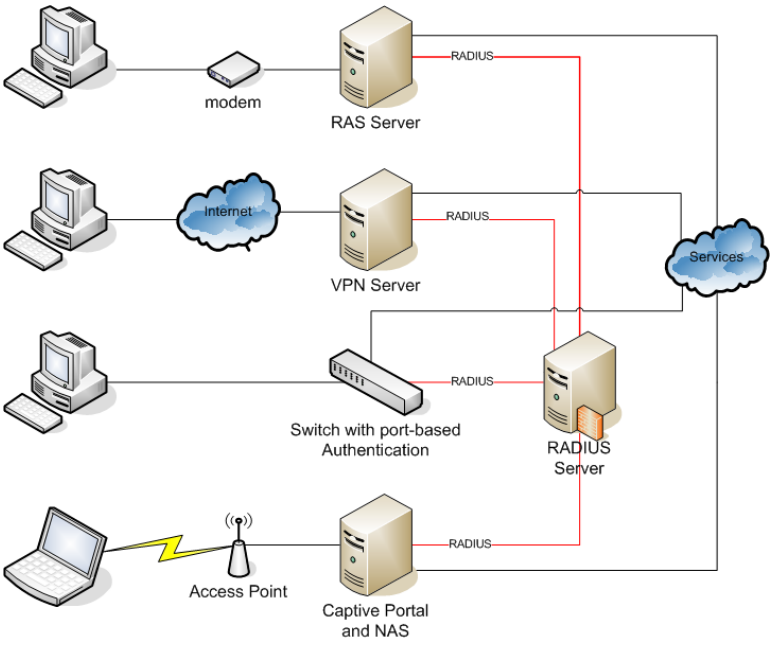

Figure 4.9: An Example of RADIUS Use in an Enterprise. Figure Reproduced from [110].

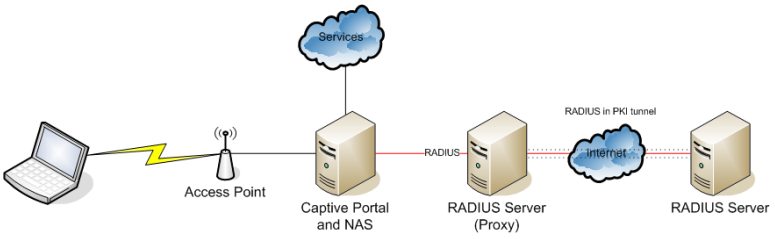

Figure 4.10: An Example of RADIUS Roaming. Figure Reproduced from [110].

4.3.1.2 Diameter

The Diameter base protocol [21] is intended to provide an AAA framework for new access technologies. These access technologies, including wireless, DSL, Mobile IP and Ethernet, routers and NAS, have increased in complexity and density, putting new demands on AAA protocols which were not taken into account when RADIUS was initially deployed. Like RADIUS, Diameter supports EAP [5].

Diameter provides failover, transmission-level security, reliable transport (by using TCP), agent support, server-initiated messages, auditabil-

ity, transition support, capability negotiation, roaming support and peer discovery and configuration. Diameter also facilitates delivery of Attribute Value Pairs (AVPs), capabilities negotiation, error notification, extensibility (through addition of new commands and AVPs) and basic services necessary for applications (such as handling of user sessions or accounting). Diameter AVPs carry specific authentication, accounting, authorization, routing and security information as well as configuration details for the request and reply [21].

All data delivered by the protocol is in the form of an AVP. Some of these AVP values are used by the Diameter protocol itself, while others deliver data associated with particular applications that employ Diameter. AVPs are used by the base Diameter protocol to support the following features [21]:

- Transporting of user authentication information, for the purposes of enabling the Diameter server to authenticate the user.
- Transporting of service-specific authorization information, between client and servers, allowing the peers to decide whether a user's access request should be granted.
- Exchanging resource usage information, which MAY be used for accounting purposes, capacity planning, etc.
- Relaying, proxying and redirecting of Diameter messages through a server hierarchy.

4.3.2 IEEE 802.1X

The Institute of Electrical and Electronics Engineers (Institute of Electrical & Electronics Engineers (IEEE)) 802.1X is an IEEE standard for port-based network access control. It uses the physical access characteristics of IEEE 802 LAN infrastructures in order to provide a means of authenticating and authorizing devices attached to a LAN port that has point-to-point connection characteristics and of preventing access to that port in cases in which the authentication and authorization process fails. A port in this context is a single point of attachment to the LAN infrastructure [58].

Only when authenticated and authorized will the client (supplicant) be able to connect to servers and services on the internet (and the rest of the LAN or Wide Area Network (WAN)). Figure 4.11 depicts an example of a 802.1X setting in combination with a wireless access point.

IEEE 802.1X does not require a centralized authentication server and can be deployed in stand-alone access points. Based on EAP it supports authentication schemes including, smart cards, Kerberos, public key encryption, one-time passwords and others. IEEE 802.1X is designed to be fully

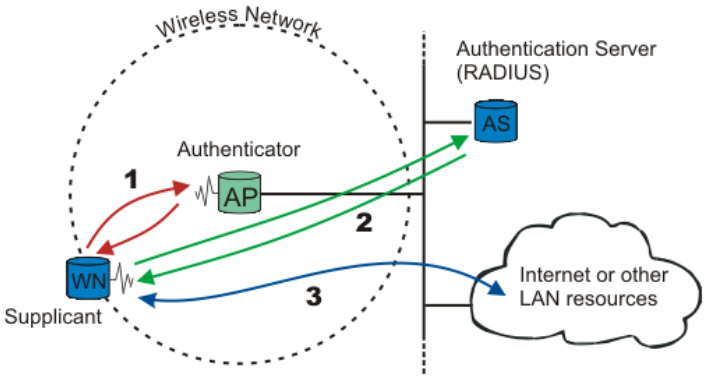

Figure 4.11: An Example of 802.1X Use with a Wireless Access Point [128].

compatible with AAA, making IEEE 802.1X usable as AAA client. This also enables the use of IEEE 802.1X with a central authentication and accounting server [26, 58].

4.3.3 IMS Security ACA

The IP Multimedia Subsystem (IMS) was developed to provide QoS to narrow- and broadband access to real time internet multimedia services such as streaming audio, video and multimedia conferences. First intended to provide QoS and inter-operator roaming in mobile systems, it now also has support for WLAN and fixed access types [103].

The focus of IMS security is on authentication, authorization, integrity, confidentiality and availability. IMS users must be authenticated and authorized in order to use IMS services. Authentication and authorization functions reside, along with other user information, on a smart card that is inserted in the user equipment. The smart card is known as Universal Integrated Circuit Card (UICC). UICC can hold different applications: Subscriber Identity Module (SIM), Universal Subscriber Identity Module (USIM) or IP Multimedia Services Identity Module (ISIM) [103].

There are five different security associations and different needs for security protection of IMS [103, 39] and they are numbered 1 to 5 in Figure 4.12 where:

1. Provides mutual authentication between the User Equipment (UE) and the IMS Core Network Subsystem (IM CN SS). The The Home Subscriber Server (HSS) is responsible for generating keys and challenges while the authentication is carried out by the Serving CSCF (S-CSCF).

2. Provides a secure link and a security association between the UE and the Proxy CSCF (P-CSCF) with authentication of data origin.

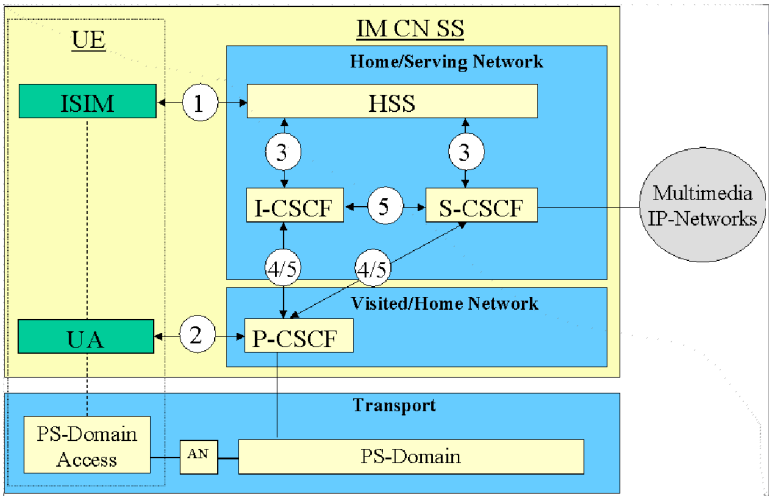

Figure 4.12: The IMS Security Architecture. Figure Reproduced from [103].

3. Provides security within the network domain internally for the Cx-interface.

4. Provides security between different networks for Session Initiation Protocol (SIP) capable nodes. This security association is applicable when the roaming UE is in a visited network.

5. Provides security within the network internally between SIP capable nodes. This security association is applicable when the UE is located in his own home network.

The following security features are included in IMS access security [103].

- Authentication of the subscriber and the network, allowing the network and the user to authenticate each other (see security association number 1 in Figure 4.12).

- Confidentiality and Integrity protection of the IMS signaling (see security association number 2 in Figure 4.12).

- Policy control system, which allows the network to control the traffic to and from the UE.

IMS uses SIP (see [112]) as the signalling protocol for creating and terminating multimedia sessions [39]. For Authentication and Key Agreement (AKA) IMS uses a scheme called IMS AKA that achieves mutual authentication between the ISIM and the Home Network (HN). IMS AKA is basically the 3G UMTS AKA protocol that is integrated into HTTP Digest protocol [103]. The technical specification on IMS security features and mechanisms is given in [39].

4.3.4 Kerberos

Kerberos is an authentication protocol, based on tickets and symmetric keys for use in a distributed environment. Kerberos, by itself, does not provide authorization. However it can make use of separate authorization procedure and put its findings in a Service Granting Ticket (SGT) [95]. It requires a centralized Authentication Server (AS) which contains user credentials and a Ticket Granting Server (TGS) that maintains security associations with all entities.

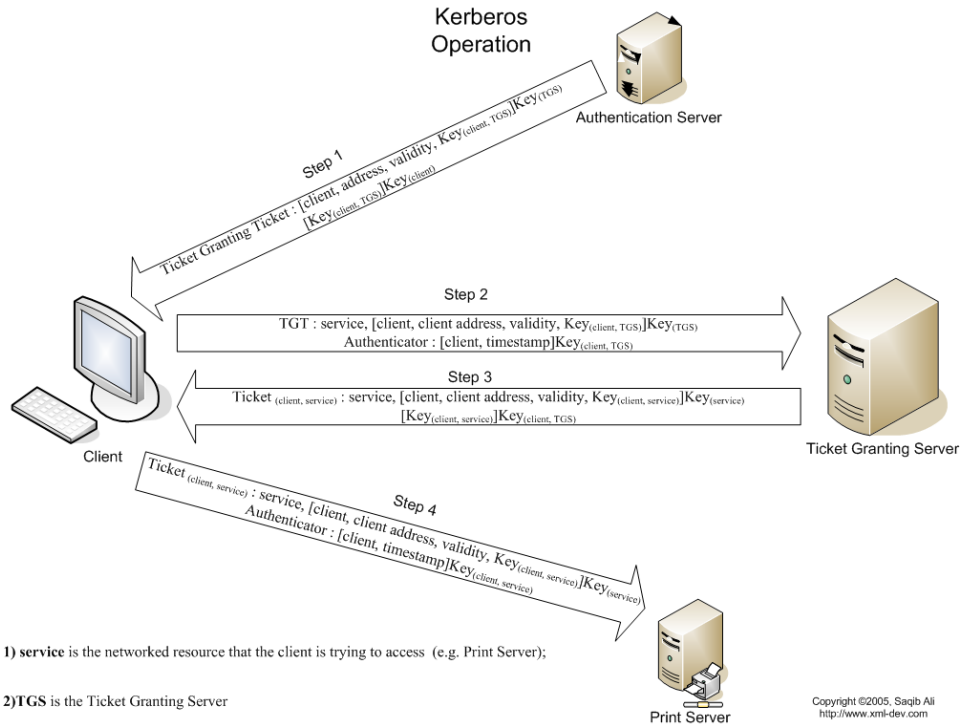

Figure 4.13: Kerberos Explained Visually. Figure Taken from `www.xml-dev.org`.

As depicted in Figure 4.13, a user can ask the AS for a Ticket Granting Ticket (TGT) and a session key (between the TGS and the client) which enables him to request access to services. The user can decrypt this information based on his own password. So only the user can decrypt the TGT and use it and the password itself is never sent over the network. When the user wishes to access a service, it requests a service granting ticket for this service at the TGS. Using the TGT the user needs not enter his credentials again for this request (single sing-on). The TGT is encrypted using the key shared between TGS an AS and contains the session key for use between client and TGS. The service granting ticket is encrypted using the key shared between the TGS and the service provider (server) and contains a session key for use

between client and server [121, Section 4.1].

The usage of Kerberos comes with a few down sides:

- The Kerberos server is a single point of failure.
- Due to timestamping clock synchronization is an issue.
- The administration protocol is not standardized.
- Since the secret keys for all users are stored on the central server, a compromise of that server will compromise all users' secret keys.
- A compromised client will compromise the user's password.

Version five of Kerberos is explained in total in [95] and is updated by [148, 72]. In [147] Elliptic Curve Cryptography (ECC) support in Kerberos is explained.

4.3.5 Security Architectures That Are Described in Virtual Organizations

Although grids were initially designed for sharing access to High Performance Computing (HPC) resources with widely distributed but a mostly static user-base, the concept of a Virtual Organization (VO) is more compatible with the dynamic nature of large user groups and distributed management. Shibboleth and PrivilEge and Role Management Infrastructure Standards (PERMIS) are often found in VOs and are discussed next.

4.3.5.1 Shibboleth

The Shibboleth system [1] is a middleware application that enables organizations to federate to extent their identity-management services. It is used to secure user access web-based resources by means of attribute-based authorization. Providing controls to protect the privacy of personal information [88]. At a request for a protected content the user is redirected to a login site from the organization he is registered with. After login the user is returned to the original service provider with a handle, a temporary identifier known to the home organization. The service provider connects to the home organization and using the handle it can retrieve (anonymous) attributes based on which the source provider grants access to the protected content.

4.3.5.2 PERMIS

PrivilEge and Role Management Infrastructure Standards (PERMIS) supports delegation of authority, decentralizing credential management. The

PERMIS authorization infrastructure provides facilities for policy management, credential management, credential validation and access control decision making. PERMIS has a credential validation service, that verifies a user's credentials prior to access control decisions and enables distributed management of credentials. Applications need to intercept users' requests and ask PERMIS to validate the user's credentials and make an access control decision. PERMIS supports hierarchical RBAC, where role hierarchies apply to organizational roles and any attribute that has natural precedence in attribute values. PERMIS also supports enhanced Attribute Based Access Control (ABAC) [25]. The PERMIS infrastructure is depicted in Figure 4.14.

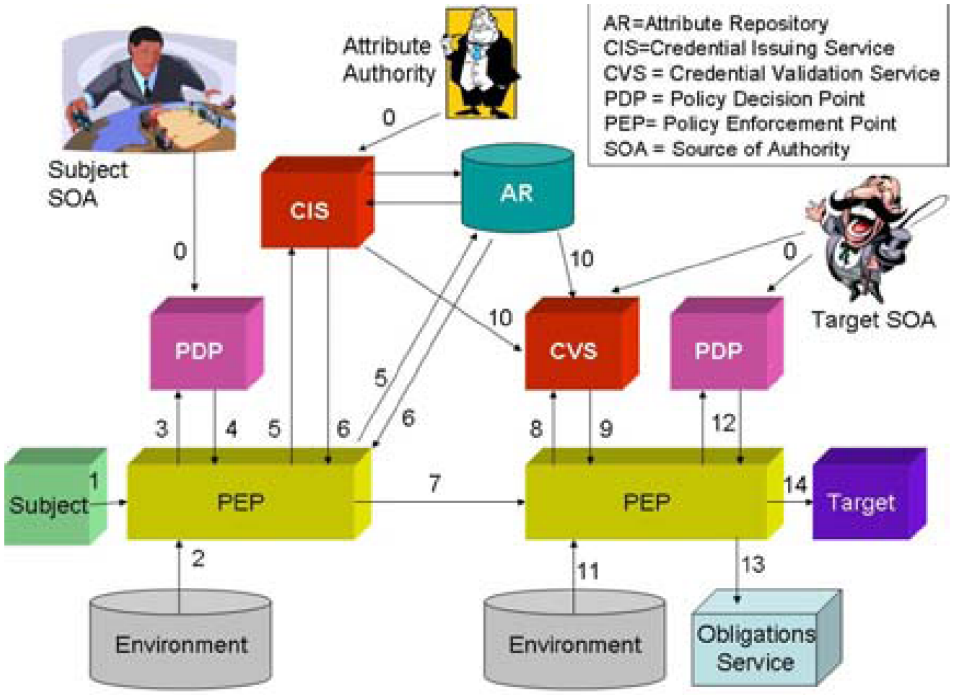

Figure 4.14: High Level Conceptual Model of the PERMIS Authorization Infrastructure. Figure Reproduced from [25].

Figure 4.14 is explained by [25] as follows.

> Step 0 is the initialization step for the infrastructure, when the policies are created and stored in the various components. Each subject may possess a set of credentials from many different Attribute Authoritys (AAs), that may be pre-issued, long lived and stored in a repository or short lived and issued on demand, according to their Credential Issuing Policies. The Subject Source of Authority (SOA) dictates which of these credentials can

leave the subject domain for each target domain. When a subject issues an application request (step 1), the application independent PDP informs the application's PEP which credentials to include with the user's request (steps 3-4). These are then collected from the Credential Issuing Service (CIS) or Attribute Repository by the PEP (steps 5-6). The user's request is transferred to the target site (step 7) where the target SOA has already initialized the Credential Validation Policy that says which credentials from which issuing AAs are trusted by the target site and the Access Control policy that says which privileges are given to which attributes. The user's credentials are first validated (step 8). This may require the CVS to pull additional credentials from an AA's repository or issuing service (step 10). The valid attributes are returned to the PEP (step 9), combined with any environmental information, such as current date and time (step 11) and then passed on to the PDP for an access control decision (step 12). If the decision is granted the user's request is allowed by the PEP (step 14), otherwise it is rejected. In either case, the PDP may also return a set of obligations, which are actions that the PEP must enforce along with the access control decision (step 13). An obligations service is the functional component that is responsible for enacting these obligations. In more sophisticated systems there may be a chain of PDPs that are called by a master PDP, with each PDP in the chain holding a different policy possibly written by a different SOA and possibly written in a different policy language. In this case the master PDP needs to hold a policy combining policy written by the target SOA, which determines the ultimate response to give to the PEP based on the set of granted, denied or don't know responses returned by the chain of PDPs. Application PEPs however should be shielded from needing to know about this more sophisticated functionality.

PERMIS is a modular concept as shown in Figure 4.15, its modules allow for a solid base that can be applied in many situations.

4.3.6 Security ACAs That Are Described in Past or Ongoing FedNet Projects.

4.3.6.1 MAGNET Beyond

In the MAGNET Beyond project, the Context Aware Security Manager (CASM), that is part of the Secure Context Management Framework (SCMF), assures security and privacy of context information inside the PN and FedNet. The CASM controls the requests that are coming in and are going out of nodes. This ensures that all information and request messages are authenti-

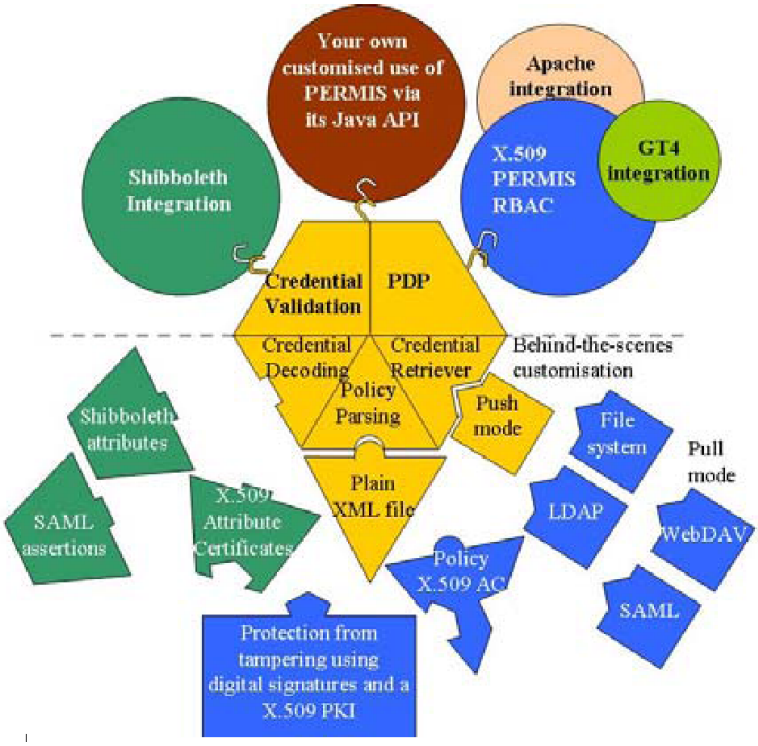

Figure 4.15: The PERMIS Authorization Decision Engine. Figure Reproduced from [25].

cated, authorized and that information going out from a node is ensured to fulfill the security and privacy requirements of the owner of the device [122].

The CASM is designed by integrating AAA module functionalities and is explained in detail in [75].

4.3.6.2 Freeband PNP2008

PNP2008 takes a two-level approach FedNet access control [56]. At the first level, access control takes place when a new member joins a FedNet. This is carried out by the FM.

Figure 3.2 on page 22 depicts the two access control levels in the lifecycle of a FedNet. At the first level three actions occur:

1. Mutual authentication on FA and FM
2. Authorization of a PN to become a member, checking whether the services offered by the PN are suitable for this FedNet.
3. Issuing a membership token for the accepted FedNet member, which the member can use to prove his membership to a PN.

The interactions between FA and FM are depicted in Figure 3.4.

At the second level, access control takes place when a FedNet member requests a FedNet service. This is carried out by the FA. In this level the following actions occur at the FA of the PN offering the service:

1. Mutual authentication between the PN that requests a service and the PN that provides this service.
2. Authorization of a service requesting PN, verifying its membership token and granting a particular access to the requested service.
3. Service proxy configuration, which is a process of configuring a requested service proxy to a particular member in a FedNet.

PNP2008 provides an Access Control and Management Framework (ACM) consisting of the FM and FA component, as discussed in Subsection 3.3.2 and 3.3.3 and a FedNet management protocol enabling PNs to communicate initiate and join a FedNet. The protocol also allows for exchanging authorization messages and manage formation, operation and dissolution of the FedNet. The framework is depicted in Figure 4.16. The inter-actions between FA and FM are depicted in Figure 3.6.

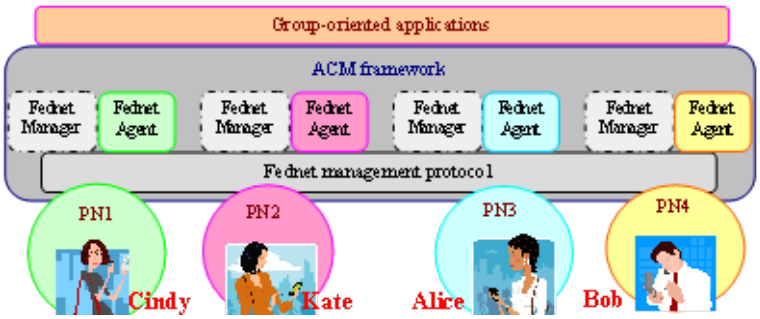

Figure 4.16: The ACM Framework. Figure Reproduced from [56].

4.3.6.3 IST SHAMAN

The main goal of the SHAMAN project is to provide a comprehensive security framework for a distributed terminal. Its security architecture should provide both a common language as well as with the basic models and real-world scenarios needed for future development. [143] provides a detailed description on PAN security.

The security aspects that they consider in their security architecture are:

1. Trust model for components and users of components
2. Creation of security associations between different components defined in the trust model

3. Distribution of the keys needed to create the security associations
4. Authentication of components and users
5. User identity and location privacy
6. Secure communication between components
7. Communication security policies and management of the policies
8. Secure installation of executables on certain components within the PAN
9. Intra-PAN authorization
10. Access control of local and distributed resources
11. Access control policies and management of the policies
12. Delegated authorization

The main objectives are:

1. A simple and practical trust model
2. Low complexity security configuration mechanisms (e.g., semi-automatic key management)
3. High level of cryptographic strength (using state-of-the-art, generally accepted cryptosystems)
4. High quality and flexible security protocols
5. Usability
6. Mechanisms and principles that can be easily analyzed and evaluated and open to public scrutiny
7. Low complexity implementations for devices with limited resources
8. Use of standard solutions whenever possible

On access security SHAMAN compares three methods: ACL, (encrypted) PAN ticket and (encrypted) PAN certificate. Based on their comparison they recommend use of ACL whereby access control records are stored on the component that acts as service provider and authentication is based on established SAs.

4.4 Selection Criteria

4.4.1 Use Case

In the introductions of the previous chapters, several use cases based on the storyline in Chapter 1 are given.

4.4.2 Assumptions

We make the following assumptions:

- The basic PN architecture is that provided in the PNP2008 project.
- The communication channels are considered out of scope in the framework.
- Due to the mobile environment, communication (overhead) should be kept to a minimum.
- Medical data is sensitive personal information.

4.4.3 Requirements

In this section the requirements for a security architecture that supports the use cases in this thesis are given. The requirements are derived based on the described use cases, the given assumptions, literature research and discussion with TI-WMC experts.

The requirements are divided in points of view of the stakeholders. Some of these requirements are applicable to multiple points of view, to avoid repetition of requirements they are denoted incrementally.

- From Harry's point of view, the following requirements are derived.
    1. The framework must support mutual authentication. As the BSK must not communicate with systems it does not trust.
    2. The framework must support end-to-end confidentiality and integrity. Credentials need to be protected.
    3. The framework must support fine grained access control configuration. Harry needs to be able to define *who* may access *what*.
    4. The framework must be context aware; that is, the framework must support context access restrictions such as time of day, location and concurrent logins.
    5. The framework should be easy to manage.
    6. The framework should be scalable in terms of devices and FedNets. Harry's PN could consist of hundreds of devices and Harry might participate in dozens of FedNets. These devices and FedNets could be added at any time.

7. The framework should be able to incorporate multiple encryption technologies. Harry should not need to install a completely new platform when, for example, a new device or service is introduced that requires a different — possibly new — encryption scheme.
8. The framework should be able to incorporate future authentication technologies. Analog to the reason stated in Requirement 7.

- From the gym's point of view, the following requirements are added.

9. The framework must be configurable in the sense that only active members with a valid subscription to the service can use it. A fee might, for example, only grant access to the service for a period of six months.
10. The framework should be configurable in the sense of protocols allowed and disallowed. The gymnasium might specify *how* users can connect in an access control configuration.

- From Harry's coach point of view, the following requirements are added.

11. The framework must support access control on the aggregated sensor readings.

- From the medical centers point of view, the following requirements are added.

12. The framework must be robust against replay attacks.
13. The framework must be scalable in the sense that it must support thousands of FedNets between as many patients and hundreds of medical staff.
14. The framework must support management based on roles. A new doctor needs to be able to check on a patient assigned to him, without the patient needing to come to the medical center for new credentials.
15. The framework must support authentication based on *public key certificates*. This is a requirement of the VITRUVIUS project.

- From a technical point of view, the following requirements are added.

16. The framework should minimize overhead because wireless communication drains the battery of mobile devices. This is also needed because it effects the efficiency of communications due to the limited bandwidth of wireless channels.
17. The framework must support infrastructure-based FedNets.

18. The framework must support access control on the network level. This is needed to prevent direct connections to sensors (nodes) before network-level-authorization.
19. The framework must support access control on the service level. This is needed to control which services are accessed on the node.
20. The framework must support *distributed* access control configurations. These configurations might reside in different storage entities located in different security domains. Each participating entity in a FedNet can have their own configuration(s).
21. The framework must support re-authentication. This is needed in case of temporary connection loss.
22. The framework must support re-authorization. This is needed when configurations are changed dynamically.
23. The framework must support data origin authentication. This is needed when devices change their network point of attachment.

4.5 Evaluation of Security ACAs in FedNets

This section evaluates the architectures stated in Section 4.3 against the requirements stated in Section 4.4. These requirements are set out against the frameworks in Table 4.1. Requirements that the framework *must* support are in bold font, requirements that it *should* support are not. '$\surd$' marks a requirement that the framework can (not necessarily *must*) support, '$\times$' marks an unsupported requirement, '$\subsetneq$' marks a partially support for the requirement and 'u' marks when support for the requirement is unknown.

Table 4.1: Comparison of ACA's.

Requirement	AAA	IEEE 802.1X	IMS	Kerberos	Shibboleth	PERMIS	Magnet Beyond	Freeband PNP2008
1: Mutual authentication	√	√	√	√	√[1]	×[2]	√	√
2: End-to-end confidentiality and integrity	√	√[3]	√	√	$\not\subset$[4]	$\not\subset$[4]	√[5]	√
3: Fine grained Access Control	√	-[6]	-[7]	√	$\subsetneq$[8]	√	√	√
4: Context aware configurable	√	√	$\subsetneq$	√[9]	√[10]	√	√	√
5: Easy management	√	p[11]	×	√	$\subsetneq$[8]	√	√	√
6: Scalable (services)	√	√	$\subsetneq$[12]	√	√	√	√	√
7: Extensible types of encryption	√[13]	√[14]	×	√	√[13]	√[2]	√[15]	√[15]
8: Extensible types of authentication	√[14]	√[14]	×	×	√	√[2]	√[15]	√[15]
9: Configurable Access Control	√	√	√	√[9]	$\subsetneq$[16]	√	√	√
10: Configurable protocols	√[13]	×	$\subsetneq$[17]	√	×	√	√[15]	√
11: Access control on aggregated data	-[18]	×	×	√[9]	√[13]	√	√	√
12: Robust against replay attack	√	√[19]	√[20]	√	√[13]	√[2]	√	√[13]
13: Scalable user pool	√	√	√	√	√	√	√	√
14: RBAC	√	×	×	√[9]	√[21]	√	×	√
15: Public Key Certificate support	√[22]	√[22]	√	√	-[23]	√	√	√
16: Minimize overheadtnotextn:x	√[13]	√	×	√[25]	×	$\subsetneq$[26]	u	u
17: Support infrastructure based	√	√	√	√	√	√	√	√
18: Network level Access Control	√	√	√	×	×	×	√	×
19: Service level Access Control	√	√	√	√	√	√	√	√
20: Distributed Access Control	√	×	-[27]	√	√	√	√	√
21: Re-authentication support	√[14]	√[14]	√	×	×	√[2]	√	×
22: Re-authorization support	√	×	×	×	×	√[28]	×	$\subsetneq$[29]
23: Data origin authentication	√	√	√	√	×	×	√	√[30]

Continued on Next Page...

Table 4.1: (continued)

[1] When combined with TLS.
[2] Actually, the PERMIS Application Programming Interface (API) is authentication agnostic. It does not itself support any authentication, only authorization. However, PERMIS can be integrated with any authentication service.
[3] Based on ESP.
[4] Based on signed X.509 certificates, integrity is provided but confidentiality is not.
[5] Using either IP security (IPsec) or TLS.
[6] Actually the service provider can limit, to extent, the resources. However in this context, Harry is both uses a service and provides a service (sensor data). 802.1X does not allow Harry to adjust *his* settings.
[7] IMS does not support two-way configuration as meant by this requirement. Several privacy options are configurable.
[8] The *user* cannot control which of his user attributes are made available, the identity provider can configure fine-grained control of information release.
[9] Kerberos does not provide authorization, but supports the passing of authorization information generated by other services.
[10] Shibboleth allows for each site to specify its own access control services.
[11] Harry can not manage at all, however the service provider can configure the system based on pre-defined options.
[12] Only IPMultimedia applications.
[13] Depending on implementation.
[14] Based on EAP.
[15] Based on IPsec.
[16] The service provider can specify which attributes are needed, but only the home organization can assign them to a user.
[17] Very limited support.
[18] The AAA framework is designed to provide authorization to internet services and recourses. Application specific attributes can be enforced.
[19] Based on IEEE 802.11i.
[20] Based on the GIBA Security Mechanism.
[21] Based on attributes.
[22] Using EAP-TLS or TTLS.
[23] The user is redirected to a web-login, the service providers are likely to use a PKI amongst themselves.
[24] [90] shows that Abstract Syntax Notation One (ASN.1) outperforms eXtensible Markup Language (XML).
[25] Kerberos provides single-sign-on, reusable SGTs and uses ASN.1.
[26] PERMIS uses X.509 attribute certificates over SAML, thus ASN.1 over XML, however policies are denoted in XML.
[27] Actually a distributed home subscriber server is supported, however, this does not conform to the requirement.
[28] Note that this is pull based, each time a user makes a request to an application to perform a task the PDP is consulted.
[29] Currently not supported, however it should be in a future version of the compliance assurance component.
[30] Based on tunnels between gateways.

4.6 Selection of a Suitable FedNet Security ACA

In the previous section, Table 4.1 clearly shows that an AAA framework is most suitable. AAA can provide, in principle, all but one requirement: access control on aggregated data. Individual implementations of the AAA framework might lack a few requirements.

The AAA framework is designed to provide authentication, authorization and accounting. PERMIS only provides authorization. PERMIS can however be integrated with any authentication application. We find PERMIS to be more suitable for *authorizing* in a FedNet environment.

AAA implementations typically provide (network) access control, whereas PERMIS provides an extensive authorization mechanism that can be used for each and every action a user performs while connected. PERMIS has a richer format for denoting such access control parameters — AAA uses attribute pairs, PERMIS supports several policy languages to denote parameters. PERMIS provides hierarchies of roles, distributed management of attributes, policy controlled decisions based on evaluated conditions and the ability to validate credentials and delegation chains. Also PERMIS supports history based decision making and multi-session separation of duties [25].

Therefore we suggest usage of the AAA framework in combination with PERMIS for use in a FedNet.

4.7 Summary

In Section 4.1 we discussed security threats. In Section 4.2 we discussed techniques that counter these threats. They are: authentication, authorization, data integrity, confidentiality — both of data and identity — and non repudiation. These techniques rely heavily on keys. In Section 4.3 we discussed several existing ACAs. In Section 4.4 we discussed selection criteria for a ACA in the context of a FedNet. These are derived on the described use case, assumptions, literature research and discussion with TI-WMC experts. In Section 4.5 we set out the criteria against the available ACAs, allowing us to select a suitable one in Section 4.6. AAA enhanced with PERMIS for authorization is suggested as the best candidate for a secure ACA for FedNets

Chapter 5

The Authentication Protocol

Before authorization can take place, the identity of the user needs to be verified — i.e., authenticated. Because authorization is based on the identity of a user, authentication needs to be performed in a secure way. In our previous work [15] we showed that Diameter is a suitable authorization framework for use in a FedNet. Diameter implementations can incorporate authentication methods based on AVPs. Diameter also supports EAP, this enables diameter to incorporate new EAP-based authentication methods without need for a new Diameter application [21].

EAP [3] typically runs over data links layers such as PPP or IEEE 802 networks, without requiring IP. EAP can be used on dedicated links, switched circuits, wired links and wireless links. EAP finds its use in hosts and routers that connect via switched circuits or dial-up lines using PPP [118]. It also finds use in switches and ACs using IEEE 802. EAP encapsulation on IEEE wired media is described in [58] and encapsulation on IEEE wireless LANs in [57].

As Diameter is not the only application using EAP, many authentication methods have been made compatible with EAP. Examples are EAP-TLS [117], EAP-IKEv2 [132] and EAP-AKA [10].

5.1 Available Authentication Methods

We limit our evaluation of authentication methods to those that are EAP compatible for the reasons stated above. The IANA keeps a registry of assigned identifiers for method types. At the time of writing, IANA has fifty-one [53] registered EAP type-values. A comparison of all fifty-one goes beyond the scope of this Thesis. Therefore, based on requirements from preliminary research [15], only those that support public keys and certificates will be compared in this Thesis. Undocumented, possibly proprietary, protocols are also omitted.

5.1.1 RSA Public Key Authentication

Ron Rivest, Adi Shamir and Leonard Adleman (RSA) public key authentication is documented in an (expired) IETF internet draft [141] and is a patented method. It authenticates based on public key certificates, exchanges a session key and is integrity protected by public keys.

The latest IETF internet draft, version four [141], is publicly available on the internet. According to [114], there is a patented method of the PPP EAP RSA Public Key Authentication Protocol. We have found mention of a version seven in of the internet draft mentioned in [141]. We were unable to obtain this version seven of the internet draft.

RSA has IANA EAP type 9 [53]. Due to the patented method and poor documentation we decided not to examine RSA any further.

5.1.2 EAP-TLS

Extensible Authentication Protocol (EAP) Transport Layer Security (TLS) provides certificate-based mutual authentication and key derivation. It uses protected cipher suite negotiation, mutual authentication and key management capabilities of the TLS protocol version 1.1 ([31]) [117]. EAP-TLS was first defined in [4] which was obsoleted by [117]. The latter specifies a fully backward compatible, extended and enhanced version of the protocol. Both [4, 117] dictate the use of TLS version 1.0 [30] or higher — currently version 1.1 [31] and version 1.2 [32] are also available. When using TLSv1.2, EAP-TLS provides key derivation function negotiation [117].

EAP-TLS has IANA EAP type 13 [53].

5.1.3 EAP-TTLS

(EAP-TTLS) allows legacy — e.g., password-based — authentication protocols to be used against existing authentication databases, while protecting the security of these legacy protocols against eavesdropping, negotiating-down, man-in-the-middle and other attacks [45].

There exists a version 0 a and version 1 of the protocol, both have IANA EAP type 21 [53].

- **EAP-TTLSv0**
  When EAP-TTLS is mentioned, EAP-TTLSv0 is meant, which is documented in [45]. EAP-TTLSv0 consists of two phases, in the first — the handshake phase — the server is authenticated using standard TLS procedures. During this phase keying material is generated and a cryptographically secure tunnel for information exchange in the subsequent phase — the data phase– is set up. Instead of server authentication, mutual authentication might be performed in the first phase. However, client authentication is usually performed in the second (data) phase.

During the data phase the client is authenticated to the server using an arbitrary authentication mechanism encapsulated within the secure tunnel. Again, mutual authentication might be performed during this phase instead of client authentication only. The data phase may also be used for additional, arbitrary data exchange [45].

TLS1.0-1.2 [30, 31, 32] state: "The negotiation of a shared secret is secure: the negotiated secret is unavailable to eavesdroppers and for any authenticated connection the secret cannot be obtained, even by an attacker who can place himself in the middle of the connection."

- **EAP-TTLSv1**
  EAP-TTLSv1 uses as underlying protocol TLS with the Inner Application extension (TLS/IA). In TLS/IA, the TLS handshake is followed by an exchange of messages with record type "InnerApplication", in which an arbitrary exchange of messages between client and server is conducted under the confidentiality and integrity protection afforded by the TLS handshake. In contrast to version 0, version 1 provides cryptographic binding and (optionally) channel binding. Version 1 also provides stronger key strength.

  EAP-TTLSv1 is documented in the expired internet draft [46].

5.1.4 PEAP

Protected Extensible Authentication Protocol (PEAP), pronounced 'peep', "is a joint proposal by Cisco Systems, Microsoft and RSA Security as an open standard. It is already widely available in products and provides very good security. It is similar in design to EAP-TTLS, requiring only a server-side PKI certificate to create a secure TLS tunnel to protect user authentication." PEAP has IANA EAP type 25 [53].

- **PEAPv0/EAP-MSCHAPv2**
  "PEAPv0/EAP-MSCHAPv2 is the technical term for what people most commonly refer to as "PEAP". Whenever the word PEAP is used, it almost always refers to this form of PEAP since most people have no idea there are so many different versions of PEAP. Behind EAP-TLS, PEAPv0/EAP-MSCHAPv2 is the second most widely supported EAP standard in the world."

  PEAPv0 [89] is basically an outer TLS session based on a server certificate, with a inner EAP protocol — usually MS-CHAPv2 — to authenticate the client. The aim is to prevent the rollout of an infrastructure requiring clients to have certificates. Note that it MUST implement TLS version 1.0 [4], although version 1.1 [31] and 1.2 [32] have been defined [89].

- **PEAPv1/EAP-GTC**
  "PEAPv1/EAP-GTC was created by Cisco as an alternative to PEAPv0/ EAP-MSCHAPv2. It allows the use of an inner authentication protocol other than Microsoft's MSCHAPv2. EAP-GTC is defined in RFC 3748. It carries a text challenge from the authentication server and a reply which is assumed to be generated by a security token. EAP-GTC does not protect the authentication data in any way." [wikipedia]

  Because PEAPv1 is based on a token card, it falls out of the scope of this thesis and therefore will not be examined any further.

- **PEAPv2**
  Protected Extensible Authentication Protocol (PEAP)v2 is documented in an expired (April 2005) internet draft [100]. It is unclear why PEAPv2 did not make it to full RFC, nor why there are hardly any implementations of the draft. Lack of in the field deployment makes this method unsuitable, because it has not been rigorously tested.

  PEAPv2 provides identity protection, dictionary attack resistance, protected negotiation, header protection, protected termination, fragmentation and reassembly, fast reconnect, standard key establishment, sequencing of multiple EAP methods, protected exchange of arbitrary parameters, credential provisioning, optimization for light weight devices and server unauthenticated tunnel provisioning mode [100].

5.1.5 MAKE

Mutual Authentication with Key Exchange (MAKE) provides as the name suggest mutual authentication. It also supports certificates. Based on the little information we could find [13], several requirements are not met. Because of its scarce documentation we will omit this authentication method from further investigation.

The MAKE protocol has IANA EAP type 27 [53].

5.1.6 EAP-FAST

EAP-Flexible Authentication via Secure Tunneling (EAP-FAST) [22] is another tunneling protocol based on TLS. It aims to provide security to network access solutions that require user friendly and easily deployable secure authentication mechanisms. These are often based on user credentials. User credentials typically are weaker than (strong) mutual authentication protocols, such as certificate-based authentication protocols. EAP-FAST secures these weaker mechanisms by first setting up a more secure tunnel based on TLS.

The EAP-FAST protocol has IANA EAP type 43 [53].

5.1.7 EAP-IKEv2

EAP-Internet Key Exchange Protocol version 2 (EAP-IKEv2) [132] is based upon Internet Key Exchange (IKE)v2 [73]. EAP-IKEv2 provides mutual authentication and session key establishment supporting authentication techniques based on passwords, high-entropy shared keys (symmetric keys) and public key certificates (asymmetric keys). EAP-IKEv2 further provides support for cryptographic cipher suite negotiation, hash function agility, identity confidentiality (in certain modes of operation), fragmentation and an optional "fast reconnect" mode. EAP-IKEv2 provides similar functionality as, for example, EAP-TLS. However, EAP-IKEv2 does not tunnel other EAP methods [132].

The EAP-IKEv2 method has IANA EAP type 49 [53].

5.1.8 EAP-PSK

EAP-Pre-Shared Key Extensible Authentication Protocol (EAP-PSK)[14] is based on an key shared in advance that is only known to the authenticator and user. A concept also know as PSK. We take this method into account, even though it lacks support for certificates, as an informative reference. EAP-PSK is light-weight and can be used to provide mutual authentication, integrity protection, replay protection, key derivation, dictionary attack resistance and session Independence with the ease of not having to set up a key-distribution-mechanism. It has been adopted by WiMAX[78].

As a design choice for the sake of simplicity EAP-PSK only supports a single cryptographic primitive, Advanced Encryption Standard (AES)-128. This prevents the use of (protected) cipher negotiation and the support of features such identity protection (confidentiality). In particular, EAP-PSK does not provide fast reconnect, fragmentation, cryptographic binding, perfect forward secrecy and channel binding.

The EAP-PSK protocol has IANA EAP type 47 [53]. As EAP-PSK does not support certificates, we omit this method from further examination.

5.2 Authentication Protocol Requirements

The risks involved in authenticating over a wireless medium are considerably higher than over a wired medium [46]. Because of this, we adopt the requirements from [123] for use in a FedNet. [123] specifies EAP method requirements for WLANs, from which the following MUST requirements are taken:

1. Generation of keying material. “Key derivation” as defined in [3, Section 7.2.1].

2. 128-bits of effective key strength, as defined in [3, Section 7.2.1.][1].

3. Key derivation exports a Master Session Key (MSK) of at least 64 octets.

4. Key derivation exports a Extended Master Session Key (EMSK) of at least 64 octets.

5. Mutual authentication support. "Mutual authentication" as defined in [3, Section 7.2.1].

6. Shared state equivalence. "The shared EAP method state of the EAP peer and server must be equivalent when the EAP method is successfully completed on both sides. This includes the internal state of the authentication protocol but not the state external to the EAP method, such as the negotiation occurring prior to initiation of the EAP method. The exact state attributes that are shared may vary from method to method, but typically include the method version number, the credentials presented and accepted by both parties, the cryptographic keys shared and the EAP method specific attributes negotiated, such as cipher suites and limitations of usage on all protocol state. Both parties must be able to distinguish this instance of the protocol from all other instances of the protocol and they must share the same view regarding which state attributes are public and which are private to the two parties alone. The server must obtain the authenticated peer name and the peer must obtain the authenticated server name (if the authenticated server name is available). "[123]

7. Resistance to dictionary attacks. "Dictionary attack resistance" as defined in [3, Section 7.2.1].

8. Protection against man-in-the-middle attacks. This corresponds to "Cryptographic binding", "Integrity protection", "Replay protection" and "Session independence" security claims as defined in [3, Section 7.2.1].

9. Protected cipher suite negotiation. *When* the method negotiates a cipher suite to protect the EAP conversation, then it MUST support the "Protected cipher suite negotiation"[2] as defined in [3, Section 7.2.1].

Additionally [123] specifies the following SHOULD requirements.

[1] We base key strength on the comparison by the National Institute of Standards and Technology (NIST) of the USA documented in [94].

[2] This refers to the ability of an EAP method to negotiate the cipher suite used to protect the *EAP conversation*, as well as to integrity protect the negotiation. It does not refer to the ability to negotiate the cipher suite used to protect *data*.

10. Fragmentation. “Fragmentation” as defined in [3, Section 7.2.1].
11. End-user identity hiding. “Confidentiality” as defined in [3, Section 7.2.1].

Finally [123] specifies the following MAY requirements.

12. Channel binding. “Channel binding” as defined in [3, Section 7.2.1].
13. Fast reconnect. “Fast reconnect” as defined in [3, Section 7.2.1].

The client/peer must establish a connection to the authenticator in this case, a wireless connection. An important requirement is the secure channel between the authenticator and the EAP server. This is vital because the specification does not indicate how this is established, but it requires one. Based on requirements listed in Section 4.4.3, the following must requirements are derived in the context of an authentication protocol.

14. Seamless handover (data origin authentication). The session needs to be resumable when the user is roaming to another Access Point (AP)
15. Public Key Certificate. The method supports certificates based on asymmetric keys.
16. Re-authentication. Server initiated re-authenticate request.
17. Cipher suite negotiation. Enforce use of cipher suite negotiation, to optimally protect the transmitted data.
18. The authentication protocol is interoperable with the authorization framework.
19. The protocol has been tested and proven in the field.

5.3 Comparison of Authentication Methods

In this section, the listed methods in Section 5.1 are set out against the requirements listed in Section 5.2 in Table 5.1. The must requirements are denoted in a bold font, the should in an italic font and the may requirements in a normal font. Denotation: $\surd$ means requirement supported although it might need to be configured, $\times$ means requirement not met, $\subsetneq$ means has partial support for this requirement, $\neg$ means does not apply and ? means that we were not able to verify this requirement.

The client/peer must establish a connection to the authenticator — in this case, a wireless connection. An important requirement is the secure channel between the authenticator and the EAP server. This is vital because the specification does not indicate how this is established, but it requires one.

Table 5.1: Evaluation of authentication protocols.

Requirement	EAP-TLS [117]	EAP-TTLSv0 [45]	EAP-TTLSv1 [46]	PEAPv0 [89]	PEAPv2 [100]	EAP-FAST [22]	EAP-IKEv2 [132]
1 Key generation	√	√	√	√	√	√	√
2 ≥ 128 bits effective key strength	√	√	√	×[3]	√	√	√
3 ≥ 64 octet MSK key derivation	×[4]	×[4]	√	×[4]	√	√	√
4 ≥ 64 octet EMSK key derivation	×[4]	×[4]	×	×[4]	√	√	√
5 Mutual authentication	√	√	√	√	√	√	√
6 Shared state equivalence	**?**	**?**	**?**	**?**	√	**?**	**?**
7 Dictionary-attack resistance	√	√	√	√	√	√	√
8 Man-in-the-middle protection	√	$\subsetneq$[5]	√	√	√	√	√[6]
Cryptographic binding	¬	×	√	√	√	√	¬
Integrity protection	√	√	√	√	√	√	√
Replay protection	√	√	√	√	√	√	√
Session independence	√	√	√	**?**	√	√	√
9 Protected cipher suite negotiation[7]	√	√	√	√	√	√	√
10 Fragmentation	√	√	*?*	*?*	√	√	*?*
11 End-user identity hiding	√	√	√	√	√	√	√
12 Channel binding	¬	×	√	?	?[8]	×	¬
13 Fast reconnect	√	√	√	√	√	√	√
14 Seamless handover	√[9]	√[9]	√[9]	√[9]	√[9]	√[9]	√
15 Public Key Certificate	√	√	√	√	√	√	√
16 Re-authentication	**?**	$\subsetneq$[10]	**?**	**?**	$\subsetneq$[11]	**?**	$\subsetneq$[12]
17 Cipher suite negotiation (must)	√	√	√	√	√	√	√
18 Authentication-authorization interoperability[13]	√	√	√	√	√	√	√
19 Method is proven in the field[14]	√	√	√	√	×	√	√

[3]The PEAP tunnel uses a 60 octet key.

[4]TLS negotiates a 48 byte master key.

[5]Strong mutual authentication implementations (certificate or unique pre-shared based) are not vulnerable to this attack.

[6]The man-in-the-middle attack is possible when legacy authentication with EAP is used.

[7]This refers to the ability of an EAP method to negotiate the cipher suite used to protect the *EAP conversation*, as well as to integrity protect the negotiation. It does not refer to the ability to negotiate the cipher suite used to protect *data*.

[8]has connection-binding though

[9]TLS version 1.0, 1.1 and 1.2 all specify the 'session identifier' as "An arbitrary byte sequence chosen by the server to identify an active or resumable session state." [30, 31, 32].

[10]Based on predefined session time outs.

[11]Re-authentication based on TLS session resumption.

[12]Based on authentication lifetime as defined in [98]

[13]Perhaps trivial as it is implied by the use of EAP

[14]Mocana's® NanoEAP™ [92] actually incorporates all methods

Noteworthy, we discovered that all the methods do not support server initiated re-authentication. There are some workarounds, such as a session lifetime by which the server sets a limit, however this is not the same as the intended re-authentication after a policy change. A server could always disconnect/terminate the session, leaving it to the client to re-connect.

5.4 Authentication Protocol Recommendation

In Section 5.1 we looked at a number of EAP compatible authentication methods and were able to eliminate several candidates based due to patents, scarcely available documentation and lack of certification support.

Based on Table 5.1 EAP-TLS and EAP-IKEv2 are the most eligible protocols. We recommend the EAP-IKEv2 protocol based key derivation support that EAP-TLS lacks. The EAP-IKEv2 protocol can be combined with other EAP implementations, supports most of the requirements and does not need other authentication protocols to run over it for authentication, thus reducing the amount of messages needed.

5.5 Summary

In Section 5.1 we discussed several authentication protocols. In Section 5.2 we stated the requirements for such a protocol in the context of FedNets. In Section 5.3 we provided an overview of the authentication protocols that might be applicable and their conformance to the requirements.

Finally we recommended the EAP-IKEv2 protocol in Section 5.4.

Chapter 6

The Ciphersuite

A cryptographic cipher is an algorithm for encryption and/or decryption of data. Depending on the cipher it provides a certain type of security. In Chapter 4 we introduced several security threats and techniques to counter them. A ciphersuite is a specific collection of such algorithms that together provide a certain set of security measures. In Chapter 5 we proposed a suitable authentication method, one of the needed algorithms in a ciphersuite.

In this chapter we will explain how keys withstand known attacks with their computational strength, what a ciphers is and how it selects other algorithms to provide the necessary security.

6.1 Keys

Ciphers are usually based on (secret) keys. These keys need to be distributed to the using end systems. One way to do this, is to generate a symmetric key and transport this key trough a separate channel to the using entities. This could be written on paper, put on a pendrive or some other secured method. This approach is suitable for small systems.

This approach is also applicable in large systems if there is a central authority that possesses all the keys. As each entity has a shared key with the authority — called a Security Association (SA) — and all the entities trust the authority, the central authority can generate key pairs for use between entities that do not have a SA with each other. This approach is taken by the Kerberos system [95].

For very large systems, this method is not feasible as for each tuple of entities there needs to be a shared key, generating an exponential need for keys. The central authority, that has to generate a key each time a secure communication needs to be set up, is a single point of failure. Asymmetric encryption provides us with a solution. When a message needs to be send encrypted, the message can be encrypted using the public key of the recipient

that is publicly available. Only the intended recipient has the corresponding private key to decrypt the message. If the recipient wishes to reply, he cannot encrypt the reply with his own private key as the public key — that everybody can obtain — decrypts the message. Instead he needs to obtain the public key of the initiator of the conversation and encrypt the message using that key to communicate securely. As entities need to be able to obtain other entities public keys, this approach needs some kind of mechanism to make these keys publicly available. Such a mechanism is usually referred to as a Public Key Infrastructure (PKI). In the next chapter we will go into more detail about key-distribution-mechanisms.

6.1.1 Key Derivation

Asymmetric encryption requires more computational resources than symmetric encryption. When large amounts of data need to be sent, it is often useful to set up a secure connection using asymmetric keys and derive a key and agreed cipher for symmetric encryption in the process.

When two entities willing to enter secured communication both have a asymmetric key, a symmetric key can be derived. RSA and Diffie Hellman (DH) are two well known schemes for this. We shall not investigate the exact workings of these schemes in this thesis.

DH is susceptible to man-in-the-middle attack [64]. Though combining DH with digital certificates prevents such an attack.

A variant of the traditional DH scheme is the Elliptic Curve Diffie-Hellman (ECDH) scheme. This is in effect the same scheme, however it uses ECC instead of Integer Factorization Cryptography (IFC).

6.1.2 Key Strength

Due to the nature of ciphers and known attacks, the key length itself is not a comparable unit for security. Therefore we set out the key sizes in Elliptic Curve Cryptography (ECC), Integer Factorization Cryptography (IFC) and Finite Field Cryptography (FFC) ciphers against their corresponding key strength in Table 6.1. We also included Tripple-DES (3DES), SHA-1 and SHA-2 for comparison as they have been the recommendation for years and are still widely in use.

Based on the comparable keystrength given in Table 6.1, implementations must not use 3DES or SHA-1, must not use ECC lower than prime-256 and must not use IFC or FFC with a key size under 3072.

Table 6.1: Key size comparison.

Bits of security ($O(2^{bits})$)	52	80	112	128	192	256
Equivalent symmetric key size [1]	52	80	112	128	192	256-
3DES		168[2]	168[3]	-	-	-
AES	-	-	-	128	192	256
ECC (e.g., ECDH)[1] f =	-	160-223	224-255	256 383	384-511	512+
IFC (e.g., RSA)[1] k =	-	1024	2048	3072	7680	15360
FFC (e.g., DSA)[1] N =	-	160	224	256	384	512
L =	-	1024	2048	3072	7680	15360
SHA-1	160[4]	-	-	-	-	-
SHA-2	-	-	-	256[5]	384[5]	512[5]

[145] shows that the use of a symmetric encryption cipher takes less power consumption than asymmetric encryption. [137] state that symmetric encryption based schemes do not scale well for large (sensor) networks and that their asymmetric scheme is a factor 80 more expensive compared to the symmetric scheme. Therefore we recommend the use of asymmetric encryption for authentication — allowing for non repudiation and symmetric encryption of data exchange.

ECC requires less computation power than IFC and FFC [74]. Using ECC thus saves battery life, communication overhead and time as evidenced in research such as [84, 80, 16, 51, 79, 130, 136, 145]. Therefore we recommend the use of ECC.

6.2 Cipher Suites

Cipher suites define a combination of acceptable key derivation-, identity provision-, authentication- and encryption algorithm as well as a MAC. For example: "TLS_DH_DSS_WITH_AES_128_CBC_SHA" identifies the TLS authentication method, using the DH key exchange and a Digital Signature Standard (DSS) signature to set up the connection and the symmetric cipher AES to encrypt data protected using the SHA Message Authentication Code (MAC).

6.2.1 Cipher Suite Assumptions

With regard to the cipher suite we assume the following:

[1] Based on [94]
[2] Based on Two key Triple DES (2TDEA)[94] — i.e., $key_1 = key_3 \neq key_2$
[3] Based on Three key Triple-DES (3TDEA)[94] — i.e., $key_1 \neq key_2 \neq key_3 \neq key_1$
[4] Based on [85]
[5] Based on [59]

- The PN takes care of intra-cluster and inter-cluster — i.e. intra-PN — network-layer security.
- We focus on the federation of networks either via a third party infrastructure (i.e., the internet) or directly (i.e., ad hoc) through wireless technology.
- The data being communicated between the FedNet members is top secure, such as it is sensitive medical data.
- The FedNet has access to the Certificate Authority (CA).

6.2.2 Cipher Suite Requirements

In order to determine a suitable cipher suite we need to determine which ciphers to use with the proposed authentication protocol (see Section 5.4). Some cipher suites specify one or more of encompassed methods to be "null" — i.e., not used. This provides some security threats. To ensure authentication a signature is needed. To ensure confidentiality a encryption method needs to be specified. To ensure integrity a Message Authentication Code (MAC) needs to be specified. All of these need to have sufficient cryptographic strength. In Section 6.1.2 we will compare the cryptographic strength of various ciphers.

Based on the above, Section4.4.3 and [3] we derive the following requirements in regard to cipher suites.

A suitable cipher has the following requirements.

- Ciphers used in the suite are not broken.
- The cipher has been proven in the field.
- Secure key derivation.
- The suite provides authentication.
- The suite provides data confidentiality.
- The suite provides data integrity.
- Cipher has at least the equivalent strength of a 128-bit-symmetric-key encryption as defined in [3, Section 7.2.1.].

6.2.3 Broken Ciphers

Over the centuries many encryption ciphers and other cryptographic techniques have been invented and broken. It goes beyond the scope of this thesis to cover all of them and therefore we only look at the most popular ciphers that were recently used in computer science. The following methods

are considered broken and insecure [105]: RC4, RC5, Message Digest Algorithm 4 (MD4), MD5, One-Way Hashing Algorithm with Variable Length of Output (HAVAL) and Race Integrity Primitives Evaluation Message Digest (RIPEMD). A feasible (hardware enabled) brute force attempt in 1998 was successful performed against DES, making it insecure [121, Section 2.2]. According to [121, Section 3.2], SHA-0 and SHA-1 should not be used in newly developed applications and be phased out by 2010. Studies [138, 85] have shown SHA-1 to be more likely to have collisions as was intended during design. Therefore implementations must not use RC4, RC5, MD4, MD5, HAVAL, RIPEMD, DES, SHA-0 and SHA-1.

6.2.4 Security and Encryption Recommendations

Based on requirements stated in Section 5.2, equivalent key sizes given in Section 6.1.2, and the authentication protocol selected in the previous chapter we recommend cipher suites that:

- base key-exchange on ECDH with a (minimum of) 256-bit prime moduli key using the EAP-IKEv2 protocol; *and*
- base digital signatures on Elliptic Curve Digital Signature Algorithm (ECDSA) with a (minimum of) 256-bit prime moduli key; *and*
- base encryption of data on AES with a minimal key size of 128 bits; *and*
- base MAC on SHA-2 with a minimum of 256 bits.

These protocols are in line with the NSA suite B recommendations [99], with the exception of the authentication protocol as authentication is not part of the recommendations. All these method are available as open standard, or relevant patents are either expired or royalty free.

6.3 Summary

A cipher suite is a combination of key derivation, identity provision, encryption algorithm and MAC for use with a given authentication protocol. We recommend use of a security suite that uses ECDH, ECDSA, AES and SHA-2. All using key strength with 128 or more bits of security ($O(2^{bits})$). These recommendations are in line with NSA suite B recommendations [99].

Chapter 7

The Credential Provider

In this chapter we select a suitable credential provider for use in a FedNet. A credential provider is a mechanism for providing certificates and status information. We will first introduce several credential providers in Section 7.1. Next we will look at requirements in Section 7.2. Compare credential providers and select a suitable one in Section 7.3 and finally summarize in Section 7.4.

7.1 Credential Providers

As we will use Public Key Certificates (PKCs) we need to have certificates and be able to validate them. A certificate is a digitally signed statement binding the key holder's (principals's) name to a public key and various other attributes. The signer — i.e., the issuer — is commonly called a Certificate Authority (CA) [48]. [27] documents the Internet X.509 Public Key Infrastructure Certificate and the Certificate Revocation List (CRL) Profile. As X.509v3 certificates are de facto standard [131, 47] we base our system on X.509.

Many papers have been written on the subject of certificate revocation. [142, 35, 146, 48] discuss and compare different revocation methods, [106] discusses certificate revocation in vehicular networks.

Originally certificates were validated using a Certificate Revocation List (CRL)[27]. This is not ideal in the FedNet context as it does not allow for realtime state information [48] and can create a lot of overhead [35, 24]. Therefore we propose to retrieve the status of a certificate upon presentation, using an online retrieval system. [48] points out that an online system provides a heavy burden on the revocation server as it needs to digitally sign each status request. Thus online over CRL mechanisms increases CPU cost and decrease bandwidth cost. It is a well accepted fact that in wireless (sensor) networks computational cost is preferred over communication cost.

Not only do the individual certificates need to be validated, the whole trust-chain needs to be verified. If the chain contains certificates that are not known to the authenticator and are not provided by the authenticatee these certificates need also be retrieved.

We will look at the following options and explain them in limited detail:

1. File system
2. CRL
3. Certificate Revocation Status
4. OCSP
5. LDAP
6. WebDAV
7. SAML
8. AAA server
9. Tickets
10. Domain Name System (DNS)
11. PERMIS

1. **File system**
   One simple solution is to store credentials in the (local) file system. This is the case for many password based UNIX systems that use stores user information in the 'passwd' file and their corresponding hashed passwords in the 'shadow' file. This is feasible if all entities connect to the same system. When the entities need to authenticate each other this is not possible. Kerberos [95] provides a solution in the latter case as the central Kerberos server maintains a security association with all entities, providing new keys when entities need to communicate between each other.

   As we use certificates in a dynamic environment, storing all the necessary certificates on all the local filesystems would create the same problems for symmetric keys stated in [137] — i.e., the scalability issue, the storage issue, re-deployment issue and key distribution issue. Using Kerberos would create a single point of failure effectively disabling new communications when the Kerberos server is unavailable. For these reasons we omit the file system as a feasible option.

2. **CRL**
   The main advantage of a CRL is that the list can be stored in an untrusted directory. Due to the signature of the CA that revoked the individual certificates over the whole list, the directory cannot withhold an individual certificate revocation [86].

   However, CRL comes with a few downsides. Such as delay between publications of revocations and the necessity for retrieval of the whole list [24, 109, 48]. Despite the introduction of Δ-CRL to limit the amount of data transferred by providing an incremental set of revocations since the latest full CRL, a delay remains between the revocation updates. [146] also points out the risk for CRL request implosion — i.e., at or near CRL publication time a burst of requests for the updated CRL may occur as clients wish to limit the window of vulnerability.

3. **Certificate Revocation Status**
   [49, 146] Micali introduced Certificate Revocation Status (CRS) in 1995, improved 1996. further improved in 2002 as NOVOMODO. Requires multiple hashing which is computationally intensive (however less costly than signing) and increases transmission cost.

4. **OCSP**
   Online Certificate Status Protocol (OCSP)[91] solves the distribution issue of CRL, allowing for on demand verification of certificates. OCSP does not resolve the delay issue of CRL and it does not scale well as it computes a digitally signature for each query [49, 146].

5. **LDAP**
   The LDAP [116] provides access to distributed directory services that act in accordance with X.500 data and service models. There are many LDAP based implementations of directory services [77]. LDAP can store X.509 certificates and CRLs. LDAP uses ASN.1[63] enabling storage of certificates in Basic Encoding Rules (BER) [135].

6. **WebDAV**
   WebDAV[33] is a set of extensions to the HTTP/1.1 protocol [43] which allows users to collaboratively edit and manage files on remote web servers. WebDAV can be used to connect to the server via HyperText Transfer Protocol Secure (HTTPS) and then check for availability of the certificate. When this fails, it should abort authentication.

   [24] proposes a method that uses WebDAV to improve certificate revocation and publication addressing both, the delay and overhead issue of CRL. Using this method, (the expensive) signature verification is not needed. This requires the retrieval of the certificate and a bitwise comparison. This might be beneficial in certain situations where com-

putation power is limited and the retrieval of the certificate is either faster or less power consuming.

7. **SAML**
   SAML[38] specifies syntax and semantics for XML-encoded assertions about authentication, attributes and authorization and for the protocols that convey this information. SAML does not specify how the authentication, attributes and authorization information is to be stored, this therefore depends on the implementation.

8. **AAA**
   AAA conforming protocols, such as radius and diameter have a back-end server that stores credentials. These credentials are often in the form of an identifier — e.g., username — and a password which are stored in a database. When auditing is in place these same protocols are sometimes called Authentication, Authorization, Accounting and Auditing (AAAA)

9. **Tickets**
   Tickets can provide credentials. However, the ticket needs to be issued by some trusted entity, e.g. Kerberos system. As stated in Section 4.3 this creates a single point of failure and is therefore not suitable to our needs. This is one of the reasons why certificates were mandated as a requirement.

10. **DNS**
    DNS based revocation lists are prone to a spoofing attack and are therefore not suitable for our needs.

11. **PERMIS**
    PrivilEge and Role Management Infrastructure Standards (PERMIS) can be considered a credential provider as the framework allows for creating and storing credentials. However, the main focus of PERMIS is to provide a reasoning engine. As such it is capable with several credential providers — e.g., local file system and LDAP. The PERMIS framework does facilitate in the creation and storage of credentials. As PERMIS relies on — rather than supplies — a credential providers, we will not consider PERMIS an individual credential provider.

7.2 Requirements

To select a suitable credential provider we derive the following requirements.
The credential provider:

1. can store X.509 certificates

2. can store certificates revocations
3. provides realtime updates of revocations
4. is able to provide selective information (as not to download the entire certificate or CRL)
5. keeps overhead (transmission cost) to a minimum — i.e. ASN.1 is preferred over XML
6. is computational efficient.

7.3 Comparison and Selection

In Table 7.1 we set out the requirements against the credential providers mentioned in Section 7.1 that have not already been eliminated. Denotation: $\surd$ means requirement supported although it might need to be configured, $\times$ means requirement not met and $\subsetneqq$ means has partial support for this requirement.

Table 7.1: Evaluation of authentication protocols.

Requirement	LDAP (CRL)	SAML	WebDAV	OCSP	Filesystem
1 Certificate storage	$\surd$	$\times$	$\surd$	$\times$	$\surd$
2 Certificate revocation	$\surd$	$\subsetneqq$[1]	$\surd$	$\surd$	$\subsetneqq$[2]
3 Realtime updating	$\subsetneqq$[3]	$\surd$	$\surd$	$\times$	$\subsetneqq$
4 Selective retrieval	$\subsetneqq$	$\surd$	$\surd$	$\surd$	$\surd$
5 Overhead minimization	$\surd$[4]	$\times$[5]	$\surd$	$\surd$[6]	$\times$[7]
6 Computational efficient	$\surd$	$\subsetneqq$	$\surd$	$\times$	$\subsetneqq$[8]

Table 7.1 clearly shows that the WebDAV approach as described in [24] is best suited for credential provisioning.

[1] SAML could assert whether a certificate is valid or not.
[2] In a distributed environment, distribution needs to be taken care of.
[3] Although LDAP supports realtime updating, the used CRL is based on intervals.
[4] Based on ASN.1 BER.
[5] SAML is based on XML which is bulky by nature.
[6] Based on ASN.1 Distinguished Encoding Rules (DER).
[7] All the revocations need to be distributed
[8] Depending on the distribution mechanism

7.4 Summary

WebDAV is the most suitable for storing and retrieving certificate status.

Chapter 8

The Policy Language

In Chapter 4 we opted for AAA in combinations with PERMIS. PERMIS has its own policy language, however PERMIS is modular and can be adapted to use any policy language. [25] states that PERMIS' own policy language is more advanced than the de facto standard [81] eXtensible Access Control Markup Language (XACML). We intended to compare and select a suitable policy for use in the FedNet context, however in light of the research presented in [25] we instead will focus on the storage and retrieval of the policies in conjunction with PERMIS in this chapter.

8.1 Basic Terms

We use the following definitions taken from [126]:

- Authorization policy: A set of rules, part of an access control policy, by which access by security subjects to security objects is granted or denied. An authorization policy may be defined in terms of access control lists, capabilities, or attributes assigned to security subjects, security objects, or both.

- Access control policy: A set of rules, part of a security policy, by which human users, or their representatives, are authenticated and by which access by these users to applications and other services and security objects is granted or denied.

- Security policy: A general term covering both access control policies and authorization policies.'

Thus, in essence a policy states who — or which principal — is allowed to do what.

A *privacy policy* describe the data practices of (usually) a company. That is, it specifies what information about a user is retained, for which purposes and how the data is used [28, 87].

8.2 Assumptions

The way users can efficiently and comprehensively specify a policy such as discussed in [60, 23, 18] is out of scope of this research. Note that PERMIS comes with a policy manager, a GUI to set policies. Given that PERMIS is considered more advanced in specifying access policies than the de facto standard XACML [25, 81], we assume PERMIS' policy language is sufficiently capable to denote fine grained policies.

8.3 Storage

PERMIS has out-of-the-box support for policies retrieval via a local file store, LDAP and WebDAV. These options have been discussed in Chapter 7.

PERMIS policies are XML based, which is bulky by nature, making the local file store option preferable. LDAP and WebDAV will generate more traffic, consuming more resources. As local file storage could be out of sync and thus creating possible security issues we drop it as viable option. As WebDAV has been selected as the most suitable protocol for storing and retrieving certificate status in Chapter 7, we decide to reuse this functionality rather than adding LDAP to the mix.

PERMIS uses RBAC, which is preferable over DAC and Mandatory Access Control (MAC) [42, 41, 115]. These roles are stored in attribute certificates, which are are X.509 based and are as such implemented using ASN.1 syntax — the preferred format, as pointed out in Section 7.2.

As for the bulky XML, we suggest that this text based format is saved in a compressed form to save power consumption of the transfer. Further we suggest local caching of these policies as the HTTP protocol on which WebDAV is build can save on the needed data exchange. This caching should be flushable and caching duration should be configurable via a policy.

8.4 Summary

PERMIS has its own policy language that supports RBAC and is more sophisticated than the de facto standard XACML. A detailed analysis of different policies is therefore omitted. PERMIS supports several retrieval methods for the policies, under which WebDAV. Compression of these XML-based-policies is recommended to save power on the transfer of these policies.

Chapter 9

Design and Implementation

This chapter specifies a suitable ACA for FedNets based on the previous chapters. The architecture ensures a high degree of security and privacy to a FedNet.

In this chapter we will first discuss the assumptions and requirements for the architecture, followed by the architecture itself. Next we will discuss the prototype architecture and its components.

9.1 Assumptions

In order to define the architecture, we assume the following:

- The basic PN architecture is provided in the PNP2008 project.
- All *PN* related tasks and services other than federation management — i.e.,the FM and FA — as defined in the PNP2008 project [56] are considered in place and working. These include but are not limited to: service & content discovery, link layer security and management consoles.
- Within the PN data is ubiquitous — i.e., either data is stored centrally and universally available or a distributed database is in place.
- Medical data is sensitive personal information, which requires high security.
- Due to the mobile environment, communication (overhead) should be kept to a minimum.
- FedNets are assumed to be precreated — i.e., the FedNet setup is not part of the topotype.

- At least for non-commercial applications — privacy policies are at this point unnecessary.

9.2 System Architecture

We introduce the architecture gradually in the following sections by following the examples in the case given in Section 1.1.

9.2.1 Harry's BSK

Harry first purchases a BSK. Harry's personal cluster contains several sensors and his smartphone. His smartphone automatically switches from using WLAN at home to using the mobile phone infrastructure when needed. The Gateway (GW) acts as a NAS and can perform NAT. This GW connects to the AAA server that contains the relevant data to grant both network level access and service level access.

These physical components are depicted in Figure 9.1.

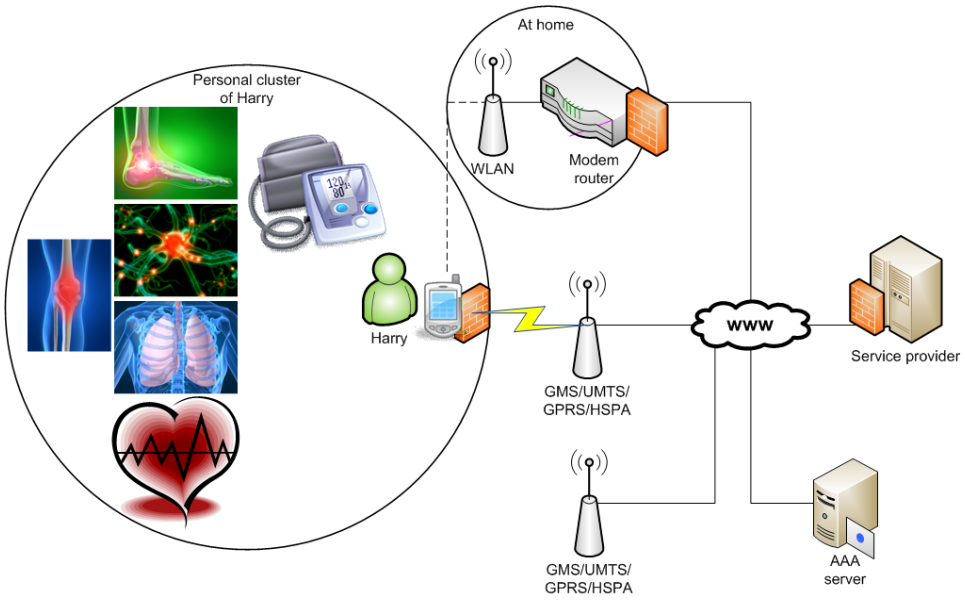

Figure 9.1: Harry's BSK.

9.2.2 The Gymnasium

When Harry goes to the Gymnasium he first needs to sign up, in the process he joins a federation that allows him to use the sensors of the Gymnasium. Harry himself talks to the clerk, while his smartphone communicates directly to the computer of the clerk. Through an out of bounds authentication

mechanism they are able to make sure that Harry's smartphone communicates securely with the computer of the clerk, allowing them to securely set up the federation between Harry and the Gymnasium. Figure 9.2 gives an impression on how this looks.

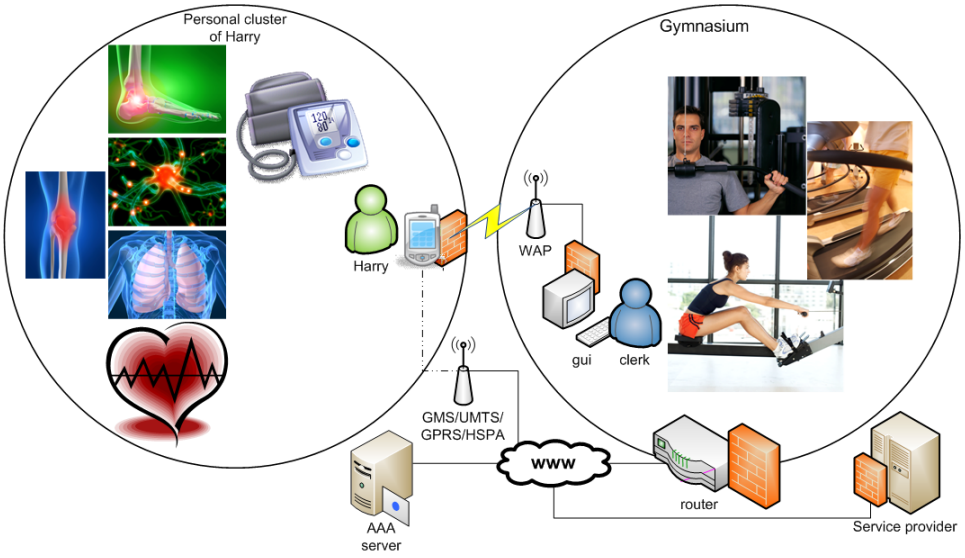

Figure 9.2: Harry Signs Up at the Gymnasium.

After signing up Harry is able to connect to the network of the Gymnasium, when he is on the premises., thereby getting access to sensors and internet connectivity. When Harry uses these facilities of the Gymnasium, the involved physical components are connected differently. This situation is depicted in Figure 9.3

9.2.3 Harry's Coach

After Harry sets up a federation with his professional coach his coach can access Harry's devices via the internet. This might be via a mobile communications network or via a modem connected to an ISP that uses a wired medium. His coach provides Harry with additional high grade sensors. One of these replaces Harry's hart rate measuring device. This situation is depicted in Figure 9.4.

9.2.4 Trouble Sleeping

Harry's sleeping disorder brings him to a specialized medical center. Here he is supplied with extra sensors and starts a federation. At this time his personal cluster and home cluster are disjoined. Nevertheless he is able to set up the FedNet like he did at the gymnasium. An example of this situation is given in Figure 9.5.

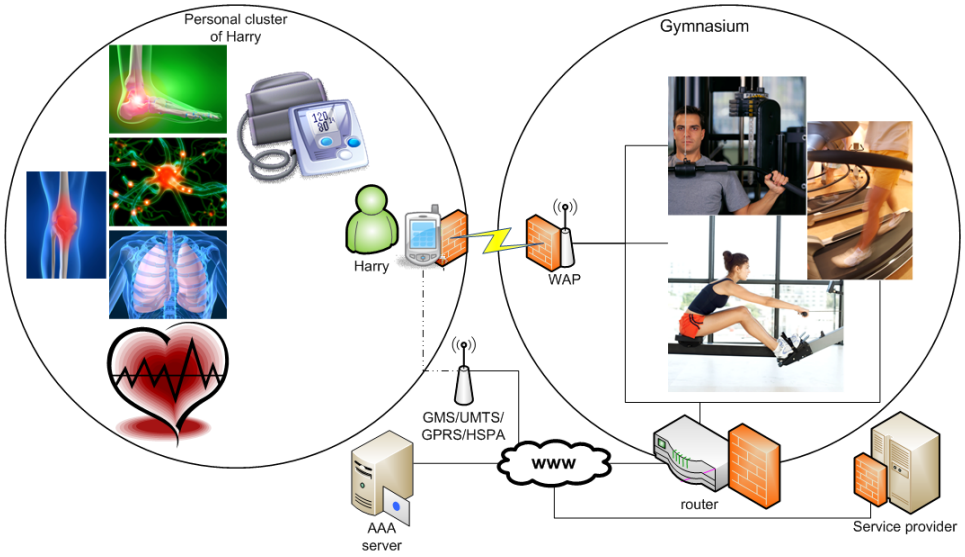

Figure 9.3: Harry Trains at the Gymnasium.

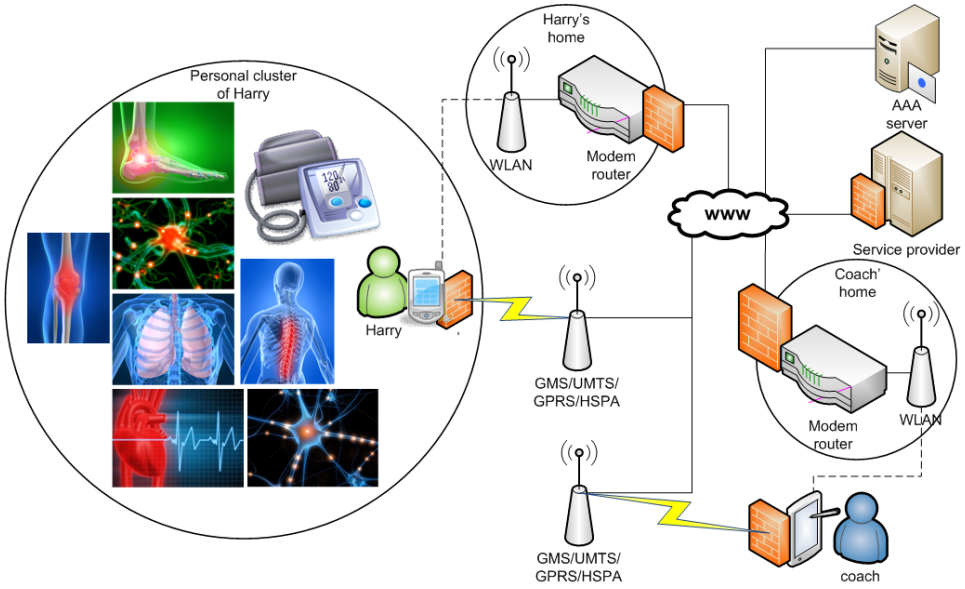

Figure 9.4: Harry and His Professional Coach.

At night, when Harry is sleeping, another doctor in the hospital can monitor the sensors in Harrys PN and read up on relevant medical dossiers stored in an EPD. Even when Harry's modem is down due to maintenance at the ISP the doctor can still connect via Harry's smartphone. This situation is depicted in Figure 9.6.

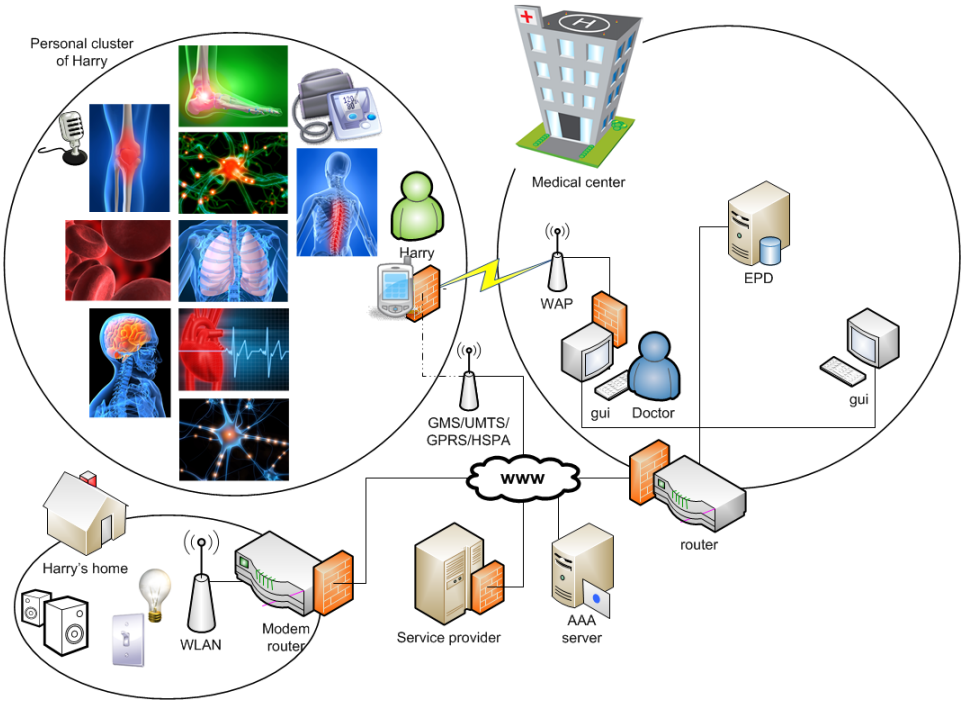

Figure 9.5: Harry Signs Up at the Hospital.

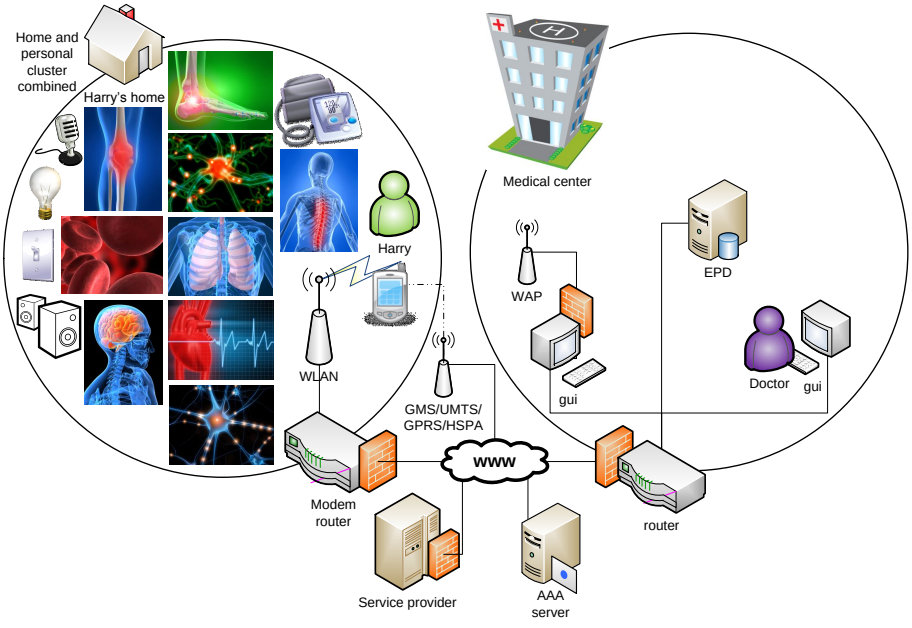

Figure 9.6: Harry Is Sleeping at Night.

9.2.5 The FedNet View

Having introduced all the physical entities we now look at the components from the FedNet view. We start with an simplified overview in which two persons wish to share a service with each other. This is depicted in Figure 9.7.

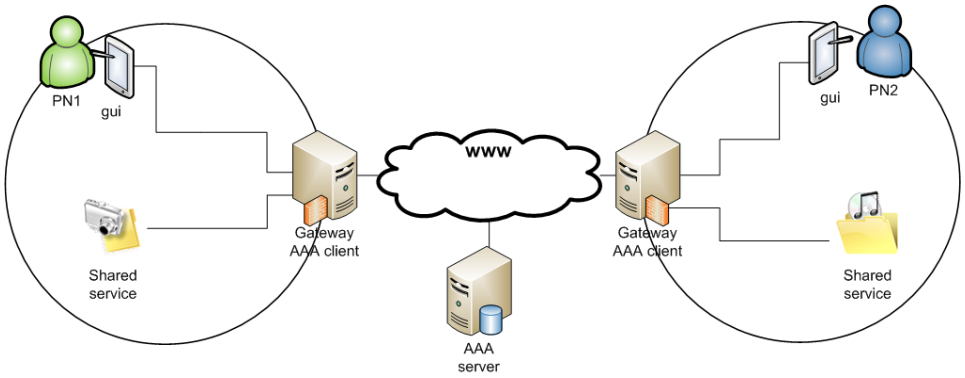

Figure 9.7: High Level Architecture.

Next we look at the *logical* components involved in this situation. The gateway functionality is performed by strongSwan, taking care of the required NAS, NAT and routing functionality. The persons in this federation can set policies and configure access control rights via their GUI, the functionality is taken care of by the FedNet Agent (FA). The FA relays this information — provided with a digital signature — to the FedNet Manager (FM) which then stores this information in the WebDAV database. The information in the WebDAV database is used by the AAA server to determine access control. Communication between the PNs is protected by an IPsec tunnel between the PNs gateways. This tunnel is set up by strongSwan through the EAP-IKEv2 protocol. The strongSwan gateway, acting as a Network Access Server (NAS), communicates with the the AAA server using the Diameter protocol. As required in [21] the communication is secured using IPsec or TLS. The communication between the FA and FM is also handled over a secure channel. This process is depicted in Figure 9.8.

9.2.6 Putting It All Together

We have combined all of the above into one figure, Figure 9.9. In this figure the home cluster is merged with Harry's personal cluster, two doctors that have helped Harry are displayed, the clerk at the gymnasium is omitted. A third party service provider provides the AAA server and FM functionality.

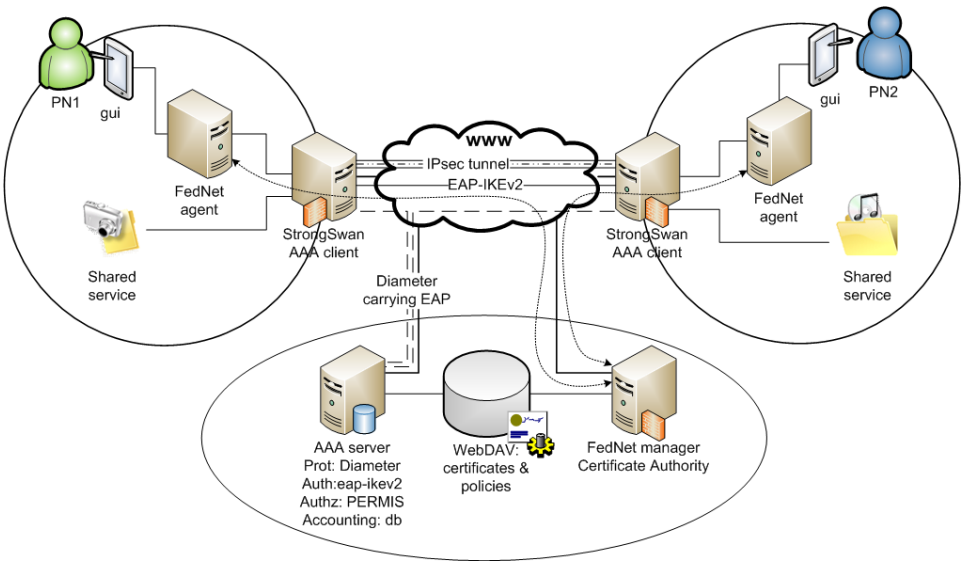

Figure 9.8: Detailed Functional Component Architecture.

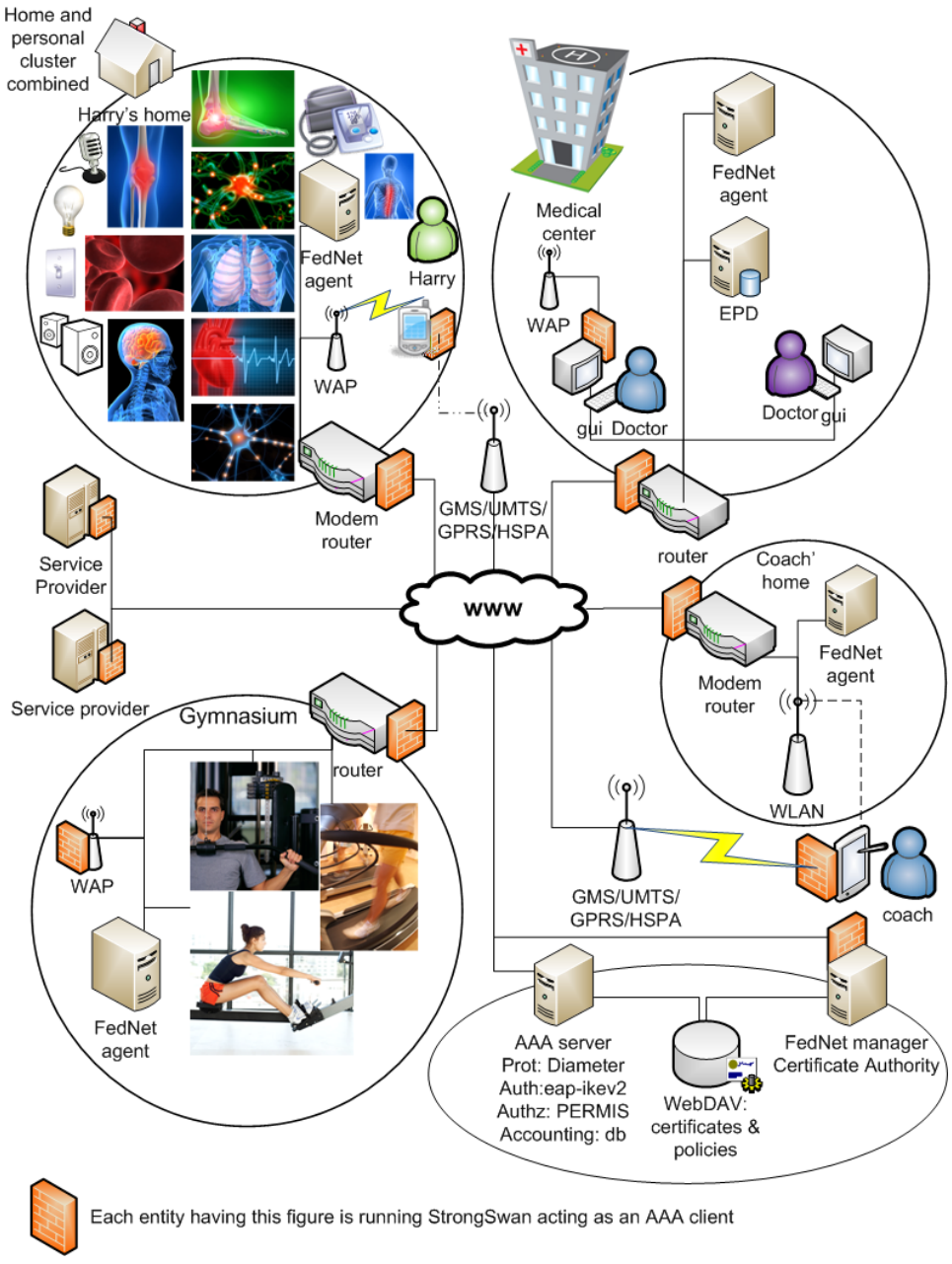

Figure 9.9: The Use Case as a Whole.

9.2.7 AAA Server Placement

In the previous subsections we described the AAA server as a central external entity. However, [55, Section 4.2] describes two scenario's for the location of the FM: the FM being located either external to both PNs, as depicted in Figure 9.10, or internal to one of the PNs, as depicted in Figure 9.11. [55] states that each PN must have a FA component.

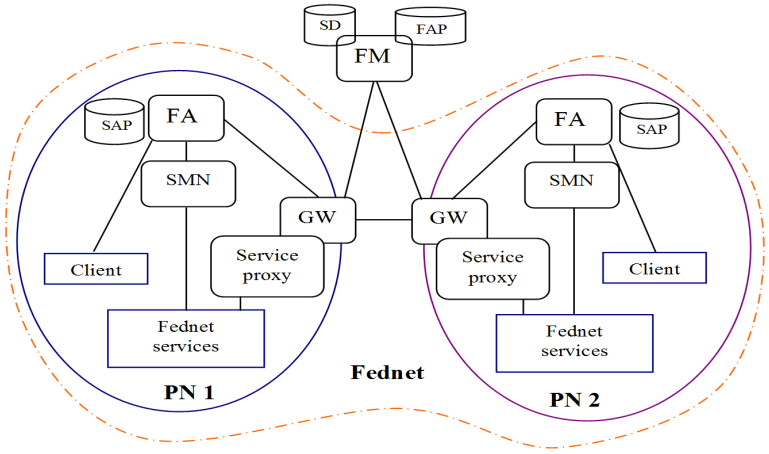

Figure 9.10: Basic Proxy-Based-Architecture of a FedNet with an External FM. Figure Reproduced from [55] with Permission from the Authors.

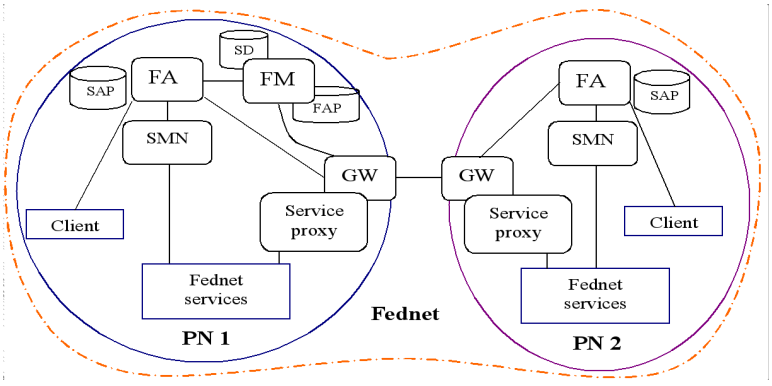

Figure 9.11: Basic Proxy-Based-Architecture of a FedNet with an Internal FM. Figure Reproduced from [55] with Permission from the Authors.

Based on the above we derive five distinct architectural solutions. In all cases each PN has at least the FA functionality.

1. The FM functionality is provided by a third party. As described in Figure 9.10.

2. Each PN has both the FA and FM functionality. No third party or central authority is used. As described in the left part of Figure 9.11. Which PN's FM is used is based on which of the PNs is considered the manager of the FedNet.

3. One of the PNs has FM functionality whereas the other does not. No third party or central authority is used.

4. Each PN has a FM integrated and a third party provider exists.

5. Each PN has a minimum of the FA functionality and *might* have the FM functionality. A third party FM is needed to facilitate FedNets that are managed by the FA that does not have its own FM.

Table 9.1: Architecture Option Comparison.

Solution	1	2	3	4	5
Supports ad-hoc FedNets	×	√	√	√	√
Supports third party infrastructure based hosting	√	×	×	√	√
Allows for 'light' PN implementations without FM	√	×	√	×	√

Table 9.1 compares relevant requirements against these 5 architectural solution for FM placement. Table 9.1 clearly shows solution 5 is the most suitable as it is the only option that meets all the requirements.

Solution 5 provides the most flexibility. Therefore we consider this architecture the most suitable to implement. It allows for a PN design that supports federating yet does not necessarily have the FM functionality integrated, allowing for lighter implementations in early systems and resource constrained devices. It has the disadvantage that, in the case of having no internal FM, the PN cannot start and manage its own FedNet without an infrastructure and a third party provider. Another disadvantage is that in the ad-hoc-scenario the PN cannot federate unless the other PN has a FM. In this case the other PN automatically is the manager of the FedNet.

9.3 Proposed Architecture

We reason that both the FM and FA have similar functionality and therefore opt to integrate their implementation.

The functions of the FM are [55]:

- Management function - to create FedNet profile and advertisements; to manage the formation, evolution and dissolution of a FedNet, to maintain the list of FedNet members and their contributions, experiences during the FedNet operation.

- Service directory look up function - to provide the directory service to FedNet members, to maintain the updated list of members and services in the FedNet.
- FedNet access control function - to produce decisions on the access control to the FedNet; to issue membership credentials for FedNet members.

The functions of the FA are [55]:

- Management function - coordinating PN services to which a PN can grant a temporal access within a FedNet, managing the participation of a PN in a FedNet, creating a participation profile, joining and leaving a FedNet.
- Service access control function - to produce access control decisions to PN services.
- Service proxy configuration function - to configure a requested service proxy to a particular member in a FedNet.

The management function: can be implemented as a GUI to define, sign and store policies and to create and sign certificates. The access control function can be implemented using the PERMIS authorization engine in combination with certificate based authentication. We reason that without an actual proxy this can be implemented as a policy stating who may access what. As stated in the beginning of this chapter, we assume PN-internal (intra-PN) routing and look-up are in place. Implementations are suggested in for example [67] and [68].

[55] defines the GW as follows: "The Gateway (GW) is a device with multiple network interfaces. A PN communicates with other PNs of the FedNet through this gateway, by making one of its interfaces publicly addressable."

We implement the GW using strongSwan, without implementing a proxy. We reason that a proxy might still be integrated into our design if required and that the accessed service of a personal node could implement security by using the PERMIS reasoning engine. The node need not derive access control decisions, instead it relies on PERMIS for these. This might be integrated into the operating system of the node so that the shared service itself need not be adopted nor will there be the need for developing a proxy for each application.

Based on [69] we reason that our proposed architecture can be used, if properly adapted, for inter-cluster security as well. As each cluster will have its own engine, we recommend that the security agent [8, 69] be integrated into our architecture.

Finally we reason that an external FM as described in [55] could be implemented as a third party PN with an internal FM, thus making the choice for architecture based on a internal FM even more logical.

We add that for revocation status and storage we propose the use WebDAV for certificates as described in [24]. Policies will be stored in WebDAV as well.

Concluding we propose an architecture with:

1. An FM component internal of a PN.
2. Each FM is a CA, which can be either selfsigned, based on a PKI or reputation based.
3. The FM and FA be implemented using the same software (PERMIS engine and policies, certificates).
4. Policies, revocation status and certificates are stored using WebDAV.
5. The GW being implemented using strongSwan.

This slimmed-down-architecture is depicted in Figure 9.12.

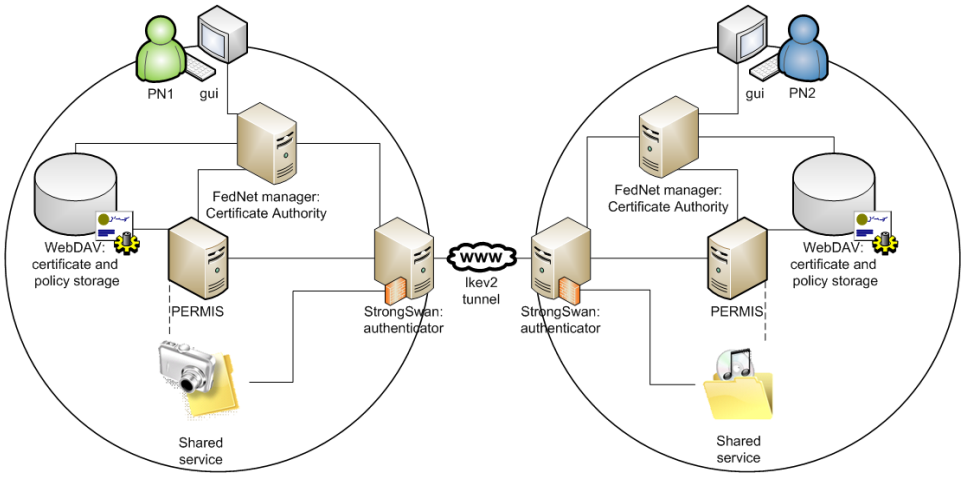

Figure 9.12: Proposed Architecture.

At first glance the architecture depicted in Figure 9.12 might look completely different from the architecture depicted in Figure 9.8, however the alterations are quite straightforward. First the (common) AAA server is moved from outside the PN to inside each PN. Second, the FM and FA are integrated. Third, the AAA server is replaced by PERMIS. Fourth and finally, the GUI has been reduced to a command line interface. We envision each cluster having a AAA/PERMIS engine with the database either being distributed throughout the PN's clusters or centralized in the home cluster.

This approach comes with some drawbacks. The location of the PERMIS engine might change, making it more difficult for FedNet members to connect to the managing entity of the relevant FedNet. This problem can be

mitigated by making the service stationary, e.g. in the home cluster. Another approach could be to make the home cluster take care of redirecting to the current location of the service. Also a PN could incorporate a PERMIS engine in each cluster, using a distributed (WebDAV) database to store the relevant policies and certificates.

This approach also comes with benefits. Functional components can be grouped together and run on the same physical machine. This will decrease energy consumption, as the communication between the functional components does not have to be transmitted. In Figure 9.13 the most communicating entities are grouped together in the overlapping section of the circle and ellipse. The GUI has been omitted from this grouping as it runs per definition where the user is.

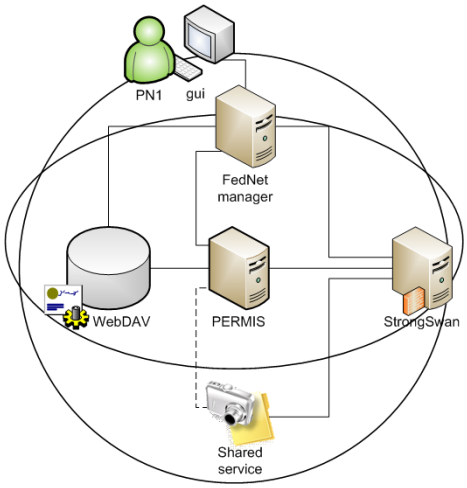

Figure 9.13: Grouped Components of the Proposed Architecture.

Also services can directly query the PERMIS reasoning engine wether a certain action is allowed or not. The questioning can be done either by a process running on a shared service or a service proxy.

9.4 Prototype

Implementing the whole architecture proposed in the previous section is out of scope of this thesis. The prototype we will provide is slimmed down architecture that proofs the concept.

In this section we discuss the functional components used in the prototype.

1. **Shared Service**
   A shared service can be any kind of resource (i.e., photos and videos)

or service (i.e., printing services, internet connectivity, displays) that has been configured to be shared.

2. **GUI**
   Trough the GUI the user — i.e., owner — , can configure which of his resources and services are to be shared with whom under what circumstances. The GUI acts as a front-end for the FM. In our prototype we will minimize effort on the GUI as we contribute a GUI to usability, not security.

3. **FedNet Manager**
   The FM processes the input the user provided via the GUI concerning FedNets. During this process the FM can generate and sign policies and certificates. The FM can retrieve, update and store these policies and certificates in the WebDAV repository. The FM in the prototype incorporates the FM and FA functionality as described in Section 9.2.7. Whereas the FM should provide automated management, we will mock its behaviour in the prototype. We will manually configure and place the FedNet certificates and policies on the relevant local file systems.

4. **WebDAV**
   We have chosen WebDAV to store and retrieve certificates and policies. A distributed WebDAV implementation in a FedNet might look like Figure 9.14.

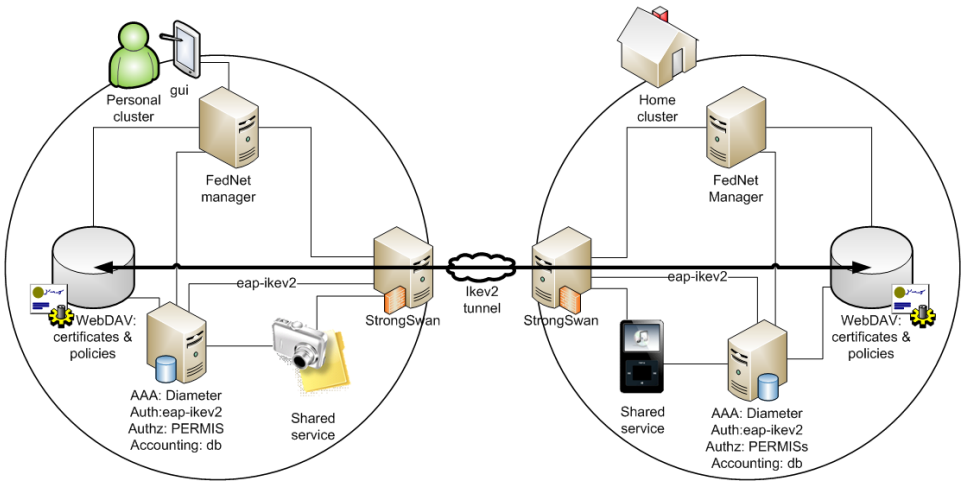

Figure 9.14: Distributed WebDAV Repository.

WebDAV is an extension to the HyperText Transfer Protocol (HTTP) 1.1 protocol. On the server side WebDAV can be enabled by simply adding the module to the webserver, which is available for several webservers. A client does not need the extension to retrieve information

for the server, it is only needed to write something to the server. The automated and secure storage on a WebDAV server is removed from the prototype in favor of other components' implementation as our time is limited.

When retrieval fails the returned error code can be used to determine how to proceed.

5. **PERMIS**
In Section 4.6 PERMIS was chosen for its extensive authorization mechanisms that can be used for each end every action a user performs. In our prototype we will use PERMIS in a simple sharing setting to test PERMIS usability in FedNets.

   PERMIS is opensource and at the time of writing it has been integrated with several applications. In fact for some applications there is no need to write any code as PERMIS is already embedded in them. All you need to do is download and install PERMIS along with [101]::

   - Globus Toolkit (v3.3 onwards); PERMIS authorization can control access to Grid Services.
   - Apache; PERMIS authorization can protect web sites.
   - Shibboleth;. PERMIS authorization can be combined with Shibboleth's Single Sign-On to provide policy driven fine grained role based access controls within federations.
   - .Net; PERMIS authorization can be combined with Microsoft .Net to authorize web services.
   - Python; PERMIS authorization can be called from Python to provide the access controls for Python programs.

   As we want to use PERMIS in conjunction with strongSwan we will have to integrate these applications.

   We installed openjdk-6-jdk in order to be able to compile and run java classes.

6. **strongSwan**
strongSwan performs the gateway functionality. In Chapter 5 we decided to use the EAP-IKEv2 protocol, but did not choose which implementation to use. StrongSwan is our implementation of choice. It is open source and is actively maintained. At the time of writing the available version is compatible with: the IKEv2 [73] protocol, the Mobility and Multihoming Protocol (MOBIKE)[36] protocol, strong AES encryption, ECDH and ECDSA protocols [76], relaying EAP messages to AAA servers, multiple IKEv2 authentication exchanges [37], X.509 certificate based authentication, group policies based on X.509 attribute

certificates [40], OCSP[91], retrieval and local caching of CRLs via HTTP or LDAP and much more [2]. In other words it has almost all of what we need.

Most importantly strongSwan lacks support of EAP-IKEv2. This is mitigated by the lack of available AAA servers that support EAP-IKEv2 and the usage of PERMIS instead of an AAA server. Therefore we will use the IKEv2 protocol instead. The authentication is not longer done by the AAA server, yet performed by the gateway. PERMIS will still perform the authorization.

We adapt strongSwan to be able to perform certificate status request using a WebDAV server and to perform a PERMIS check.

7. **AAA**
   In the end we have chosen AAA as the ACA to base our solution on. When we started our work there was no AAA implementation available to support EAP-IKEv2. We omit the AAA architecture in our prototype in favor of proving that our suggested additions and criteria can work together.

8. **Prototype**
   Our prototype consists of one system running strongSwan PERMIS and Apache and one running strongSwan only. Both systems run a SSH deamon for test setup automation. Each strongSwan instance has the necessary signed certificates to establish a connection, the corresponding keys and configuration in its own directory on the local filesystem *in directory X. Permis has the policy file and the attribute certificates in its own directory on the local filesystem* in a different directory Y. WebDAV is not actually used, as Apache impersonates the WebDAV server and all the files are created before testing and stored in the local file system. For proof of concept the retrieval without making use of the WebDAV protocol *in a seperate directory Z. Figure 9.15 depicts these prototype components. The GUI, FM, AAA and shared service components are ommitted in the prototype.*

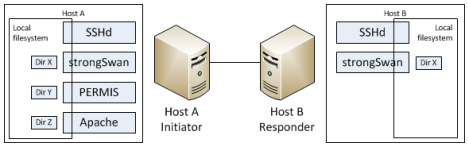

Figure 9.15: Prototype components.

Aside from the ommited components, compared to Figure 9.14 the prototype consolidates several sevices on one host. the components that haGUI, FM, AAA and shared service components are ommitted in the prototype.

9.5 Summary

In this chapter we combined the conclusions of the previous chapters and proposed a framework for secure access control in the context of a FedNet and have shown how this matches to the FedNet architecture.

We also provided our prototype architecture. For the prototype we omit an AAA implementation, adapt strongSwan so that it can perform the improved certificate revocation check and use PERMIS for authorization. The policy editor that comes with PERMIS is used as GUI and certificates will be generated via command line. In the prototype there is no automated FM that provides these files, instead these files will be manually put on local file systems.

Chapter 10

Prototype Evaluation

This chapter presents analysis of the prototype.

Section 10.1 discussed functional testing. Section 10.2 presents performance measurements of our prototype. Section 10.3 reflects on extensions of the platform with new components such as sensors, actuator and personal devices. Section 10.4 reflects on scalability with regard to new applications. Finally, we summarize our findings in Section 10.5.

10.1 Functional Testing

This section presents the functionality testing of the prototype. The functionality tests will accomplish to test the functionality of the components introduced in Section 9.4. Authentication (strongSwan), authorization (PERMIS) and certificate revocation (WebDAV) are presented in Sections 10.1.1, 10.1.2 and 10.1.3 respectively.

10.1.1 Authentication

With regard to the impact of our modifications to strongSwan on the functionality of the authentication of strongSwan, we looked at how the prototype behaves in the following cases:

- Compatibility with non adapted strongSwan, i.e. do the changes made to strongSwan result in unexpected behavior.

Due to time limitation we omitted testing:

- Changing IP (mobike).

We verified that a modified version of strongSwan works in conjunction with a unmodified one.

10.1.2 Authorization

With regard to the impact of our modifications to strongSwan for authorization on the functionality of strongSwan, we looked at how the prototype behaves in the following cases:

- Compatibility with non adapted strongSwan, i.e. do the changes made to strongSwan result in unexpected behavior.
- Check wether certificates are susceptible to different order of arguments, case sensitive, spaces.
- Check wether it accepts valid certificates and reject invalid ones.

Due to time limitation we omitted testing:

- Changing IP (mobike).
- Delegated authorization (chained certificate).

We found that a server that incorporates our modifications is compatible with a non adapted client. The extra messages that are introduced will be displayed as a number instead of a the human readable version that a modified strongSwan will show.

We found that an invalid certificate results in the expected error message during connection setup. After the IKE_SA_INIT phase, during the IKE_AUTH phase and before the CREATE_CHILD_SA phase of the connection setup, an authentication failed message will be send and the connection will be terminated. No child security association will be created.

10.1.3 Certificate Revocation

With regard to the impact of our modifications to strongSwan on the functionality of the certificate revocation checking, we examined the following:

- Compatibility with non adapted strongSwan, i.e. do the changes made to strongSwan result in unexpected behavior.
- Check how strongSwan handles the OCSP WebDAV handle when it cannot connect (connection denied)
- Check how strongSwan handles the OCSP WebDAV handle when it should be OK
- Check how strongSwan handles the OCSP WebDAV handle when wrong server is set (retrieved trivial reply)
- Check how strongSwan handles the OCSP WebDAV handle when the certificate is revoked

- Changing IP (mobike)

The introduced WebDAV revocation check only works for those systems that have the modification. strongSwan systems that do not have the modification, or configured it to not perform WebDAV, will parse the certificate and attempt to perform a regular OCSP status request. When no OCSP server is registered at the Uniform Resource Locator (URL), strongSwan falls back to normal certificate checking as were there no OCSP info available.

When the WebDAV modification is available and turned on and the certificate is valid, we found that if the WebDAV server is unavailable for any reason, strongSwan returns an authorization failed error to the initiator and terminates the connection.

When the WebDAV modification is available and turned on and the certificate is valid, we found that when the WebDAV grants authorization, strongSwan returns a authentication granted reply as designed.

We found that when the certificate is revoked, strongSwan does not perform a WebDAV authorization check. As the certificate is invalid, the process of setting up a connection is terminated before it gets to the point of WebDAV checking.

10.2 Prototype Performance

This section presents the performance testing of our prototype. The goal of this section is to provide an insight of the performance behavior of our prototype. To this end several experiments are conducted and presented.

To simulate the resource scarce environment in which the architecture is to be used, a resource constrained device is used possessing less performance then that of a PDA such as described in scenario 1.1. As this device turned out to be too resource-constrained to perform all experiments we were forced to perform some of the experiments on the more resourceful system only.

Our main interest lies in the performance in terms of latency. We determine the impact on this performance measures by examining the influence that the AAA subprocesses, background load, used hardware and used communication medium have. To that end we conducted and present the following experiments:

1. Provide a baseline using unmodified strongSwan clients. This experiment is presented in Section 10.2.2.

2. The impact of authentication, authorization and certificate revocation status checking on latency. How do the different elements of our prototype affect the performance? This experiment is presented in Subsection 10.2.3.

3. The impact of server hardware on latency. How will performance of the prototype be effected when different server hardware is used. This experiment is presented in Subsection 10.2.4.

10.2.1 General Experiment Setup

All experiments are run on two devices: a client that initiates the connection and a server that is the responder to the connection request.

To this end, we have three devices at our disposal.

1. A wireless router, a Linksys WRTGSv1.0, running at 200 MHz (Broadcom BCM4712 RISC), 8 MB flash and 32 MB RAM. We use this as our resource constrained device.

2. Host A, a quad core system running at 2.8 GHz (Intel Core i7 860), 8 GB RAM and a 320 GB HDD. The resource rich device.

3. Host B, a single core system, running at 2.2 GHz (AMD 3200+), 512 MB RAM and a 640 GB HDD. An average resourceful system.

More detailed descriptions of the systems with regard to hardware and running processes are presented in Appendix C.

For all experiments a wired LAN connection is used to minimize potential interference. In all experiments we used a prefabricated 50 centimeter long category 5e UTP cable.

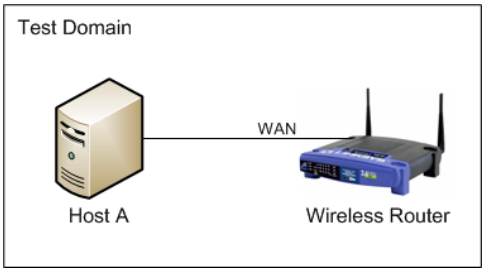

Figure 10.1: Example Test Setup

Figure 10.1 portraits a example setup where host A is connected to the WAN-port of the wireless router by means of a network cable. The wireless connection of the router is not used.

10.2.1.1 Assumptions

The following assumptions apply for all experiments. It is assumed that:

- Federations are provided beforehand — I.e., the necessary certificates and policies are already in place.

- All connections attempts should be accepted by the prototype.
- Requests are generated based on a Poisson process. Therefore inter-generation time follows an exponential distribution.
- The wireless router is a PDA like device qua performance.
- The NIC is set to run fast ethernet (IEEE 802.3u, 100BASE-TX) or gigabit ethernet (IEEE 802.3ab, 1000BASE-T) in full duplex mode (IEEE 802.3x).

10.2.1.2 Performance Parameters

The following performance parameters apply for all experiments:

- *Working threads:* By default, strongSwan is configured to use 16 working threads. We configured strongSwan to use 64 threads as our vigorous testing proved too much for strongSwan[1].
- *Connection medium:* a 50 centimeter long prefabricated category 5e UTP cable.
- *Load:* The amount of requests generated on average per second.
- *Background server load:* The percentage of CPU utilization on the server that is not related to activities under test.
- *Authorization:* Authorization checking can be configured in our prototype.
- *OCSP:* Certificate status checking can be configured in our prototype.
- *CRLs:* strongSwanwill have no CRLs to check against.

10.2.1.3 Performance Measures

We define the following performance measures:

- *Latency:* The time it takes for a client request to be accepted, as seen by the client from the moment of request initiation. This latency includes authentication and depending on the experiment might include authorization and / or a certificate revocation status check.
- *Total latency:* When the Latency described above includes authentication, authorization *and* certificate revocation status checking we use the term Total Latency.

[1] Tobias Brunner kindly provided a patch that made vigorous testing possible at all.

- *Authentication latency:* The time it takes for a client for a request to be accepted, as seen by the client from the moment of request initiation. The request only entails an authentication check; authorization and certificate revocation checking is omitted.

- *Authorization latency:* The time it takes for a client for a request to be accepted, as seen by the client from the moment of request initiation. This request is checked by the server for both authentication and authorization. As the authentication cannot be omitted, the measurement of authentication alone is subtracted from the measurement of authentication and authorization combined to derive the added latency due to authorization.

- Combined Authentication and Revocation latency: The time it takes for a client for a request to be accepted, as seen by the client from the moment of request initiation. This request is checked by the server for both authentication and certificate revocation status check.

10.2.2 Experiment 1: Baseline (Non-Modified) System Total Latency

The goal of this set of experiments is to provide a baseline for comparison to the modified version of strongSwan.

Setup and Common Settings In Section 10.2.1 we presented assumptions that apply to all the experiments as well as performance parameters and performance measures. This section states the setup and the additional performance parameters that apply to experiment 1.

The authentication latency is measured by using a strongSwanversion that is as pure as possible.

Performance Parameters Additional to the performance parameters presented in Section 10.2.1 the following parameters apply for the experiments in this section:

- Host A is used as client.

- Host B is used as server.

- The background load of both client and server is minimized.

- Experiments are performed under different amounts of load.

To minimize interference the used devices are only connected to each other. Figure 10.2 displays the setup of experiment 1.

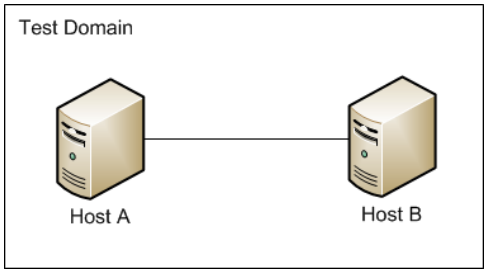

Figure 10.2: Experiment 1 Setup

Performance Measures The following performance measures apply for this subexperiment:

- Total latency as described in Section 10.2.1.3

Approach We compile strongSwan with none of the additional modifications, except for the patch provided by Tobias Brunner[2] We compile using a minimal set of compile options in which openssl is included for ECC support.

We will use the same certificates as for the other experiments, meaning that the certificates contain a reference to an OCSP server — Host B. As there is no actual OCSP service running on Host B, the verification attempts will fail and strongSwan will default to normal certificate validation checking using CRLs.

In order to get representative results the latency is measured by repeating the following steps:

1. Increase the load.
2. Restart the involved programs, strongSwan, Apache and or PERMIS, on both systems.
3. For a amount of time, being six minutes, do:
   (a) Based on the load initiate requests. The intervals between the request are generated using a exponential time distribution. These requests are in essence federations and are chosen every time in the same order from the set of preconfigured federations to allow for better comparison.
   (b) Each of these initiated connections is measured using the Portable Operating-System Interface (POSIX) 'time' command at the initiating client.

[2]During our experiments we found that our rigorous testing caused a deadlock by starvation. Tobias Brunner kindly supplied a patch allowing for configuration of the relevant resource.

(c) Save the results for later analysis.

We choose a 360 second duration for testing to allow for a warmup phase of 120 seconds, getting enough results to analyze and limit the total test time. Even with the limited duration of 360 seconds and taking any other factors out of the picture, such as human configuration error, performing all the tests in this Chapter result in about a week of lead time.

Results Of each load — i.e. number of initiated connection request per second on average, exponentially distributed — we calculate the average connection time. To this end we disregard any connections initiated in the first two minutes to allow the system to warm up. The connections that have been initiated but are not connected yet are disregarded too. This average is calculated over each of the ten reruns of this experiment. As these ten reruns have a different seed for the exponential distribution generation we can calculate a confidence interval over these averages, which are presented in Appendix D.1. Figure 10.3 shows the average total latency results versus load (number of initiated connections per second) when the baseline (non-modified) system was used.

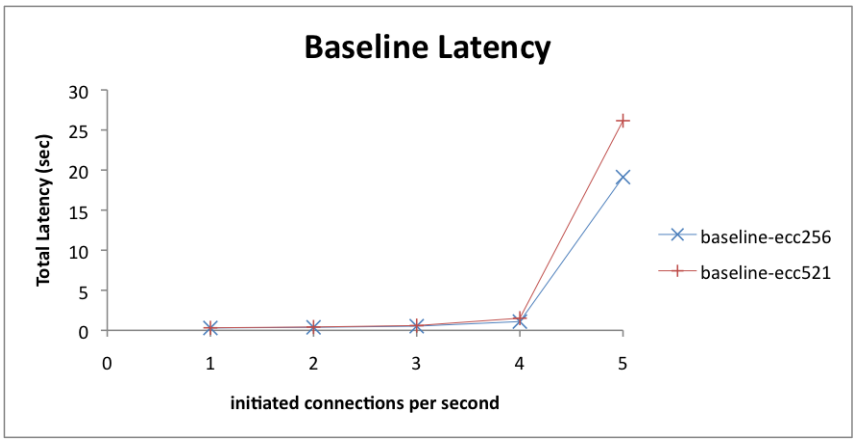

Figure 10.3: Experiment 1: Average Total Latency for Baseline (non-Modified) System

Figure 10.3 shows a significant increase in latency time starting at 4 connections per second, meaning that at around four connections per second the system starts to overload. For the experiment using 256 bits certificates 33741 connections were successfully initiated in the allotted timeframe and none timed out. For the experiment using 521 bits certificates 33158 connections were successfully initiated in the allotted timeframe and 232 timed out. Upon further inspection of the test results we discovered that all these timeouts occurred at five initiated connections per second.

10.2.3 Experiment 2: Authentication, Authorization, and Certificate Revocation of Modified System.

The goal of this set of experiments is to determine the impact on the latency by the authentication, authorization, revocation functionalities and used keystrenght on the prototype. To that end four subexperiments are performed. Section 10.2.3 presents the setup and common setting for all the subexperiments. Section 10.2.3.1 presents the authentication latency subexperiment. Section 10.2.3.2 presents the authorization latency subexperiment. Section 10.2.3.3 presents the certificate revocation latency subexperiment. Finally Section 10.2.3.4 presents the total latency and different keystrenghts.

Setup and Common Settings In Section 10.2.1 we presented assumptions that apply to all the experiments as well as performance parameters and performance measures. This section states the setup and the performance parameters that apply to the set of experiments in Section 10.2.3.

Performance Parameters Additional to the performance parameters presented in Section 10.2.1 the following parameters apply for all the experiments presented in Section 10.2.3:

- Host A is used as client.
- Host B is used as server.
- The background load of both client and server is minimized.
- Experiments are performed under different amounts of load.

To minimize interference the used devices are only connected to each other. This experiment uses the same setup as experiment 1, see Figure 10.2.

10.2.3.1 Experiment 2.1: Authentication Latency

Our modifications to strongSwan are configurable and can be turned of. By configuring the server not to perform authorization and not to check for certificate revocation status, it only performs authentication — which cannot be turned off . We use this to measure authentication latency.

Performance Measures The following performance measures apply for this subexperiment:

- Authentication latency as described in Section 10.2.1.3

Approach This subexperiment will use the same approach as experiment 1, presented in the Section 10.2.2.

Results Of each load — i.e. number of initiated connection request per second on average, exponentially distributed — we calculate the average connection time. To this end we disregard any connections initiated in the first two minutes to allow the system to warm up. The connections that have been initiated but are not connected yet are disregarded too. This average is calculated over each of the ten reruns of this experiment. As these ten reruns have a different seed for the exponential distribution generation we can calculate a confidence interval over these averages, which are presented in Appendix D.2.1. Figure 10.4 shows the average authentication latency results versus load (number of initiated connections per second) when the modified system was used.

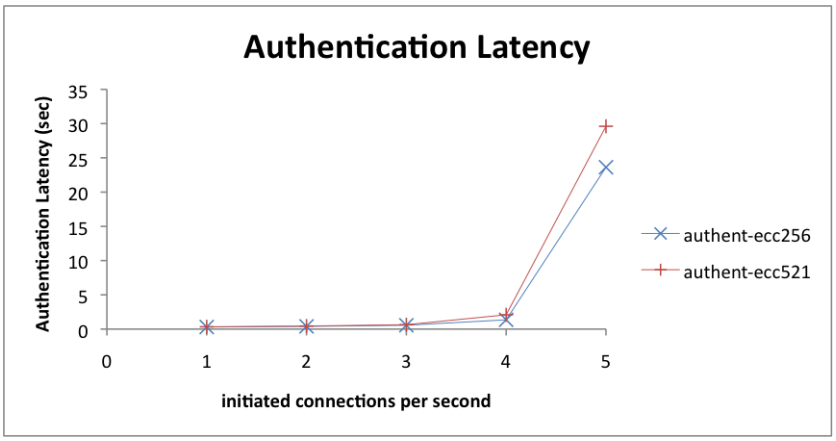

Figure 10.4: Experiment 2.1: Average Authentication Latency for Modified System

Like in the previous experiment, Figure 10.4 shows a significant increase in latency time starting at 4 connections per second.

For the experiment using 256 bits certificates 33368 connections were successfully initiated in the allotted timeframe and 142 timed out. For the experiment using 521 bits certificates 32819 connections were successfully initiated in the allotted timeframe and 671 timed out. Upon further investigation we found that all these timeouts occurred at five initiated connections per second.

10.2.3.2 Experiment 2.2: Authorization Latency

In our prototype, the authorization process takes place during the authentication handshake. Therefore the impact of the authorization can only be

determined by subtracting the previously measured authentication latency from the combined latency of the authentication and authorization processes.

Performance Measures The following performance measures apply for this subexperiment:

- Authorization latency as described in Section 10.2.1.3

Approach The server is configured to perform authorization and not to perform certificate revocation status checking. It therefore performs authentication and authorization without revocation checking. measurements are taken using the same approach as stated in Section 10.2.2. The differentiation of this measurement with the previous one, measuring authentication latency, is used to determine the impact of the authorization.

Results We intended to run this experiment using one to five initiating connections per second, however the resulting system overload forced us to use different values. As a result we limited this experiment to go to two initiating connections per second, using smaller intervals.

Of each load — i.e. number of initiated connection request per second on average, exponentially distributed — we calculate the average connection time. To this end we disregard any connections initiated in the first two minutes to allow the system to warm up. The connections that have been initiated but are not connected yet are disregarded too. This average is calculated over each of the ten reruns of this experiment. As these ten reruns have a different seed for the exponential distribution generation we can calculate a confidence interval over these averages, which are presented in Appendix D.2.2.

Figure 10.5 shows the average Authorization latency results versus load (number of initiated connections per second) for the situation that the encryption algorithm uses 256 bits long certificates and when the Modified system is used. The curve labelled as authent-author-ecc256 shows the measured average values of the composed authentication + authorization latency for the situation that the encryption algorithm uses 256 bits long certificates. The curve labelled as authent-ecc256 shows the measured average values of the authentication latency for the situation that the encryption algorithm uses 256 bits long certificates. The curve author-ecc256 depicts the difference between the curve authent-author-ecc256 and the curve authent-ecc256.

Figure 10.6 shows the average authorization latency results versus load (number of initiated connections per second) for the situation that the encryption algorithm uses 521 bits long certificates and when the modified system is used. The curve labelled as authent-author-ecc521 shows the measured average values of the composed authentication + authorization latency for the situation that the encryption algorithm uses 521 bits long certificates.

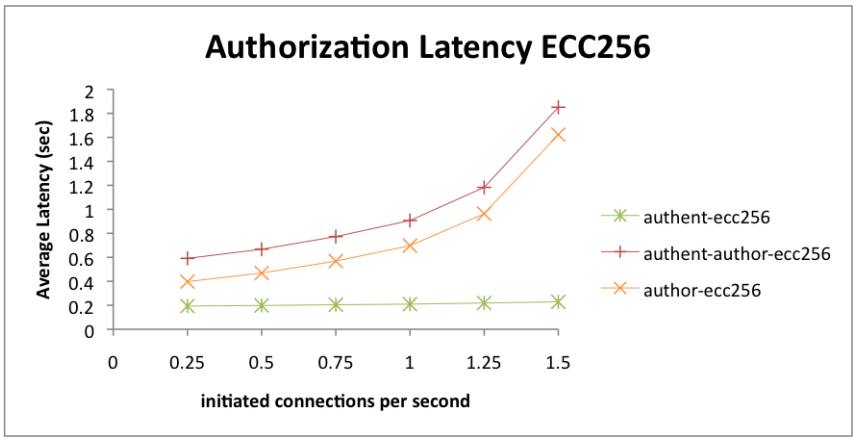

Figure 10.5: Experiment 2.2: Average Authorization Latency for Modified System when 256 Bits Certificates Are Used

The curve labelled as authent-ecc521 shows the measured average values of the Authentication latency for the situation that the encryption algorithm uses 521 bits long certificates. The curve author-ecc521 depicts the difference between the curve authent-author-ecc521 and the curve authent-ecc521.

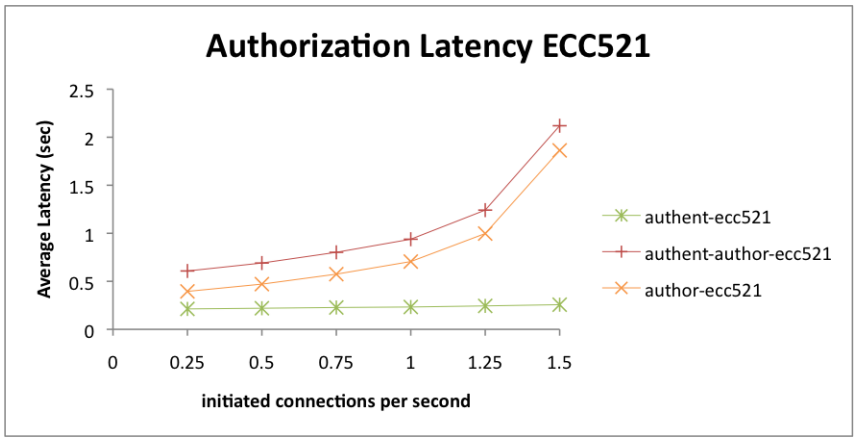

Figure 10.6: Experiment 2.2: Average Authorization Latency for Modified System when 521 Bits Certificates Are Used

Figure 10.7 shows the average Authorization latency results versus load (number of initiated connections per second) when the Modified system is used.

Figure 10.7 shows a significant increase in latency time starting at around 1.75 connections per second. The added authorization results in a system in overload at two connections per seconds comparable to five connections per second without authorization. Meaning that the modification for improved

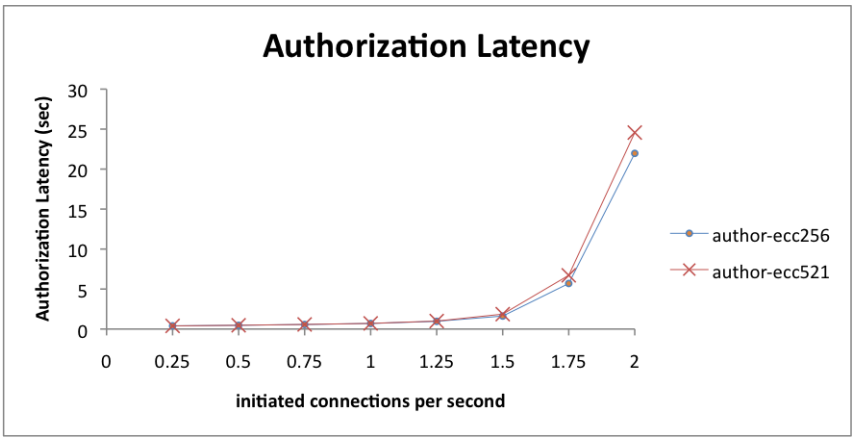

Figure 10.7: Experiment 2.2: Average Authorization Latency for Modified System

authorization has significant impact on performance. For the experiment using 256 bits certificates 20607 connections were successfully initiated in the allotted timeframe and 1 timed out. For the experiment using 521 bits certificates 20521 connections were successfully initiated in the allotted timeframe and 101 timed out. Upon further inspection of the test results we discovered that all these timeouts occurred at two initiated connections per second.

We intended to set out the results from this experiment against those of Section 10.2.3.1. Due to the significant impact on the performance we were forced to change the parameters for this experiment and rerun Experiment 2.1 to be able to compare the measurements in order to derive the authorization latency.

10.2.3.3 Experiment 2.3: Combined Authentication and Revocation Latency

In the prototype the certificate revocation status check takes place *during* the authorization and replaces part of strongSwan's code. Therefore, unlike Experiment 10.2.3.2, the impact of the certificate revocation checking latency alone cannot be determined by comparing the combined latency for authentication and revocation with the latency for authentication only obtained in Experiment 10.2.3.1.

Performance Measures The following performance measures apply for this subexperiment:

- Revocation latency as described in Section 10.2.1.3

Approach The server is configured to perform certificate revocation checking and not to perform authorization. It therefore performs authentication and certificate revocation checking without authorization. We measure this using the same approach as stated in Section 10.2.3.1. The differentiation of these measurements with the previous measurements of the authentication latency is used to determine the impact of the certificate revocation check.

For this experiment we use the same approach as stated in Section 10.2.2.

Results Of each load — i.e. number of initiated connection request per second on average, exponentially distributed — we calculate the average connection time. To this end we disregard any connections initiated in the first two minutes to allow the system to warm up. The connections that have been initiated but are not connected yet are disregarded too. This average is calculated over each of the ten reruns of this experiment. As these ten reruns have a different seed for the exponential distribution generation we can calculate a confidence interval over these averages, which are presented in Appendix D.2.2.

Figure 10.8 shows the average combined authentication and revocation latency results versus load (number of initiated connections per second) when the modified system is used. The curve labelled as authent-revocation-ecc256 shows the measured average values of the combined authentication + revocation check latency for the situation that the encryption algorithm uses 256 bits long certificates. The curve labelled as authent-revocation-ecc521 shows the measured average values of the of the composed Authentication + Revocation check latency for the situation that the encryption algorithm uses 521 bits long certificates.

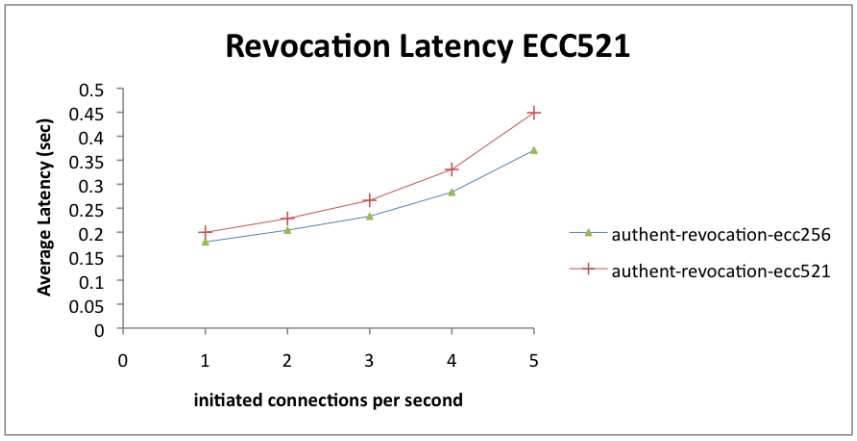

Figure 10.8: Experiment 2.3: Average Combined Authentication and Revocation Latency for Modified System

Figure 10.8 shows that at five initiated connections per second, the la-

tency does not increase as much as is the case in Experiment 1 and 2.1. This suggests that the improved certification checking actually decreases the connection latency.

In this experiment using 256 bits certificates 35166 connections were successfully initiated in the allotted timeframe and none timed out. Using 521 bits certificates 35159 connections were successfully initiated in the allotted timeframe and none timed out.

10.2.3.4 Experiment 2.4: Total latency

The total latency is determined by measuring whilst the prototype performs authentication, authorization and certificate revocation status checking.

Performance Measures The following performance measures apply for this subexperiment:

- Total latency as described in Section 10.2.1.3

Approach The server is configured to perform authorization and certificate revocation checking. It therefore performs authentication, authorization and certificate revocation checking. We measure this using the same approach as stated in Section 10.2.3.1. Also we perform the experiment for different key strengths. Again the WebDAV plug-in does not use SSL.

Results Of each load — i.e. number of initiated connection request per second on average, exponentially distributed — we calculate the average connection time. To this end we disregard any connections initiated in the first two minutes to allow the system to warm up. The connections that have been initiated but are not connected yet are disregarded too. This average is calculated over each of the ten reruns of this experiment. As these ten reruns have a different seed for the exponential distribution generation we can calculate a confidence interval over these averages, which are presented in Appendix D.2.4.

Figure 10.9 compares the average Total latency results versus load (number of initiated connections per second) when (1) the Basic (non-modified) and the Modified systems are used and (2) the encryption uses 256 and 521 bits long certificates.

Experiment 2.2 showed that the authorization has a significant increase on the latency. As this effect outweighs the performance benefit of the revocation checking improvement, we again had to adjust our testing parameters.

Figure 10.9 shows a significant increase in latency time starting at 1.5 connections per second. Meaning that the modification for improved authorization has significant impact on performance. In this experiment, using 256 bits certificates 20620, connections were successfully initiated in the allotted

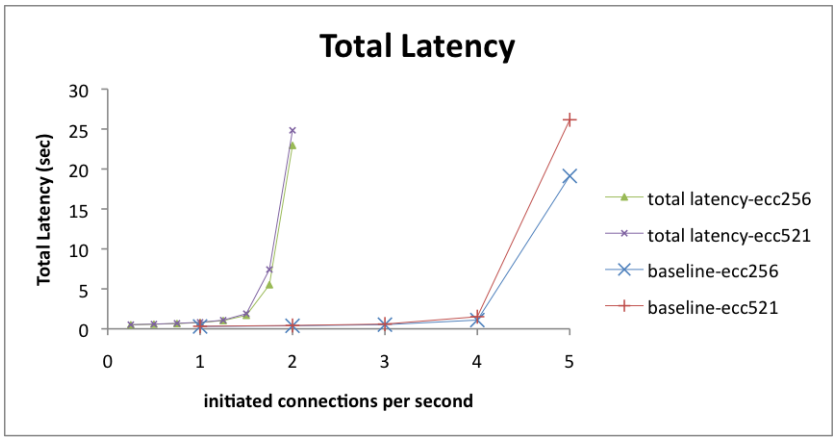

Figure 10.9: Experiment 2.4: Average Total Latency for Basic and Modified Systems

timeframe and 32 timed out. Using 521 bits certificates, 20555 connections were successfully initiated in the allotted timeframe and 117 timed out.

All the times out connections were at two initiated connections per second, when the system was overloaded.

10.2.3.5 Throughput

Throughput, or the amount of connections the system handles per second is, is another performance measure. At the performance ceiling of this measure, the point where more connections are initiated than can be handled by the system, the average latency increases as the strongSwan client retries connecting to the server. Therefore Figure 10.9 gives a strong indication on the performance ceiling — or if you will, maximum throughput — of the baseline and modified version. For the unmodified version the maximum throughput is somewhere between four and five connections per second. For the modified version the maximum throughput is between 1.5 and 1.75 connection per second. The previous sections have shown that the extra authorization is to blame for this significant decrease in performance as the revocation status checking actually improved the throughput.

10.2.4 Experiment 3: The Impact of Different Hardware

The goal of this experiment is to determine in which way the used hardware influences performance of the prototype. Like Experiment 2.4 this experiment only focuses on the total latency.

We intended to run both the WebDAV and PERMIS server on the WRT device. Due to time limitations and resource constraints that proved to strict for this, we opted to omit these.

10.3 Extendability

Other devices can make use of the PERMIS engine as long as they are able to connect to the reasoning engine and support querying it. The policies can be enriched to incorporate such devices. We opt for this approach over a proxy that evaluates actions. First and foremost because our approach allows for gradual improvement of fine-grained and context aware authorization. Secondly a proxy would require extra communications due of the intended actions, which is undesirable in a wireless environment.

10.4 New Applications

New applications that integrated in our framework need to be adapted in order to be able to enforce the policy. In its simplest form the gateway allows or disallows connectivity, providing no extra fine-grained authorization in the application. In the next maturity level the application needs to query the PERMIS engine for those actions that it cannot resolve itself. Even more mature would be an application that contains a (partial) PERMIS engine.

10.5 Summary

Our prototype showed that the modifications had impact on the performance of strongSwan. When turned off, these modifications had a slight adverse affect on performance compared to an unmodified version. The PERMIS modification proved to have a significant negative effect on the performance. The improved revocation checking had a significant positive effect on the performance. The negative effect on PERMIS outweighs that of the WebDAV-based revocation checking, though this is based on retrieval via HTTP rather than HTTPS.

At some experiments connections unexpectedly timed out. These were caused by the system being in an overloaded state.

Chapter 11

Conclusions and Further Work

We present our conclusions in regard to the research questions and present further work next.

11.1 Conclusions

Our main research goal of this thesis is *Specification, design and implementation of a scalable and refined access control architecture for a federation of personal networks that provides a high degree of security and privacy.* We have specified, designed and prototyped a security architecture that is, has a refined access control mechanism and is applicable in the field of Personal Networks, though is not scalable. To that end we first refined a suitable authentication protocol in Chapter 5 and have shown that the EAP-IKEv2 protocol is the best suitable authentication protocol to fit this context. For lack of an AAA implementation supporting EAP-IKEv2 we used the IKEv2 protocol without an AAA implementation in our prototype. We investigated possible ciphersuites and have shown that ECDH, ECDSA, AES and SHA-2 combine the best suitable ciphersuite in Chapter 6. We looked at suitable credential providers in Chapter 7 and recommend WebDAV. We looked at several policy languages in Chapter 8 and found that PERMIS is up to the task. We combined these into one architecture in Chapter 9 and derived a prototype. In Chapter 10 we evaluated this prototype and concluded that PERMIS has a drastic negative effect on performance that outweigh the benefits of the improved revocation checking.

In regard to the research question *Refine a suitable authentication protocol to work in the suggested framework*, we looked at eight EAP compatible protocols and based on 19 criteria we concluded that the EAP-IKEv2 protocol suits best. Noteworthy is that non of the protocols met one criteria: true server-initiated re-authentication.

In regard to the research question *Investigate and select suitable cipher*

suites to be used within the suggested framework, we stated requirements, compared relative key strengths and mentioned broken ciphers. Taking these security requirements and resource constrained devices into account , we found ECDH, ECDSA, AES and SHA-2 to be best suitable. These are in line with the NSA suite B recommendations.

In regard to the research question *Refine a credential provider to work within the suggested framework*, we looked at 11 ways to store and retrieve certificates. After first examination five of these were set out against six requirements that we set out. Of these five WebDAV proved best suitable.

In regard to the research question *Refine a suitable policy language to work in the suggested framework*, we found literature on this topic which led us to PERMIS.

In regard to the research question *Develop an experimental system (prototype) that demonstrates the basic principles*, we have used and adapted strongSwan to work in conjunction with PERMIS and perform the improved revocation status check. We have determined that the strongSwan can run on a resource constraint device and by during this process gave the strongSwan integration on the OpenWrt platform a boost.

In regard to *Study the scalability of the solution taking into account*, we analyzed performance of our prototype using different parameters. Authorization had a drastic negative impact on performance. The improved revocation checking mechanism had a positive impact on performance. We have successfully tested our prototype on a resource constrained device, proving that our framework can work in resource constrained environments. During which we discovered that CPU performance has significant impact on connection set up time. We found that strongSwan does not scale very well in our baseline. Note though, that we did not use the high availability mode of strongSwan. A newer version of strongSwan, tweaking options or using a different IKEv2 engine might increase performance. Also the used PERMIS solution did not scale well. As this is programmed in Java, a port might increase performance. Our solutions allows for gradual implementation of components and applications, allowing the maturity level to be incremental.

11.2 Further Work

We created a prototype to prove several concepts. A prototype by definition is not mature enough for daily needs, and therefore needs to be enhanced. We note the following

Retesting. Our test setup introduced unforeseen anomalies into our test results, i.e. connection failures. These issue need to be addressed after which the prototype should be retested. Adding experiments that using a wireless medium would be interesting to look at, as would experiments in a congested network.

Privacy policies. Privacy in this work is focused on identity privacy. However when the concept of FedNets becomes widespread with commercial applications, such as third party service providers, it becomes necessary to state the *purposes* of collecting data, the *intent of data requesters* and any additional enterprise customized data subjects' *constraints* [87].

Context awareness. The concept of FedNets allows for a context aware environment. To fully use the context awareness, access control needs to incorporate context – such as location – into its decision making [144]. Keeping privacy in mind [120, 119, 70]. [9] describes such a Location-Based Access Control (LBAC) using GSM/Third Generation (3G) technologies.

Security levels. We assumed only one security level. Literature, such as [93], suggests multiple levels of security. This, along with better context awareness should be added to the prototype.

EAP-IKEv2. For our prototype we used IKEv2 instead of EAP-IKEv2. To use EAP-IKEv2 as specified in our architecture strongSwan and RADIUS or Diameter need to be adapted to support EAP-IKEv2.

AAA. A EAP-IKEv2 supporting AAA implementation needs to be made that supports PERMIS for authorization.

strongSwan. Although strongSwan does support EAP to work with RADIUS, strongSwan needs to be adapted to facilitate EAP-IKEv2 in conjunction with either RADIUS or Diameter. strongSwan did not scale well in our experiments. strongSwan is being actively developed, using a newer version or fiddling with strongSwan's configuration parameters might provide better scalability.

Alternative IKEv2 engine As strongSwan did not scale well in our prototype. Wether other engines perform better in the context given in this thesis is unclear.

WebDAV. strongSwan pulls the certificate from the WebDAV server using plain HTTP. A method that has proved to be more faster than regular certificate checking. How retrieval using HTTPS affects performance has yet to be determined.
Our prototype does not provide a mechanism that creates the certificates and stores those on a WebDAV server as suggested in the architecture we proposed. Ideally, to prevent collisions on a highly populated WebDAV server, the referring URL should be in the form of `http://urlbase/c=<countrycode>/o=<orgname>/cn=<commonname>/serialNumber=<number>.extension`. This requires adaptation of the prototype.

PERMIS. In our prototype, the PERMIS engine uses the local filesystem to store its ACs. PERMIS is also capable of using LDAP and WebDAV for storing and retrieval of the ACs. PERMIS showed to have significant impact on the performance of the prototype. Optimization of PERMIS, such as compiling to a more efficient language then Java, should be investigated further before PERMIS can be considered inadequate for the task.

Performance optimization. Our prototype has been build as proof of

concept. Thus the code that we have written could very well be optimized for better performance.

Algorithms. Algorithms that are more lightweight and still secure could be integrated in our architecture as they arise and deemed proven in the field. More so when hardware-encryption-support in wireless portable devices becomes more widespread. [17] could be a possible candidate to replace AES in that case.

Android. Smartphones have become increasingly popular and powerful. There are even models available that have a dual core CPU running up to 1.5GHz. As strongSwan has been ported to Android, an OS for smartphones and the like, adapting our prototype to run on Android would bring it closer to the intended usage.

Appendix A

Acronyms

In this document the following acronyms were used:

2TDEA	Two key Triple DES
3DES	Tripple-DES a.k.a. TDEA
3G	Third Generation
3TDEA	Three key Triple-DES
AA	Attribute Authority
AAA	Authentication, Authorization and Accounting
AAAA	Authentication, Authorization, Accounting and Auditing
ABAC	Attribute Based Access Control
AC	Access Control
ACA	Access Control Architecture
ACL	Access Control List
ACM	Access Control and Management Framework
AES	Advanced Encryption Standard
AKA	Authentication and Key Agreement
AP	Access Point
API	Application Programming Interface
ARPANET	Advanced Research Projects Agency NETwork

AS	Authentication Server
ASM	Application Specific Module
ASN.1	Abstract Syntax Notation One
AVP	Attribute Value Pair
BER	Basic Encoding Rules
BSK	Body Sensor Kit
BSN	Body Sensor Network
CA	Certificate Authority
CASM	Context Aware Security Manager
CPU	Central Processing Unit
CHAP	Challenge-Handshake Authentication Protocol
CI	Confidence Interval
CIS	Credential Issuing Service
CRL	Certificate Revocation List
CRS	Certificate Revocation Status
CSCF	Serving Call Session Control Function
DAC	Discretionary Access Control
DACS	Design and Analysis of Communication Systems
DDR	Double Data Rate
DER	Distinguished Encoding Rules
DES	Data Encryption Standard
DH	Diffie Hellman
DNS	Domain Name System
DRM	Digital Rights Management
DSS	Digital Signature Standard
EAP	Extensible Authentication Protocol

EAP-AKA EAP-Method for 3rd Generation Authentication and Key Agreement

EAP-IKEv2 EAP-Internet Key Exchange Protocol version 2

EAP-FAST EAP-Flexible Authentication via Secure Tunneling

EAP-GTC EAP-(Generic Token Card

EAP-MSCHAP EAP-MicroSoft Challenge Handshake Authentication Protocol

EAP-PSK EAP-Pre-Shared Key Extensible Authentication Protocol

EAP-TLS EAP-Transport Layer Security

EAP-TTLS EAP-Tunneled Transport Layer Security

EC Elliptic Curve

ECC Elliptic Curve Cryptography

ECDH Elliptic Curve Diffie-Hellman

ECDSA Elliptic Curve Digital Signature Algorithm

EEMCS Electrical Engineering, Mathematics and Computer Science

EIST Enhanced Intel SpeedStep Technology

EMSK Extended Master Session Key

EPD Electronic Patient Dossier

FA FedNet Agent

FAP FedNet Access Policy

FedNet Federated Personal Network

FFC Finite Field Cryptography

FM FedNet Manager

FSB Front Side Bus

GNU GNU's Not Unix!

GSM Global Sytem for Mobile Communications

GUI Graphical User Interface

GW Gateway

HAVAL	One-Way Hashing Algorithm with Variable Length of Output
HDD	Hard Disk Drive
HN	Home Network
HPC	High Performance Computing
HLR	Home Location Register
HSS	The Home Subscriber Server
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
IEEE	Institute of Electrical & Electronics Engineers
IETF	Internet Engineering Task Force
IFC	Integer Factorization Cryptography
IKE	Internet Key Exchange
IMS	IP Multimedia Subsystem
IP	Internet Protocol
IPsec	IP security
ISIM	IP Multimedia Services Identity Module
ISM	Industrial, Scientific and Medical
ISP	Internet Service Provider
JDK	Java Development Kit
JRE	Java Runtime Environment
LAN	Local Area Network
LBAC	Location-Based Access Control
LDAP	Lightweight Directory Access Protocol
LQA	Link Quality Assessment
MAC	Mandatory Access Control
MAC	Medium Access Control

MAC	Message Authentication Code
MAGNET	My personal Adaptive Global NET
MAKE	Mutual Authentication with Key Exchange
MD4	Message Digest Algorithm 4
MD5	Message Digest Algorithm 5
MIPS	Microprocessor without Interlocked Pipeline Stages
MOBIKE	Mobility and Multihoming Protocol
MSCHAP	MicroSoft Challenge Handshake Authentication Protocol
MSK	Master Session Key
NAS	Network Access Server
NAT	Network Address Translation
NIC	Network Interface Card
NIST	National Institute of Standards and Technology
NSA	National Security Agency
OCSP	Online Certificate Status Protocol
OS	Operating System
P2P	Peer-to-Peer
PAN	Personal Area Network
PAP	Password Authentication Protocol
PDA	Personal Digital Assistant
PDP	Policy Decision Point
PEAP	Protected Extensible Authentication Protocol
PEP	Policy Enforcement Point
PIP	Policy Information Point
PERMIS	PrivilEge and Role Management Infrastructure Standards
PKC	Public Key Certificate
PKI	Public Key Infrastructure

PN	Personal Network
PN-F	Personal Network Federation
PNP2008	Personal Network Pilot 2008
PNCP	Personal Network Clustering Protocol
PNPA	PN Provisioning Administration
PNPP	PN Provisioning Party
POSIX	Portable Operating-System Interface
PPP	Point to Point Protocol
PRP	Policy Retrieval Point
PSK	Pre-Shared Key
PSP	PlayStation Portable
P-PAN	Private Personal Area Network
QoS	Quality of Service
RADIUS	Remote Authentication Dial In User Service
RAM	Random Access Memory
RBAC	Role Based Access Control
RC4	Ron's Code 4
RC5	Rivest Cipher 5
RIPEMD	Race Integrity Primitives Evaluation Message Digest
RISC	Reduced Instruction Set Computing
ROM	Read Only Memory
RSA	Ron Rivest, Adi Shamir and Leonard Adleman
SA	Security Association
SAML	Security Assertions Markup Language
SAP	Service Access Policy
SATA	Serial Advanced Technology Attachment
SCMF	Secure Context Management Framework

SCS	Service Context Service
SD	Service Discovery
SDRAM	Synchronous Dynamic Random Access Memory
SGT	Service Granting Ticket
SHA	Secure Hash Algorithm
SIM	Subscriber Identity Module
SIP	Session Initiation Protocol
SMN	Service Management Node
SOA	Source of Authority
SP	Service Proxy
SSH	Secure Shell
SSL	Secure Socket Layer
S-CSCF	Serving CSCF
TCP	Transmission Control Protocol
TDEA	Triple Data Encryption Algorithm a.k.a. 3DES
TGS	Ticket Granting Server
TGT	Ticket Granting Ticket
TLS	Transport Layer Security
TTLS	Tunneled Transport Layer Security
UA	User Agent
UDP	User Datagram Protocol
UE	User Equipment
UICC	Universal Integrated Circuit Card
UMTS	Universal Mobile Telecommunications System
URL	Uniform Resource Locator
USA	United States of America
USIM	Universal Subscriber Identity Module

UTP	Unshielded Twisted Pair
VITRUVIUS	Versatile Interface for TRUstworthy VItal User (oriented) Services
VO	Virtual Organization
VoIP	Voice over IP
VPN	Virtual Private Network
WAN	Wide Area Network
WebDAV	Web-based Distributed Authoring and Versioning
WEP	Wired Equivalent Privacy
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network
TI-WMC	Twente Institute for Wireless and Mobile Communications B.V.
WRT	Wireless Receiver/Transmitter
XACML	eXtensible Access Control Markup Language
XML	eXtensible Markup Language

Appendix B

Reproducing the Results

This chapter guides one trough the necessary steps to recreate our results.

Section B.1 states the steps needed to create a platform from which the results can be recreated. Section B.2 describes how to create a WRT image. In Section B.3 application installation Section B.4 relevant configuration files of the applications are given. In Section B.5 we give instructions on how to run the code.

B.1 Environment Setup

This section states the necessary steps we had to perform after installing a clean *Ubuntu 10.04* machine in order to get the software we need to compile and run the various tasks.

B.1.1 Ubuntu

We use Ubuntu as a platform for cross compiling. Also our test setup contains computers running on Ubuntu. Explanation on the installation of Ubuntu is beyond the scope of this document. Ubuntu has an active community that might prove helpful if any problems are encountered during base installation.

Note that we used Ubuntu 10.04.3 LTS. To upgrade to the latest distribution first run:

```
sudo apt-get update
sudo apt-get upgrade-distro
```

To update applications on the system:

```
sudo apt-get update
sudo apt-get upgrade
```

B.1.2 OpenWrt

This subsection explains the necessary steps to be able to crosscompile an image for a wireless router.

We need subversion to download OpenWrt with the OpenWrt build environment as we need to make changes to the default settings.

```
sudo apt-get install subversion
```

We create and subsequently enter a directory in which we are going to store and run the crosscompile platform. We use openwrt/trunk in the users home directory in this example.

```
mkdir ~/openwrt
cd ~/openwrt
mkdir trunk
cd trunk
```

Next we retrieve the platform:

```
svn co svn://svn.openwrt.org/openwrt/trunk/ .
```

Now we can run

```
make menuconfig
```

and the make script checks for its requirements and reports back if any are missing, in our case:

```
Build dependency: Please install the GNU C Compiler (gcc).
Build dependency: Please install the GNU C++ Compiler (g++).
Build dependency: Please install ncurses. (Missing libncurses.so
 or ncurses.h)
Build dependency: Please install zlib. (Missing libz.so or zlib.h)
Build dependency: Please install GNU awk.
Build dependency: Please install flex.
Build dependency: Please install unzip.
```

To install the missing software one can use to following commands:

```
sudo apt-get install gcc
sudo apt-get install g++
sudo apt-get install libncurses-dev
sudo apt-get install zlib1g-dev
sudo apt-get install gawk
sudo apt-get install flex
sudo apt-get install unzip
```

B.1.3 Java

Java is needed to run the PERMIS reasoning engine. A Java Runtime Environment (JRE) will suffice for this purpose. To compile self written java source code to java classes a Java Development Kit (JDK) is needed instead.

To install the JRE on Gentoo run:

```
sudo apt-get install gcj-jre
```

To install the JDK on Gentoo run:

```
sudo apt-get install sun-java6-jdk
```

Note that the JRE jamvm that can be used to run java on OpenWrt does not support the sun java classes, use the GNU Compiler for java instead:

```
sudo apt-get install gcj-jdk
```

B.2 Creating the OpenWrtImage

In S ection B.1 we have prepared a system for crosscompiling. In this section we state the necessary steps and configurations to recreate our crosscompiled wireless router image.

We enter the the crosscompile directory and check for any updates:

```
cd ~/openwrt/trunk
svn co svn://svn.openwrt.org/openwrt/trunk/ .
```

For reproducing our build, use

```
svn co -r 28611 svn://svn.openwrt.org/openwrt/trunk/ .'
```

instead. Now we have the latest crosscompiling platform ready to use. However these do not include the most recent changed packages created for this platform. To update these we run the command:

```
scripts/feeds update
```

Next we install several packages we need:

```
scripts/feeds install openssl
scripts/feeds install ntpclient
scripts/feeds install libcurl
scripts/feeds install lighttpd-mod-webdav
scripts/feeds install libpcre
```

We have created our own strongSwan package. Although our work has been integrated and is available as a package via OpenWrt's platform, this might be different than our version. To recreate our results use the package provided in Appendix F.1.

We create a symbolic link in the crosscompile platform. This makes the package available for the platform:

```
cd ~/openwrt/trunk/package
ln -s ./../../package_strongswan-4.5.3-beus
```

To minimize its footprint the openssl package provided does not include Elliptic Curve Cryptography (ECC). To enable ECC we need to configure the *openssl* Makefile by removing the 'no-ec' option. The following commands will do this, however will break the Makefile if the option is not encountered:

```
cd ~/openwrt/trunk/package/openssl
vi Makefile
/no-ec
3dw
:wq
```

We need to configure the makefile using for the cross compilation. This is done using a Graphical User Interface (GUI) provided with the platform. To start the GUI run:

```
cd ~/openwrt/trunk
make menuconfig
```

Select the correct Target System, we used 'Broadcom BCM947xx/953xx'. Be sure not to select a 2.4 variant as our prototype depends on a 2.6 kernel.

Next Select Image configuration and set 'LAN IP Address' to a value that works for you, e.g. 192.168.2.1. Go back to the main menu. Enter Network and in that, VPN, enter 'strongswan 4' and select strongswan4-beus. This will automatically select several needed options.

To create the actual WRT image run:

```
cd ~/openwrt/trunk
make
```

This will take some time. More so the first time, when it has compile the entire build environment. As indication, we advice lunch over a cup of coffee at this point...

B.3 Application Installation

This section installation of the used components.

B.3.1 OpenWrt

To install the created WRT image we on the wireless router the following steps need to be performed.

The needed image depends on wether the device is running the original stock software or a custom firmware such as we intend to install now. The ' /trunk/bin/brcm47xx/' directory contains several images. The image to use depends on wether the router is running stock or non-stock firmware. The only .trx file can be used when the router is already running OpenWrt. When the router is still running stock the .bin file corresponding to the routers make and model needs to be used.

There are several ways to flash the new image to the router. The first and most simple method is to use the web-interface of the router and perform a firmware upgrade.

The second method is less straightforward. This usually does not work by default on stock ROM, however it does by default on OpenWrt images. Note the username you are currently working with and the IP address of the machine you build the image. Open a terminal connection to the router using either telnet or SSH. If a password had not yet been set you can only login using telnet. If a password has been set, login as root using a SSH client.

Using this terminal Copy the image to the router:

```
scp USERNAME@WORKSTATION_IP:/home/USERNAME/trunk/bin/brcm47xx/\
openwrt-brcm47xx-squashfs.trx /tmp
```

Flash the router using the image:

```
mtd -r write /tmp/openwrt-brcm47xx-squashfs.trx linux
```

If mtd has not been installed you can do so using the commands:

```
opkg update
opkg install mtd
```

The router will reboot automatically. SSH might not work afterwards. This can be resolved by using telnet and setting a password.

To make the compiled packages available to the router copy the /home-/USER/trunk/bin/brcm47xx/packages directory to a HTTP server accessible by the router. E.g. http://192.168.1.134/packages

We again open a terminal session to the router. Next we run the following commands to install several packages:

```
mkdir /etc/opkg
echo "src my-pkgs http://192.168.1.134/packages" >> /etc/opkg.conf
opkg update
opkg install ntpclient
ntpclient -h 0.openwrt.pool.ntp.org -p 123 -s
```

Depending on which packages to install on the router the following commands are neccesarry:

```
opkg install strongswan
```

B.3.2 WebDAV

How to install WebDAV on unix systems is widely documented online, e.g. [129]. This section explains how to install WebDAV on OpenWrt.

To make sure that the latest version is available we update the repository:

```
opkg update
```

Next lighttpd and the WebDAV module gets installed by running the command:

```
opkg install lighttpd-mod-webdav
```

B.3.3 Java

In Appendix B.1.3 we explained how to install Java on Ubuntu. This section explains how to install Java on OpenWrt.

To load the available packages and to check for any updates we update the repository:

```
opkg update
```

Next we install the lightweight Java engine jamvm:

```
opkg install jamvm
```

We used jamvm version 1.5.4. Note that this is based on GNU classpath Java class library and Java 1.5.

Jamvm depends on the classpath java library. Our wireless router proved to resource constrained to be able to store this. As workarounds for this problem would defeat our purpose of running java on this device. Therefor we choose not to pursue this further and run java on a regular system in our tests instead. [71] provides useful steps to trim down the classpath base print, effectively cutting down on (unnecessary) functionality, if one is to attempt to run jamvm on an OpenWrt device.

B.3.4 strongSwan

During generation of the router image strongSwan is included. However strongSwan can run on any Unix system. This subsection explains how to compile strongSwan for such systems.

First retrieve the tarball from `http://download.strongswan.org`, e.g. based on version 4.5.3:

```
wget http://download.strongswan.org/strongswan-4.5.3.tar.bz2
```

Next extract the tarball:

```
tar -xvvf strongswan-4.5.3.tar.bz2
```

Enter the directory gerenerated during this process:

```
cd strongswan-4.5.3
```

At this point patches can be applied to make any necessary changes before compilation. For example, to apply the changes of our prototype on the *4.5.3* build of strongSwan run the following command:

```
patch -p1 -i /PATH/TO/205-strongswan-conf-4.5.3-mod-beus-v04.patch
```

Note that this patch is likely incompatible with other strongSwan versions and break the functionality of that version!

Configure the options. For all the options consult [125]. For experiment one we used the following options:

```
./configure --enable-openssl
```

For experiment two we used the following options:

```
./configure --build=i486-linux-gnu --disable-nls\
 --disable-ipv6 --with-random-device=/dev/random\
 --with-urandom-device=/dev/urandom --enable\
-curl --disable-aes --disable-des --disable-md5 \
--disable-sha1 --disable-sha2 --disable-fips-prf \
--disable-gmp --disable-pubkey --disable-pluto \
--disable-tools --enable-openssl --with-routing-\
table-prio=220 --with-routing-table=220 --disable-static
```

Compile the source with these options by running the command:

```
make
```

If any dependencies are missing make will report this. Installing the package(s) as in Section B.1 will solve this. In addition we needed the following:

```
sudo apt-get install libcurl4-openssl-dev
```

After make successfully compiles we can install strongSwan using the following command:

```
sudo make install
```

Removing the package from the system is performed likewise, simply run from the same directory:

```
sudo make uninstall
```

B.4 Configuration

This section details configurations for the used applications.

B.4.1 strongSwan

The configuration of strongSwan itself is stored in /etc/strongswan.conf on OpenWrt and /usr/local/etc/strongswan.conf on Ubuntu. This file contains global strongSwan options only, the individual connections are stored in a different configuration file.

On OpenWrt we configure strongSwan to perform authorization and certificate revocation:

Listing B.1: strongswan.conf

```
# /etc/strongswan.conf - strongSwan configuration file

charon {
  reuse_ikesa = no
  threads = 64
  block_threshold = 100
  cookie_threshold = 100
  half_open_timeout = 165

  plugins {
    stroke {
        max_concurrent = 16
    }
    permis {
      enable = TRUE
      servername = 192.168.1.134
      port = 5010
    }
    webdav {
        enable  = TRUE
        timeout = 5
    }
  }
}
```

The load command specifies which plugins strongSwan should start. Uncommenting this will cause strongSwan to try to load all plugins.

Change the servername value to the machine on which the local PERMIS server is running. Note that this config file is very strict on white space usage and that any unmodified version of PERMIS will ignore these settings.

On the initiating side, one would check for authorization to connect to another PN before attempting the connection. Therefore our prototype only checks for WebDAV authorization when being the responding entity. Although it would make sense for the initiating entity to check the responders certificate revocation status, we omit this so that compatibility with a normal strongSwan installation can be determined. We configure the initiator side of strongSwan not to perform authorization and revocation checking:

Listing B.2: stronfswan.conf

```
# /etc/strongswan.conf - strongSwan configuration file

charon {
  reuse_ikesa = no
  threads = 64
  block_threshold = 100
  cookie_threshold = 100
  half_open_timeout = 165

  plugins {
    stroke {
        max_concurrent = 16
    }
    permis {
      enable = FALSE
      servername = 192.168.1.134
      port = 5010
    }
    webdav {
        enable  = FALSE
        timeout = 5
    }
  }
}
```

For more configuration options see [124].

B.4.2 HTTPD/Apache/WebDAV

We did not use WebDAV functionality in our prototype. For retrieval, an out-of-the-box HTTP server will suffice.

B.4.3 PERMIS

We provide the policy we created using the GUI provided with PERMIS.

Listing B.3: policy.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<X.509_PMI_RBAC_Policy OID="test01">
    <SubjectPolicy>
        <SubjectDomainSpec ID="Anywhere">
            <Include LDAPDN=""/>
        </SubjectDomainSpec>
    </SubjectPolicy>
    <RoleHierarchyPolicy>
        <RoleSpec OID="1.2.826.0.1.3344810.1.1.14" Type="permisRole">
            <SupRole Value="a">
                <SubRole Value="b"/>
            </SupRole>
            <SupRole Value="b">
                <SubRole Value="c"/>
            </SupRole>
            <SupRole Value="c"/>
        </RoleSpec>
    </RoleHierarchyPolicy>
    <SOAPolicy>
        <SOASpec ID="Anyone" LDAPDN=""/>
```

```
</SOAPolicy>
<RoleAssignmentPolicy>
    <RoleAssignment ID="RoleAssignment1">
        <SubjectDomain ID="Anywhere"/>
        <RoleList>
            <Role Type="permisRole"/>
        </RoleList>
        <Delegate Depth="0"/>
        <SOA ID="Anyone"/>
        <Validity/>
    </RoleAssignment>
</RoleAssignmentPolicy>
<TargetPolicy>
    <TargetDomainSpec ID="http://www.test.com/test01">
        <Include URL="http://www.test.com/test01"/>
    </TargetDomainSpec>
    <TargetDomainSpec ID="http://www.test.com/test02">
        <Include URL="http://www.test.com/test02"/>
    </TargetDomainSpec>
    <TargetDomainSpec ID="http://www.test.com/test03">
        <Include URL="http://www.test.com/test03"/>
    </TargetDomainSpec>
</TargetPolicy>
<ActionPolicy>
    <Action Name="OPTIONS" ID="OPTIONS"/>
    <Action Name="GET" ID="GET"/>
    <Action Name="HEAD" ID="HEAD"/>
    <Action Name="POST" ID="POST"/>
    <Action Name="PUT" ID="PUT"/>
    <Action Name="DELETE" ID="DELETE"/>
    <Action Name="TRACE" ID="TRACE"/>
    <Action Name="CONNECT" ID="CONNECT"/>
</ActionPolicy>
<TargetAccessPolicy>
    <TargetAccess ID="TargetAccess1">
        <RoleList>
            <Role Type="permisRole" Value="b"/>
        </RoleList>
        <TargetList>
            <Target>
                <TargetDomain ID="http://www.test.com/test01"/>
                <AllowedAction ID="CONNECT"/>
            </Target>
        </TargetList>
        <IF>
            <AND>
                <GT>
                    <Environment Parameter="time" Type="Time"/>
                    <Constant Type="Time" Value="*-*-*T12:00:00"/>
                </GT>
            </AND>
        </IF>
    </TargetAccess>
    <TargetAccess ID="TargetAccess2">
        <RoleList>
            <Role Type="permisRole" Value="c"/>
        </RoleList>
        <TargetList>
            <Target>
                <TargetDomain ID="http://www.test.com/test01"/>
                <AllowedAction ID="CONNECT"/>
            </Target>
```

```
            </TargetList>
            <IF>
                <AND>
                    <LT>
                        <Environment Parameter="time" Type="Time"/>
                        <Constant Type="Time" Value="*-*-*T12:00:00"/>
                    </LT>
                </AND>
            </IF>
        </TargetAccess>
        <TargetAccess ID="TargetAccess3">
            <RoleList>
                <Role Type="permisRole" Value="a"/>
            </RoleList>
            <TargetList>
                <Target>
                    <TargetDomain ID="http://www.test.com/test01"/>
                    <AllowedAction ID="CONNECT"/>
                </Target>
            </TargetList>
            <IF>
                <AND>
                    <GT>
                        <Environment Parameter="time" Type="Time"/>
                        <Constant Type="Time" Value="*-*-*T12:00:00"/>
                    </GT>
                </AND>
            </IF>
        </TargetAccess>
    </TargetAccessPolicy>
</X.509_PMI_RBAC_Policy>
```

B.5 Running the Code

This section explains how to start the components of our prototype.

B.5.1 OpenWrt

We explained how to create a OpenWrt image file in Section B.2, how to install this image in and strongSwan in Section B.3 and how to configure strongSwan in Section B.4. Before we start strongSwan we need to perform one more step.

By default the OpenWrt software implements a firewall that blocks all incoming traffic on the WAN connection. The IKE protocol implemented in strongSwan makes use of UDP port 500 and 4500 and these need to be let trough in order for strongSwan to work properly. We open the SSH port also, allowing our test script to access the router remotely.

```
echo "
# allow ssh from wan, needed for experiment.
config rule
        option src              wan
        option dest_port        22
```

```
        option target           ACCEPT
        option proto            tcp

# allow IPsec/ESP and ISAKMP
config rule
        option src              wan
        option protocol         esp
        option target           ACCEPT

# allow ike
config rule
        option src              wan
        option dest_port        500
        option proto            udp
        option target           ACCEPT

">> /etc/config/firewall

/etc/init.d/firewall restart
```

Next we can start, as user root, strongSwan by running the command:

```
ipsec start
```

strongSwan is now ready to be used on this host.

B.5.2 Java

Assuming a JRE is set up correctly and PERMIS has attribute certificates along with the policy in place, we can start the PERMIS server.

To start the PERMIS server run the following command:

```
java -cp .:pba5_0_1.jar:include/\* PermisServer
```

To run the Java PERMIS server on the OpenWrt machine run it with the command:

```
jamvm -cp .:pba5_0_1.jar:include/\* PermisServer
```

Note that this will not work if classpath is not installed properly on the OpenWrt machine.

It might be necessary to compile the server java class using a different JDK. To compile the PermisServer from Java source run the following on a Ubuntu machine set up as described in Section B.1.3:

```
javac -cp pba5_0_1.jar PermisServer.java
```

Depending on the used JDK the command may differ.

B.5.3 Experiments

For the exponential generator, one might first need to run the following command:

```
sudo apt-get install gsl-bin
```

We used the script given in Listings B.4, B.5 and B.6.

Listing B.4: performScriptSet.script

```
#!/bin/bash

if [[ $# -ne 1 ]]
then
        echo "Usage: ‘basename $0‘ {arg}"
        echo "where arg is the integer value of the test to perform"
        exit
fi

#general settings
testID=$1
seconds=200
confidence_times=10
forStrength="256 521"
clientIP="192.168.1.134"
serverIP="192.168.1.135"

client=A
server=B
webdavHost=althor
clientHost=althor
permisHost=althor
prefix=/usr/local

if [[ $testID == 1 ]] #baseline pc-pc
then
        subTests=1
        load=8
        denominator=1
        serverHost=beus
        serverEth=3
        count=1000 # number of certificates
elif [[ $testID == 2 ]] # 1: baseline with mod, 2: permis, no webdav, 3:
    no permis, webdav, 4: permis and webdav
then
        subTests="1 2 3 4"
        load=10
        denominator=1
        serverHost=beus
        serverEth=3
        count=1000 # number of certificates
elif [[ $testID == 3 ]] # wrt device
then
        subTests="1 2 3 4"
        load=10
        denominator=2
        serverHost=wrt
        serverEth=0
else
        echo "test $testID not found, quitting.."
```

```
        exit
fi


echo client:
ssh root@$clientHost "uname -a; ps aux; ifconfig; ethtool eth0"
echo server:
ssh root@$serverHost "uname -a; ps aux; ifconfig; ethtool eth${serverEth}"


for subtest in $subTests
do

  #exception to the rule
  #if [[ $testID == 3 ]]
  #then
    #withouth patch 10 seconds is too short for wrt device to load all
        certs
    # src/starter/invokecharon.c
    #seconds=8
    #load=4
    #if [[ $subtest == 2 ]] || [[ $subtest == 4 ]]
    #then
    #  load=4
    #fi
  #fi
  if [[ ($testID == 2) || ($testID == 3) ]]
  then
        if [[ ($subtest == 2) || ($subtest == 4) ]] #account for permis
            delay
        then
                denominator=$(( denominator * 2 ))
        fi
  fi

  for strength in $forStrength
  do
        echo "#####  Performing  set-up  for  test  '${testID}',  for  strength  '
            ${strength}'.  #####"
        #clear local conf
        ssh root@$clientHost "rm  ${prefix}/etc/ipsec.d/cacerts/*  ;rm  ${
            prefix}/etc/ipsec.d/certs/*  ;rm  ${prefix}/etc/ipsec.d/private
            /*;rm  ${prefix}/etc/ipsec.conf;rm  ${prefix}/etc/ipsec.secrets;
            rm  ${prefix}/etc/strongswan.conf"
        #clear remote conf
        ssh root@$serverHost "rm  ${prefix}/etc/ipsec.d/cacerts/*  ;rm  ${
            prefix}/etc/ipsec.d/certs/*  ;rm  ${prefix}/etc/ipsec.d/private
            /*;rm  ${prefix}/etc/ipsec.conf;rm  ${prefix}/etc/ipsec.secrets;
            rm  ${prefix}/etc/strongswan.conf"
        #clear webdav cache
        if [ $webdavHost = wrt ]
        then
                ssh root@$webdavHost "rm  /www/webdav/C\=NL/O\=WMC/*"
        else
                ssh root@$webdavHost "rm  /var/www/webdav/C\=NL/O\=WMC/*"
        fi

        #copy certs and keys, wget is not secure, but faster than scp for
            large numbers.
        echo "WGETing  certs  and  keys  for  host  ${client}..."
        ssh root@$clientHost "rm  wget.in;  for  i  in  \$(seq  1  1  $count);  do
            echo  http://althor/tmp/host${client}_${strength}-\${i}_Key.pem
```

```
    ␣>>␣wget.in;␣done␣;␣wget␣--input-file=wget.in␣--directory-
    prefix=${prefix}/etc/ipsec.d/private␣-q"
ssh root@$clientHost "rm␣wget.in;␣for␣i␣in␣\$(seq␣1␣1␣$count);␣do␣
    echo␣http://althor/tmp/host${client}_SIGNED_BY_${server}
    _Cert_${strength}-\${i}.der␣>>␣wget.in;␣done␣;␣wget␣--input-
    file=wget.in␣--directory-prefix=${prefix}/etc/ipsec.d/certs␣-q
    "
echo "WGETing␣certs␣and␣keys␣for␣host␣${server}..."
ssh root@$serverHost "rm␣wget.in;␣for␣i␣in␣\$(seq␣1␣1␣$count);␣do␣
    echo␣http://althor/tmp/host${server}_${strength}-\${i}_Key.pem
    ␣>>␣wget.in;␣done;␣wget␣--input-file=wget.in␣--directory-
    prefix=${prefix}/etc/ipsec.d/private␣-q"
  ssh root@$serverHost "rm␣wget.in;␣for␣i␣in␣\$(seq␣1␣1␣$count);␣
      do␣echo␣http://althor/tmp/host${server}_SIGNED_BY_${client}
      _Cert_${strength}-\${i}.der␣>>␣wget.in;␣done␣;␣wget␣--input-
      file=wget.in␣--directory-prefix=${prefix}/etc/ipsec.d/certs␣
      -q"

#copy certs to webdav store
echo "copying␣certs␣into␣WebDAV␣store␣on␣host:␣${webdavHost}..."
if [ $webdavHost = wrt ]
then  # store @ althor
        ssh root@$webdavHost "for␣y␣in␣${strength};do␣for␣i␣in␣\$(
            seq␣1␣1␣$count);␣do␣wget␣http://althor/tmp/host${
            client}_SIGNED_BY_${server}_Cert_\${y}-\${i}.der␣-O␣/
            www/webdav/C\=NL/O\=WMC/CN\=host${client}\${y}-\${i}␣-
            q␣;done;done"
else    # wrt device uses different base dir
        ssh root@$webdavHost "for␣y␣in␣${strength};do␣for␣i␣in␣\$(
            seq␣1␣1␣$count);␣do␣wget␣http://althor/tmp/host${
            client}_SIGNED_BY_${server}_Cert_\${y}-\${i}.der␣-O␣/
            var/www/webdav/C\=NL/O\=WMC/CN\=host${client}\${y}-\${
            i}␣-q␣;done;done"
fi

echo "installing␣root␣CA␣for␣client␣(host:␣${client})..."
scp root@althor:/temp/host${client}_CA_${strength}_Key.pem root@${
    clientHost}:${prefix}/etc/ipsec.d/private
scp root@althor:/temp/host${client}_CA_${strength}_Cert.der root@$
    {clientHost}:${prefix}/etc/ipsec.d/cacerts
echo "installing␣root␣CA␣for␣server␣(host:␣${server})..."
scp root@althor:/temp/host${server}_CA_${strength}_Key.pem root@${
    serverHost}:${prefix}/etc/ipsec.d/private
scp root@althor:/temp/host${server}_CA_${strength}_Cert.der root@$
    {serverHost}:${prefix}/etc/ipsec.d/cacerts

echo "generate␣and␣install␣client␣keyfile..."
./genIpsecSecrets.useArgs.script $client $strength $count > ipsec.
    secrets
scp ./ipsec.secrets root@${clientHost}:${prefix}/etc/ipsec.secrets
cp ./ipsec.secrets ./log/ipsec.secrets.${testID}\-${client}${
    strength}

echo "generate␣and␣install␣client␣ipsec.conf..."
./genIpsecConf.useArgs.script $client $strength $count $clientIP
    $serverIP > ipsec.conf
scp ./ipsec.conf root@${clientHost}:${prefix}/etc/ipsec.conf
cp ./ipsec.conf ./log/ipsec.conf.${testID}\-${client}${strength}

echo "generate␣and␣install␣server␣keyfile..."
./genIpsecSecrets.useArgs.script $server $strength $count > ipsec.
    secrets
```

```
        scp ./ipsec.secrets root@${serverHost}:${prefix}/etc/ipsec.secrets
        cp ./ipsec.secrets ./log/ipsec.secrets.${testID}\-${server}${
            strength}

        echo "generate␣and␣install␣server␣ipsec.conf..."
        ./genIpsecConf.useArgs.script $server $strength $count $clientIP
            $serverIP > ipsec.conf
        scp ./ipsec.conf root@${serverHost}:${prefix}/etc/ipsec.conf
        cp ./ipsec.conf ./log/ipsec.conf.${testID}\-${server}${strength}

        echo "installing␣client␣and␣server␣strongswan.conf..."
        scp ./strongswan.conf.$testID.$subtest root@${serverHost}:${prefix
            }/etc/strongswan.conf
        scp ./strongswan.conf.$testID.$subtest root@${clientHost}:${prefix
            }/etc/strongswan.conf

        logfile=./log/results.exp${testID}.sub$subtest.$strength
        #do the test
        echo "#####␣Running␣script␣test␣'${testID}',␣for␣strength␣'${
            strength}',␣using␣${count}␣certs/keys.␣#####"

        # 1st arg: keystrength, i.e. 256, 384, 521
        # 2nd arg: perform how many times, i.e. confidence level
        # 3rd arg: seconds to run
        # 4th arg: max load (connections per second)
        # 5th arg: divider (actual tested load = 1/divider, 2/divder,...
            load)
        # 6th arg: testID
        # 7th arg: peer_host

echo  time ./exp.useArgs.script $strength $confidence_times $seconds $load
     $denominator exp${testID}.${subtest} $serverHost $webdavHost
    $permisHost;

        { time ./exp.useArgs.script $strength $confidence_times $seconds
            $load $denominator exp${testID}.${subtest} $serverHost
            $webdavHost $permisHost; } > $logfile 2>&1
  done
done
```

Listing B.5: exp.useArgs.script

```
#!/bin/bash

# 1st arg: keystrength, i.e. 256, 384, 521
# 2nd arg: perform how many times, i.e. confidence level
# 3rd arg: seconds to run
# 4th arg: max load (connections per second)
# 5th arg: denominator (actual tested load = 1/denominator, 2/denominator
    ,... load)
# 6th arg: testID
# 7th arg: peer_host
# 8th arg: webdavHost
# 9th arg: permisHost

confidence_times=$2
seconds_to_run=$3
denominator=$5
max_load=$[$4 * $denominator]
id=$6.
peer_host=$7
webdav_host=$8
```

```
permis_host=$9
base_conn_name="ecc"$1

sleepSecs=30

maxcount=1000
if [[ testID:0:4 == exp3 ]]  #router
then
 maxcount=100
 sleepSecs=60
fi

echo "This␣script␣requires␣password␣free␣access␣to␣root␣on"
echo "both␣the␣host␣and␣client␣machine␣to␣restart␣services."
echo "Be␣sure␣to␣have␣keys␣in␣place␣to␣allow␣for␣this."
echo
echo "This␣script␣expects␣configuration␣and␣certificates␣to␣be"
echo "in␣place.␣Parameters␣can␣be␣set␣by␣editing␣this␣script."
echo
echo "Running␣script␣with␣the␣following␣parameters:"
echo "-␣Iterate␣$confidence_times␣times␣for␣confidence␣interval."
echo "-␣Run␣each␣test␣for␣$seconds_to_run␣seconds."
echo "-␣Increase␣the␣load␣up␣to␣$max_load␣connections␣per␣second."
echo "-␣Remote␣peer␣host␣FQDN/ip␣=␣$peer_host."
echo "-␣certificate␣revocation␣host␣(webdav)␣=␣$webdav_host."
echo "-␣Strongswan␣is␣configured␣for␣connections␣$base_conn_name-#."
echo

# --- initiate array of certs/federations --- #
for (( i=1; i <= maxcount; i++ ))
do
        connection[i]=$base_conn_name-$i
done

#running the actual test
for ((s=1; s <= confidence_times ; s++))
do
        echo
        echo "#####␣confidence␣run:␣$s␣#####"
        echo maxload=$max_load
        for (( load=1 ; load <= max_load ; load+=1 ))
        do
                actualLoad=$(echo "scale=8;$load/$denominator" | bc)

                echo "###␣testing␣with␣load:␣$actualLoad␣###"

                echo "restarting␣java␣reasoning␣engine␣Permis,␣local␣
                    strongSwan,␣remote␣strongSwan␣and␣remote␣Apache␣
                    servers␣(in␣background)..."
                ssh -n -f root@${permis_host} "sh␣-c␣\"killall␣java;␣sleep
                    ␣1;␣cd␣/home/beus/permis/sw4.5.3-10;␣nohup␣java␣-cp␣.:
                    pba5_0_1.jar:include/\*␣PermisServer␣>␣/dev/null␣2>&1␣
                    &\""

                echo "#␣Restarting␣local␣strongSwan␣deamon...␣"
                ssh root@localhost "ipsec␣stop;␣sleep␣3␣;␣killall␣-9␣
                    starter;␣killall␣-9␣charon;␣sleep␣1␣;␣rm␣-f␣/var/run/
                    starter.pid␣/var/run/charon.pid␣;␣ipsec␣start"


                echo "#␣Restarting␣remote␣strongSwan␣deamon...␣"
```

```
ssh root@$peer_host "ipsec␣stop;␣sleep␣3␣;␣killall␣-9␣
    starter;␣killall␣-9␣charon;␣sleep␣1␣;␣rm␣-f␣/var/run/
    starter.pid␣/var/run/charon.pid␣;␣ipsec␣start"


echo "#␣Restarting␣webdav␣apache␣server...␣"
if [ $webdav_host = wrt ]
then
        ssh root@$webdav_host "/etc/init.d/lighttpd␣
            restart"
else
        ssh root@$webdav_host "/etc/init.d/apache2␣restart
            "
fi


#echo -n "(Re)shuffeling␣for␣randomness..."
#note: see shuf man page on how to reproduce the results
    of an earlier invocation
#connection=( $(echo ${connection[@]} | tr "␣" "\n" | shuf
    ) )
#echo "␣done."

echo -n "flushing␣caches."
ssh root@localhost "sync;␣echo␣3␣>␣/proc/sys/vm/
    drop_caches"
echo -n .
ssh root@$webdav_host "sync;␣echo␣3␣>␣/proc/sys/vm/
    drop_caches"
echo -n .
ssh root@$peer_host "sync;␣echo␣3␣>␣/proc/sys/vm/
    drop_caches"
echo "␣done."

echo -n "jolting␣permis␣in␣background..."
ssh -n -f root@$permis_host "cd␣/home/beus/permis/;␣java␣
    TestServer␣\"C=NL,␣O=WMC,␣CN=hostA256-999\"␣localhost;
    ␣java␣TestServer␣\"CN=Jan␣Willem␣Beusink,OU=student,O=
    WMC,C=NL\"␣localhost␣>␣/dev/null"
#give ipsec some time to load all the certs and keys
echo "started␣in␣background"
echo "Waiting␣$sleepSecs␣seconds␣to␣allow␣for␣strongswan␣
    to␣read␣the␣keys␣and␣certificates."
sleep $sleepSecs
echo "done␣sleeping"

mu=$(echo "scale=8;1/$actualLoad" | bc )
i=0
echo "Starting␣launch␣of␣connections"
timeOffsetBase=$(date +"%s%N")
for interval in $(gsl-randist $s $(echo "scale=8;$load*
    $seconds_to_run/$denominator"| bc) exponential $mu)
do
        echo "sleeping:␣$interval"
        sleep $interval
        #launch initiation in background yet time it.
        ./timeJob $id $s $actualLoad $i ipsec up ${
            connection[$(( i%maxcount +1 ))]}
            $timeOffsetBase &
        i=$(expr $i + 1)
done
```

```
                wait    #waiting for connections setups that run in
                    background to be done...
        done
done

echo
echo --- DONE ---
exit 0
```

Listing B.6: timeJob

```
#!/bin/bash
#param1 = parent script name
#param2 = confidence run
#param3 = load (connections/second)
#param4 = the n'th-1 started job in this run
#param5 = command
#param6 = command arg 1
#param7 = command arg 2 (expected to be the identifier)
#param8 = time in 'date "%s%N"' format when the very fist job started, i.e
    . when this load cycle was initiated.
#param9 = background transfer speed kbit/s
echo "performing: $5 $6 $7"
a=$(date +"%s%N")
time $5 $6 $7 2>&1 > log/$1conf_run$2-load$3-conn$4-$7
b=$(date +"%s%N")
c=`expr $b - $a`

#determine success status
found=$(ipsec status | grep $7{ | wc -l)
active=$(ps aux | grep timeJob | wc -l)
charonMem=$(ps aux | grep charon | grep use-syslog | awk '{print $4}')
#bring it down
time $5 down $7 2>&1 >> log/$1conf_run$2-load$3-conn$4-$7

startedAfterNanosec=`expr $a - $8`

echo -e "script:$1\trun:$2\tload(conn/sec):$3\tjob #$4\tidentifier:$7\
    tstarted after\t$startedAfterNanosec\tjob took:\t$c\tnanosec\tfound: 
    $found\tactive:\t$active\tcharonMem% $charonMem"
```

Appendix C

Used hardware

This Appendix states the hardware used in our experiments. Table C.1 lists the specifications of these machines.

Table C.1: Hardware used

	Wireless router	Host A	Host B
CPU model	Broadcom BCM4712 BCM3300 v0.7	Intel Core i7-860	AMD Athlon 64 3200+
CPU architecture	MIPS32	Intel 64 (x86-64)	AMD 64 (x86-64)
CPU cores	1	4	1
CPU threads	1	8	1
CPU speed	200 MHz	2.8 GHz 3.46 GHz (Turbo) 1.2 GHz (EIST)	2.2 GHz
CPU Bogomips	197.63	5596/core	4423
RAM type	N/A	DDR3-1333 SDRAM	DDR1-400 SDRAM
RAM model	N/A N/A	2x Corsair XMS3 TW3X4G1333C9	PQI DDR 400
RAM size	32 MB	8 GB	512 MB
RAM clock speed	N/A	666.2 MHz (DDR)	201 MHz (DDR)
RAM configuration	N/A	Dual channel 4 banks occupied	Single channel 1 bank occupied
Mainboard	Linksys WRT54GS v1.1	Gigabyte GA-P55M-UD4 BIOS: F11	MSI K8N Neo3-F (MS-7135 ver. 1) BIOS: V1.7
FSB speed	unknown	133 MHz	200 MHz
Storage type	Flash memory	SATA HDD	SATA HDD
Storage model	Intel TE28F640	Maxtor 6V320F0	Samsung HD642JJ
Storage quantity	8 MB	320 GB	640 GB
Network chip	ADMtek 6996L	RTL8111D NICs	nVidia CK804 rev a3
Network speed	100 Mbit	1 Gbit	1 Gbit
OS	OpenWrt	Kubuntu	Ubuntu Server Ed.
kernel version	3.0.3	2.6.32-37-generic	2.6.32-37-server
OS version	trunk, rev 28611 packages: rev 28713	10.04	10.04

Appendix D

Confidence Intervals

This appendix presents confidence intervals corresponding to the means presented in Chapter 10.

These are based on 10 reruns using different seeds. Each run lasted 6 minutes of which the first 2 are disregarded. Connection attempts that timed out after 47 seconds or had not finished at the time we stopped initiating new requests are also disregarded in the calculations.

D.1 Experiment 1: Average Total Latency for Baseline (non-modified) system

This section presents the confidence intervals corresponding with the means presented in Subection 10.2.2.

Table D.1: Experiment 1.1: Confidence Intervals of Total Latency when Using ECC256 Certificates.

Load	sample mean $\bar{x}$	$c\frac{s}{\sqrt{n}}$	$\bar{x} - c\frac{s}{\sqrt{n}}$	$\bar{x} + c\frac{s}{\sqrt{n}}$
1.00000000	0.294	0.0050	0.288	0.299
2.00000000	0.378	0.012	0.366	0.391
3.00000000	0.522	0.034	0.487	0.556
4.00000000	1.095	0.136	0.959	1.231
5.00000000	19.125	4.04	15.085	23.165

0 of 33741 connections during measurement window timed out.

Table D.2: Experiment 1.1: Confidence Intervals of Total Latency when Using ECC521 Certificates.

Load	sample mean $\bar{x}$	$c\frac{s}{\sqrt{n}}$	$\bar{x} - c\frac{s}{\sqrt{n}}$	$\bar{x} + c\frac{s}{\sqrt{n}}$
1.00000000	0.318	0.0060	0.312	0.324
2.00000000	0.419	0.016	0.403	0.435
3.00000000	0.595	0.042	0.554	0.637
4.00000000	1.517	0.325	1.192	1.843
5.00000000	26.162	3.379	22.783	29.54

232 of 33158 connections during measurement window timed out.

D.2 Experiment 2: Authentication, Authorization and Certificate Revocation of Modified System

This section presents the confidence intervals corresponding with the means presented in Subection 10.2.3.

D.2.1 Experiment 2.1: Authentication Latency

Table D.3: Experiment 2.1: Average Authentication Latency for Modified System when Using ECC256 Certificates.

Load	sample mean $\bar{x}$	$c\frac{s}{\sqrt{n}}$	$\bar{x} - c\frac{s}{\sqrt{n}}$	$\bar{x} + c\frac{s}{\sqrt{n}}$
1.00000000	0.306	0.0050	0.301	0.311
2.00000000	0.398	0.013	0.385	0.412
3.00000000	0.56	0.039	0.522	0.599
4.00000000	1.355	0.248	1.108	1.603
5.00000000	23.626	3.965	19.662	27.591

142 of 33368 connections during measurement window timed out.

Table D.4: Experiment 2.1: Average Authentication Latency for Modified System when Using ECC521 Certificates.

Load	sample mean $\bar{x}$	$c\frac{s}{\sqrt{n}}$	$\bar{x} - c\frac{s}{\sqrt{n}}$	$\bar{x} + c\frac{s}{\sqrt{n}}$
1.00000000	0.336	0.0060	0.33	0.343
2.00000000	0.445	0.017	0.428	0.462
3.00000000	0.642	0.047	0.595	0.689
4.00000000	2.083	0.899	1.184	2.982
Continued on Next Page...				

Table D.4: (continued)

Load	sample mean $\bar{x}$	$c\frac{s}{\sqrt{n}}$	$\bar{x} - c\frac{s}{\sqrt{n}}$	$\bar{x} + c\frac{s}{\sqrt{n}}$
5.00000000	29.609	2.748	26.862	32.357

671 of 32819 connections during measurement window timed out.

The experiment was rerun using smaller increments in load for comparison with other experiments. These results with confidence intervals are given in table D.5 and D.6

Table D.5: Experiment 2.1: Average Authentication Latency for Modified System when Using ECC256 Certificates and Using Smaller Increments in Load.

Load	sample mean $\bar{x}$	$c\frac{s}{\sqrt{n}}$	$\bar{x} - c\frac{s}{\sqrt{n}}$	$\bar{x} + c\frac{s}{\sqrt{n}}$
.25000000	0.194	0.0040	0.19	0.197
.50000000	0.198	0.0040	0.194	0.203
.75000000	0.204	0.0040	0.2	0.208
1.00000000	0.21	0.0030	0.206	0.213
1.25000000	0.219	0.0030	0.216	0.222
1.50000000	0.229	0.0040	0.224	0.233
1.75000000	0.239	0.0050	0.235	0.244
2.00000000	0.25	0.0050	0.245	0.256

0 of 21019 connections during measurement window timed out.

Table D.6: Experiment 2.1: Average Authentication Latency for Modified System when Using ECC521 Certificates and Using Smaller Increments in Load.

Load	sample mean $\bar{x}$	$c\frac{s}{\sqrt{n}}$	$\bar{x} - c\frac{s}{\sqrt{n}}$	$\bar{x} + c\frac{s}{\sqrt{n}}$
.25000000	0.213	0.0030	0.21	0.215
.50000000	0.22	0.0050	0.215	0.224
.75000000	0.227	0.0030	0.224	0.231
1.00000000	0.233	0.0030	0.229	0.236
1.25000000	0.244	0.0040	0.24	0.248
1.50000000	0.257	0.0060	0.251	0.262
1.75000000	0.267	0.0070	0.261	0.274
2.00000000	0.282	0.0080	0.274	0.29

0 of 21016 connections during measurement window timed out.

Noteworthy is the difference between the values of Table D.3 and D.5, and between Table D.4 and D.6. We expected a slight diffence due to the

randomness of the intervals, however not in the order of tens of procents. We were unable to determine the cause, however note that the turbo property of the CPU might be to blame.

D.2.2 Experiment 2.2: Authorization Latency

Table D.7: Experiment 2.2: Average Authorization Latency for Modified System Using ECC256 Certificates.

Load	sample mean $\bar{x}$	$c\frac{s}{\sqrt{n}}$	$\bar{x} - c\frac{s}{\sqrt{n}}$	$\bar{x} + c\frac{s}{\sqrt{n}}$
.25000000	0.591	0.02	0.571	0.612
.50000000	0.667	0.033	0.634	0.7
.75000000	0.773	0.057	0.715	0.83
1.00000000	0.907	0.105	0.801	1.012
1.25000000	1.183	0.071	1.112	1.253
1.50000000	1.851	0.291	1.56	2.142
1.75000000	5.935	3.876	2.059	9.811
2.00000000	22.216	4.896	17.32	27.113

1 of 20607 connections during measurement window timed out.

Table D.8: Experiment 2.2: Average Authorization Latency for Modified System Using ECC521 Certificates.

Load	sample mean $\bar{x}$	$c\frac{s}{\sqrt{n}}$	$\bar{x} - c\frac{s}{\sqrt{n}}$	$\bar{x} + c\frac{s}{\sqrt{n}}$
.25000000	0.608	0.021	0.587	0.628
.50000000	0.69	0.036	0.654	0.727
.75000000	0.802	0.068	0.734	0.87
1.00000000	0.938	0.107	0.832	1.045
1.25000000	1.241	0.07	1.171	1.311
1.50000000	2.12	0.613	1.507	2.732
1.75000000	6.984	3.907	3.077	10.892
2.00000000	24.847	4.687	20.161	29.534

1010 of 20521 connections during measurement window timed out.

D.2.3 Experiment 2.3: Certificate Revocation Check Latency

Table D.9: Experiment 2.3: Average Revocation Check Latency for Modified System using ECC256 certificates.

Load	sample mean $\bar{x}$	$c\frac{s}{\sqrt{n}}$	$\bar{x} - c\frac{s}{\sqrt{n}}$	$\bar{x} + c\frac{s}{\sqrt{n}}$
1.00000000	0.18	0.0030	0.177	0.182
2.00000000	0.204	0.0040	0.2	0.208
3.00000000	0.233	0.0070	0.226	0.24
4.00000000	0.283	0.012	0.272	0.295
5.00000000	0.371	0.026	0.345	0.397

0 of 35166 connections during measurement window timed out.

Table D.10: Experiment 2.3: Average Combined Authentication and Revocation Latency for Modified System Using ECC521 Certificates.

Load	sample mean $\bar{x}$	$c\frac{s}{\sqrt{n}}$	$\bar{x} - c\frac{s}{\sqrt{n}}$	$\bar{x} + c\frac{s}{\sqrt{n}}$
1.00000000	0.2	0.0040	0.195	0.204
2.00000000	0.228	0.0040	0.225	0.232
3.00000000	0.267	0.0080	0.258	0.275
4.00000000	0.331	0.014	0.317	0.345
5.00000000	0.449	0.035	0.414	0.484

0 of 35149 connections during measurement window timed out.

Table D.11: Experiment 2.3: Average Combined Authentication and Revocation Latency for Modified System Using ECC256 Certificates and Using Smaller Increments in Load.

Load	sample mean $\bar{x}$	$c\frac{s}{\sqrt{n}}$	$\bar{x} - c\frac{s}{\sqrt{n}}$	$\bar{x} + c\frac{s}{\sqrt{n}}$
.25000000	0.167	0.0030	0.164	0.17
.50000000	0.171	0.0050	0.166	0.175
.75000000	0.175	0.0040	0.171	0.179
1.00000000	0.178	0.0030	0.176	0.181
1.25000000	0.183	0.0020	0.181	0.185
1.50000000	0.19	0.0030	0.187	0.193
1.75000000	0.195	0.0030	0.192	0.198
2.00000000	0.2	0.0030	0.198	0.203

0 of 21021 connections during measurement window timed out.

Table D.12: Experiment 2.3: Average Combined Authentication and Revocation Latency for Modified System Using ECC521 Certificates and Using Smaller Increments in Load.

Load	sample mean $\bar{x}$	$c\frac{s}{\sqrt{n}}$	$\bar{x} - c\frac{s}{\sqrt{n}}$	$\bar{x} + c\frac{s}{\sqrt{n}}$
.25000000	0.186	0.0030	0.183	0.189
.50000000	0.192	0.0040	0.188	0.195
.75000000	0.196	0.0020	0.193	0.198
1.00000000	0.2	0.0040	0.196	0.204
1.25000000	0.208	0.0030	0.205	0.211
1.50000000	0.215	0.0040	0.211	0.218
1.75000000	0.223	0.0050	0.218	0.228
2.00000000	0.23	0.0040	0.226	0.234

0 of 21020 connections during measurement window timed out.

Unlike Experiment 2.1 the results presented in Table D.9 and D.11, and Table D.10 and D.12 are within the expected range of deviation due to the used randomness of the experiments.

D.2.4 Experiment 2.4: Total Latency

Table D.13: Experiment 2.4: Average Total Latency for Modified System Using ECC256 Certificates.

Load	sample mean $\bar{x}$	$c\frac{s}{\sqrt{n}}$	$\bar{x} - c\frac{s}{\sqrt{n}}$	$\bar{x} + c\frac{s}{\sqrt{n}}$
.25000000	0.504	0.02	0.483	0.524
.50000000	0.574	0.034	0.54	0.608
.75000000	0.658	0.054	0.605	0.712
1.00000000	0.782	0.103	0.678	0.885
1.25000000	1.032	0.059	0.973	1.091
1.50000000	1.685	0.346	1.339	2.031
1.75000000	5.51	3.751	1.76	9.261
2.00000000	22.949	5.072	17.876	28.021

32 of 20620 connections during measurement window timed out.

Table D.14: Experiment 2.4: Average Total Latency for Modified System Using ECC521 Certificates.

Load	sample mean $\bar{x}$	$c\frac{s}{\sqrt{n}}$	$\bar{x} - c\frac{s}{\sqrt{n}}$	$\bar{x} + c\frac{s}{\sqrt{n}}$
.25000000	0.52	0.021	0.499	0.541
Continued on Next Page...				

Table D.14: (continued)

Load	sample mean $\bar{x}$	$c\frac{s}{\sqrt{n}}$	$\bar{x} - c\frac{s}{\sqrt{n}}$	$\bar{x} + c\frac{s}{\sqrt{n}}$
.50000000	0.59	0.033	0.557	0.623
.75000000	0.688	0.06	0.628	0.748
1.00000000	0.811	0.104	0.707	0.915
1.25000000	1.079	0.068	1.012	1.147
1.50000000	1.894	0.452	1.443	2.346
1.75000000	7.423	4.268	3.155	11.692
2.00000000	24.84	4.817	20.024	29.657

117 of 20555 connections during measurement window timed out.

D.3 Experiment 3: The Impact of Different Hardware

This section presents the confidence intervals corresponding with the means presented in Subsection 10.2.4.

Omitted due to reasons stated in Subsection 10.2.4

Appendix E

Java Code

This appendix contains the java code we have written for the experiments. This includes vital functionality of the prototype as well as tools automating the setup and analisys of the experiments.

E.1 Policy Files

The PERMIS engine attribute certificates in conjunction with policies. Open PERMIS supplies a GUI to generate such attribute certificates and a policy editor to generate such policies. As we needed to use large numbers of such attribute certificate we wrote a java program to facilitate command line generation of these.

Note that this java file relies on severall packages that require a licence. These are available free of charge for educational purposes.

Listing E.1: SignAttributeCertifice.java

```
/*
* Copyright (c) 2006-7, University of Kent
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are
    met:
*
* Redistributions of source code must retain the above copyright notice,
    this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice
    ,
* this list of conditions and the following disclaimer in the
    documentation
* and/or other materials provided with the distribution.
*
* 1. Neither the name of the University of Kent nor the names of its
* contributors may be used to endorse or promote products derived from
    this
```

```
* software without specific prior written permission.
*
* 2. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "
    AS
* IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO
    ,
* THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
* PURPOSE ARE DISCLAIMED.
*
* 3. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
    THE
* POSSIBILITY OF SUCH DAMAGE.
*
* 4. YOU AGREE THAT THE EXCLUSIONS IN PARAGRAPHS 2 AND 3 ABOVE ARE
    REASONABLE
* IN THE CIRCUMSTANCES.  IN PARTICULAR, YOU ACKNOWLEDGE (1) THAT THIS
* SOFTWARE HAS BEEN MADE AVAILABLE TO YOU FREE OF CHARGE, (2) THAT THIS
* SOFTWARE IS NOT "PRODUCT" QUALITY, BUT HAS BEEN PRODUCED BY A RESEARCH
* GROUP WHO DESIRE TO MAKE THIS SOFTWARE FREELY AVAILABLE TO PEOPLE WHO
    WISH
* TO USE IT, AND (3) THAT BECAUSE THIS SOFTWARE IS NOT OF "PRODUCT"
    QUALITY
* IT IS INEVITABLE THAT THERE WILL BE BUGS AND ERRORS, AND POSSIBLY MORE
* SERIOUS FAULTS, IN THIS SOFTWARE.
*
* 5. This license is governed, except to the extent that local laws
* necessarily apply, by the laws of England and Wales.
*/

import iaik.asn1.BIT_STRING;
import iaik.asn1.CodingException;
import iaik.asn1.DerCoder;
import iaik.asn1.ObjectID;
import iaik.asn1.structures.AlgorithmID;
import iaik.asn1.structures.Name;
import iaik.utils.RFC2253NameParser;
import iaik.utils.RFC2253NameParserException;
import iaik.x509.X509Certificate;

import issrg.ac.ACCreationException;
import issrg.ac.AttCertIssuer;
import issrg.ac.AttCertValidityPeriod;
import issrg.ac.AttCertVersion;
import issrg.ac.Attribute;
import issrg.ac.AttributeCertificate;
import issrg.ac.AttributeCertificateInfo;
import issrg.ac.AttributeValue;
import issrg.ac.Extension;
import issrg.ac.Extensions;
import issrg.ac.Generalized_Time;
import issrg.ac.Holder;
import issrg.ac.IssuerSerial;
import issrg.ac.ObjectDigestInfo;
import issrg.ac.Util;
import issrg.ac.V2Form;
import issrg.ac.attributes.PermisRole;
```

```
import issrg.ac.extensions.AttributeAuthorityInformationAccess;
import issrg.ac.extensions.AuthorityInformationAccess;
import issrg.ac.extensions.NoRevocation;
import issrg.acm.IllegalInputException;
import issrg.acm.SigningUtility;
import issrg.security.DefaultSecurity;
import issrg.security.PKCS12Security;
import issrg.security.SecurityException;
import issrg.utils.CreateAAIALocation;

import java.io.*;
import java.math.BigInteger;
import java.util.*;

import javax.security.auth.login.LoginException;



public class SignAttributeCertificate {

        /**
         * Simplified and stripped command line utility variant of the
             Permis Attribute Certifate Manager.
         * This will create a singned attribute certificate containing a
             permisRole
         *
         * @param args[0] The pkcs12 file to sign the Attribute
             Certificate
         * @param args[1] The password of the pkcs12 file
         * @param args[2] The subjects DN.
         * @param args[3] Save AC as "".ace.
         * @param args[4] = serialNumber
         * @param args[5] = permisRole values separated by a comma
         */
    public static void main( String args[] ){
        //debug
        if (args.length == 1 && "test".equalsIgnoreCase(args[0])){
            args = new String[6];
            args[0]="/Users/jbeusink/Documents/workspace/afstudeerproject/
                include/acm_5_0_1/permis.p12";
            args[1]="l3tM3InNow";
            args[2]="CN=hostB123,O=WMC,C=NL";
            args[3]="testAceFile";
            args[4]="12345";
            args[5]="a,c";
        }
        if (args.length != 6 ){
              System.out.print("Usage: java -cp .:/acm.jar:./include/\\* "
                  SignAttributeCertificate " +
                  "signUsingFile.p12 p12Password subjectDN saveAsBaseName
                      serialnumber\n\n"+
                  "This class depends on the Permis acm library (which
                      uses the IAIK-jce library).\n"+
                  "Be sure to have downloaded these with the appropriate
                      licenses.");
              System.exit(0);
        }

        new SignAttributeCertificate(args);
        System.exit(0);
    }
```

```
/**
 * Class to create a simple attribute certificate
* @param args[0] The pkcs12 file to sign the Attribute Certificate
* @param args[1] The password of the pkcs12 file
* @param args[2] The subjects DN.
* @param args[3] Save AC as "".ace.
* @param args[4] = serialNumber
* @param args[5] = permisRole values separated by a comma

*/
public SignAttributeCertificate(String args[]){
    // Initialize
    vars = new Hashtable();
    vars.put("AC.Version", AttCertVersion.V2.toString());
    // Holder Name
    vars.put("AC.Holder.Name", args[2]);
    // Serial number
    vars.put("AC.SerialNumber", args[4]);
    // Save to:
    String saveTo = (args[3].endsWith(".ace") ? args[3] : args[3] + ".
        ace");
    vars.put("AC.Save.FileName", saveTo);

    //Validity Period From - To -- save as string.
    Date validFrom = new Date(); // i.e. nowcalendar.getTime();
    calendar = new GregorianCalendar(TimeZone.getTimeZone("GMT"));
    calendar.setTime(validFrom);
    calendarFrom = (GregorianCalendar)calendar.clone();
    calendar.add(Calendar.YEAR, 1);
    calendarTo = (GregorianCalendar)calendar.clone();
    Date validTo = calendar.getTime();

    String vt = "";
    calendar.setTime(validFrom);
    vt = Util.timeToString(calendar);
    vt = (new StringBuilder(String.valueOf(vt))).append(";").toString
        ();
    calendar.setTime(validTo);
    vt = (new StringBuilder(String.valueOf(vt))).append(calendar).
        toString();
    vars.put("AC.ValidityPeriod", vt);
    //\\ end validity period

    //Gui options:
    vars.put("Build␣Version␣2␣AC", true);
    jIssuerNameCheckBox = true;
    jIssuerBCIDCheckBox = false;
    chkWebdav = false;

    // set attribute(s) ?stripped createASN1 call:
    Vector attributes = new Vector();

    PermisRole role = new PermisRole(args[5]);

    /*  Class PrivilegeEditor
     *   public abstract String getOID();
     *   public abstract String getName();
     *   public abstract ASN1Type run(Frame frame, Map map, ASN1Type
         asn1type, Registry registry)
     *       throws ACCreationException;
     */
    //Instead of above; quick and dirty hack, hard coded OID:
```

```
    try {
      /* This would be the place to parse args[5] into separate roles
          if more than one was being provided
      * for example using a comma separated string. e.g.*/
        String[] roles = args[5].split(",");
        for (int x = 0; x < roles.length; x++){
            attributes.add(new Attribute("1.2.826.0.1.3344810.1.1.14",
                 new AttributeValue((new PermisRole(roles[x])).
                toASN1Object())));
        }
    } catch (CodingException e1) {
        System.err.println("error adding attribute, failed to convert
            to ASN1Object");
        e1.printStackTrace();

    }

    vars.put("AC.Attributes", attributes);
    // set extensions
    //... none

    // Issuer
    vars.put("DefaultSecurity.LastFile",args[0]);
    char[][] pws = new char[1][args[1].length()];
    for (int i = 0 ; i < args[1].length();i++){
        pws[0][i] = args[1].charAt(i);
    }
    pws[0] = args[1].toCharArray();
    vars.put("Passwords",pws);


    //generate, sign and save:
    try{
        save(generateAC());
    }
    catch(ACCreationException acce)
    {
        System.out.println("Errors saving: " + acce.getMessage());
    } catch (SecurityException e) {
        // Auto-generated catch block
        e.printStackTrace();
    } catch (LoginException e) {
        // Login in to signingUtility (PKCS12Security) failed.
        e.printStackTrace();
        System.err.println( "Failed to log in: " + e.getMessage().
            toString());

    } catch (CodingException e) {
        // Auto-generated catch block
        e.printStackTrace();
    }

}


protected byte[] generateAC()
    throws ACCreationException, SecurityException, LoginException,
        CodingException
{

    boolean v2wanted;
    BigInteger ACSerialNumber;
```

```
AttCertValidityPeriod validity_period;
Vector attributes;
Holder holder;
RFC2253NameParser.register("SERIALNUMBER", ObjectID.serialNumber);
boolean v2allowed = true;
Object o;
v2wanted = (o = vars.get("Build␣Version␣2␣AC")) != null ? ((
    Boolean)o).booleanValue() : false;
try
{
    ACSerialNumber = new BigInteger((String)vars.get("AC.
        SerialNumber"), 16);
}
catch(NumberFormatException nfe)
{
    try
    {
        ACSerialNumber = new BigInteger((String)vars.get("AC.
            SerialNumber"));
    }
    catch(NumberFormatException e)
    {
        throw new IllegalInputException("AC␣Serial␣Number␣must␣be␣
            a␣valid␣Integer␣value");
    }
}

ACSerialNumber = ACSerialNumber.abs();
if((String)vars.get("AC.Holder.Name") == "" && vars.get("AC.Holder
    .ObjectDigestInfo") == null)
    throw new IllegalInputException("Please␣specify␣one␣of␣the␣
        optional␣parameters␣for␣the␣Holder");
if(!jIssuerNameCheckBox && !jIssuerBCIDCheckBox && v2allowed)
    throw new IllegalInputException("Please␣select␣one␣of␣the␣
        optional␣parameters␣for␣the␣Issuer␣for␣inclusion");
if(!jIssuerNameCheckBox && !v2allowed)
    throw new IllegalInputException("Please␣select␣the␣name␣for␣
        the␣Issuer␣for␣inclusion:\nVersion␣1␣AC␣is␣enforced");

validity_period = new AttCertValidityPeriod(new Generalized_Time(
    calendarFrom), new Generalized_Time(calendarTo));

attributes = collapseAttributes((Vector)vars.get("AC.Attributes"))
    ;
if(attributes.size() == 0)
    throw new IllegalInputException("Attribute␣set␣cannot␣be␣empty
        ");

v2wanted = true;
if(v2wanted && !v2allowed)
    throw new IllegalInputException("Please␣explicitly␣unselect␣
        Version␣2␣features␣or␣enable␣creating␣V2␣ACs");
iaik.asn1.structures.GeneralNames hn = Util.buildGeneralNames((
    String)vars.get("AC.Holder.Name"));
holder = new Holder(null, hn, getHolderDigestInfo(v2wanted));

AlgorithmID signatureAlg;
AttributeCertificateInfo aci;
DefaultSecurity signingUtility;
signingUtility = new PKCS12Security();
signingUtility.setDigestAlgorithm(DefaultSecurity.DIGEST_ALGORITHM
    ); // ='SHA1'
```

```
signingUtility.login( (String)vars.get("DefaultSecurity.LastFile")
    , ((char[][])vars.get("Passwords"))[0]);
java.security.cert.X509Certificate signerPKC = signingUtility.
    getVerificationCertificate();

String subjectDN;
String issuerDN;
if(signerPKC instanceof X509Certificate)
{
    try
    {
        subjectDN = ((Name)signerPKC.getSubjectDN()).
            getRFC2253String();
        issuerDN = ((Name)signerPKC.getIssuerDN()).
            getRFC2253String();
    }
    catch(RFC2253NameParserException rnpe)
    {
        throw new ACCreationException("Failed␣to␣decode␣DNs", rnpe
            );
    }
} else
{
    subjectDN = signerPKC.getSubjectDN().getName();
    issuerDN = signerPKC.getIssuerDN().getName();
}
V2Form signer = new V2Form(Util.buildGeneralNames(subjectDN), new
    IssuerSerial(Util.buildGeneralNames(issuerDN), signerPKC.
    getSerialNumber(), null), null);
Vector ext = (Vector)vars.get("AC.Extensions");
if(ext == null)
    ext = new Vector();
Extensions extensions;
if(NOREV)
{
    System.out.println("NoRevocation␣extension␣being␣created");
    String location = (String)vars.get("LDAPSavingUtility.
        ProviderURI");
    NoRevocation norev = new NoRevocation(location, subjectDN,
        ACSerialNumber);
    ext.add(norev);
    extensions = new Extensions(ext);
} else
{
    if(chkWebdav)
    {
        System.out.println("Adding␣WebDAV␣revocation␣extension");
        AuthorityInformationAccess aia = new
            AuthorityInformationAccess(vars, subjectDN, ((String)
            vars.get("AC.Holder.Name")).intern(), ACSerialNumber);
        ext.add(aia);
        extensions = new Extensions(ext);
    } else
    {
        System.out.println("Just␣AAIA␣extension");
        if(ext != null && !ext.isEmpty())
        {
            for(Iterator i = ext.iterator(); i.hasNext();)
            {
                Extension e = (Extension)i.next();
                if(e instanceof
                    AttributeAuthorityInformationAccess)
```

```
                    {
                        i.remove();
                        break;
                    }
                }

            }
            String lFlag = (String)vars.get("AAIA_extension");
            if(lFlag != null && !lFlag.equals(""))
            {
                String location = (String)vars.get("LDAPSavingUtility.
                    ProviderURI");
                String aaiaLocation = CreateAAIALocation.
                    createLocation(location, subjectDN);
                if(aaiaLocation != null)
                {
                    AttributeAuthorityInformationAccess e = new
                        AttributeAuthorityInformationAccess(new String
                        [] {
                        aaiaLocation
                    });
                    if(ext == null)
                        ext = new Vector();
                    ext.add(e);
                }
            }
        }
        if(ext == null)
            ext = new Vector();
        extensions = new Extensions(ext);
    }
    if(!jIssuerNameCheckBox)
        signer.setIssuerName(null);
    if(!jIssuerBCIDCheckBox)
        signer.setBaseCertificateID(null);
    signer.setObjectDigestInfo(null);
    AttCertIssuer issuer = new AttCertIssuer(v2wanted ? null : signer.
        getIssuerName(), v2wanted ? signer : null);
    byte bt[] = signerPKC.getSigAlgParams();
    iaik.asn1.ASN1Object algParams = bt != null ? DerCoder.decode(bt)
        : null;
    signatureAlg = new AlgorithmID(new ObjectID(signingUtility.
        getSigningAlgorithmID()));
    aci = new AttributeCertificateInfo(new AttCertVersion(v2wanted ?
        AttCertVersion.V2 : AttCertVersion.DEFAULT), holder, issuer,
        signatureAlg, ACSerialNumber, validity_period, attributes,
        null, extensions);
    byte certificate[];
    //System.out.println("attributeCertificateInfo: "+aci.toString());
    byte b[] = aci.getEncoded();
    certificate = (new AttributeCertificate(aci, signatureAlg, new
        BIT_STRING(signingUtility.sign(b)))).getEncoded();
    signingUtility.logout();
    return certificate;
}



private void save(byte ac[]) // issr.acm.DiskSavingUtil derivate.
   throws ACCreationException
{
```

```
    try
    {
        AttributeCertificate thisAC = new AttributeCertificate(
            DerCoder.decode(ac));
        try
        {
            //file name to pathName in canonical form?
                String pathname = (String)vars.get("AC.Save.FileName")
                    ; // custom added var

                //change the following if statement to get normal
                    filename.
                if (true){
                    iaik.asn1.structures.GeneralNames gn;
                    gn = thisAC.getACInfo().getHolder().getEntityName
                        ();
                    byte hb[] = Util.hashName(gn);
                    String hash = Util.hashToString(hb);
                    int i = pathname.lastIndexOf(".");
                    pathname = (new StringBuilder(String.valueOf(
                        pathname.substring(0, i)))).append(".").append
                        (hash).append(pathname.substring(i)).toString
                        ();
                }

                OutputStream os = new FileOutputStream(pathname);
                os.write(ac);
                os.close();
        }
        catch(FileNotFoundException fnfe)
        {
            throw new ACCreationException(fnfe.getMessage(), fnfe);
        }
        catch(IOException ioe)
        {
            throw new ACCreationException((new StringBuilder("IO␣
                Exception:␣")).append(ioe.getMessage()).toString(),
                ioe);
        }
    }
    catch(CodingException ce)
    {
        throw new ACCreationException("Error␣occurred␣while␣decoding␣
            the␣AC", ce);
    }
}

protected ObjectDigestInfo getHolderDigestInfo(boolean v2)
{
    if(!v2)
        return null;
    else
        return (ObjectDigestInfo)vars.get("AC.Holder.ObjectDigestInfo"
            );
}

public Vector collapseAttributes(Vector src)
{
    if(src == null)
        src = new Vector();
    Object o[] = src.toArray();
    Vector result = new Vector();
```

```
        for(int i = 0; i < o.length; i++)
        {
            String a = ((Attribute)o[i]).getType();
            Vector v = ((Attribute)o[i]).getValues();
            for(int j = i; j-- > 0;)
                if(o[j] != null && ((Attribute)o[j]).getType().equals(a))
                {
                    ((Attribute)o[j]).getValues().addAll(v);
                    o[i] = null;
                    break;
                }

            if(o[i] != null)
                result.add(o[i]);
        }

        return result;
    }

    private String createRoleValuesString(String type, Vector values)
    {
        StringBuffer roleValuesString = new StringBuffer();
        for(int i = 0; i < values.size(); i++)
        {
            Attribute attr = (Attribute)values.get(i);
            if(attr.getType().equals(type))
            {
                Vector v = attr.getValues();
                if(v.size() > 0)
                {
                    for(int k = 0; k < v.size(); k++)
                        try
                        {
                            Object o = v.get(k);
                            PermisRole av = null;
                            if(o instanceof PermisRole)
                                av = new PermisRole((PermisRole)o);
                            else
                                av = new PermisRole((AttributeValue)o);
                            if(!av.getRoleValue().startsWith("<?xml"))
                                roleValuesString.append(av.getRoleValue())
                                    .append("+");
                        }
                        catch(Exception e)
                        {
                            return null;
                        }

                    if(roleValuesString.length() > 0)
                        roleValuesString.deleteCharAt(roleValuesString.
                            length() - 1);
                }
            }
        }

        return roleValuesString.toString();
    }

    private String createRoleValuesString(Vector values)
    {
        StringBuffer buf = new StringBuffer();
        try
```

```
        {
            if(values.size() > 0)
            {
                for(int i = 0; i < values.size(); i++)
                {
                    PermisRole av = (PermisRole)values.get(i);
                    buf.append(av.getRoleValue()).append("+");
                }

                buf.deleteCharAt(buf.length() - 1);
            }
        }
        catch(ClassCastException classcastexception) { }
        return buf.toString();
    }
    //some default values found in other classes, might not be used.
    public static final String TIMES_SEPARATOR = ";";
    public static final String DIRECTORY_NAMES_SEPARATOR = ",";
    public static final String DIGEST_ALGORITHM_NAME = "SHA1";
    public static final String AC_USE_EXPLICIT = "AC.
        UseExplicitTagEncoding";
    public static final String SEPARATOR = "|";
    public static final String LOCATION_SEPARATOR = ";";
    public static boolean WEBDAVREV = false;
    public static boolean NOREV = false;
    public static boolean AAIA = false;

    private Map vars;
    private GregorianCalendar calendar,calendarFrom, calendarTo;
    private boolean jIssuerNameCheckBox, jIssuerBCIDCheckBox,chkWebdav ;
    private SigningUtility signingUtility;
}
```

E.2 PERMIS

Our prototype uses the PERMIS engine to evaluate authorization requests. We wrote a program that waits for a modified strongSwan requesting such an evaluation, performing the evaluation using the PERMIS enginge and returning the result.

Listing E.2: PermisServer.java

```
//PermisServer

// Jan Willem Beusink 2010

import java.io.*;
import java.net.*;
import java.awt.*;

// package test.small.permis;

import java.io.File;
import java.io.IOException;

import issrg.pba.Action;
import issrg.pba.PbaException;
import issrg.pba.Response;
import issrg.pba.Subject;
```

```
import issrg.pba.Target;
import issrg.pba.rbac.BadURLException;
import issrg.pba.rbac.CustomisePERMIS;
import issrg.pba.rbac.LDAPDNPrincipal;
import issrg.pba.rbac.PermisAction;
import issrg.pba.rbac.PermisRBAC;
import issrg.pba.rbac.PermisTarget;
import issrg.pba.rbac.PolicyFinder;
import issrg.simplePERMIS.SimplePERMISPolicyFinder;
import issrg.utils.RFC2253ParsingException;


public class PermisServer {

        boolean VERBOSE = true;          // turn on/off debugging output
        ServerSocket server;
        Socket sock;
        HandleSock newSock;

        public PermisServer(int dataport) throws IOException
        {
if (VERBOSE){System.out.println("Class␣PermisServer␣started."); }
if (VERBOSE){System.out.print("opening␣serversocket...");}
                try
                {
                        server = new ServerSocket(dataport, 100);
                                if (VERBOSE){
                                        System.out.println("done.");
                                        System.out.println("Waiting␣for␣
                                            connection.");
                                }
                }
                catch ( IOException e ) {
                                e.printStackTrace();
                }
                // while loop, never ending :)
                while (true){
                        try{
if (VERBOSE){System.out.println("Waiting␣on␣new␣connection");}
                                sock = server.accept();
if (VERBOSE){System.out.println("New␣connection␣accepted");}
                                newSock = new HandleSock(sock, VERBOSE);
if (VERBOSE){System.out.println("Starting␣processing␣new␣connection");}
                                newSock.start();
                        }
                        catch ( IOException e ) {
                                e.printStackTrace();
                        }
                }//<< end while
        }

        public static void main( String args[] ) throws IOException
        {
                int port = 5010;

                if (args.length>1)
                {
                        port = (new Integer(args[0])).intValue();
                }

                PermisServer mylink = new PermisServer(port);
```

```
        /**
         *  Insert some cleanup code here?
         *  Sighup -> close link/socket
         */
        }
}


class HandleSock extends Thread {
        BufferedInputStream input;
        BufferedOutputStream output;
        String address, answer, req_str;
        Socket mySock;
        Boolean VERBOSE;
        static int BUFFSIZE = 128000;         // how many bytes our
            incoming buffer can hold
        byte data[];
        byte buff[];
        private static final char terminal = (char) '\0';

        public HandleSock(Socket thisSock, Boolean verbose){
                super();
                mySock = thisSock;
                VERBOSE = verbose;
        }

        public void run(){
//TODO do I need locking?

                        address = mySock.getInetAddress().getHostName();
if (VERBOSE){System.out.println("Connection␣from:␣" +address + "␣accepted.
    ");}

                        try{
                                input = new BufferedInputStream(mySock.
                                    getInputStream(), BUFFSIZE);
                                output = new BufferedOutputStream(mySock.
                                    getOutputStream(),BUFFSIZE);
if (VERBOSE){System.out.println("Streams␣openend␣on␣connection,␣waiting␣
    for␣request");}
                                req_str = recv_string(terminal);
if (VERBOSE){System.out.println("Received␣request␣for:␣" +req_str);}
                                answer = getPermisAuthz(req_str);
if (VERBOSE){System.out.println("Received:␣\""+answer +"\"␣from␣PERMIS␣
    ENGINE");}
                        }
                        catch( IOException io){
                                System.out.println("Error␣receiving␣packet
                                    ,␣something␣went␣very␣wrong" );
                                io.printStackTrace();
                                answer = "Java␣reported␣an␣error␣during␣
                                    receiving␣the␣request.";
                        }


if (VERBOSE){System.out.print("About␣to␣send␣answer:␣\""+answer +"\"...")
    ;}
                        try{
                                send_string(answer, terminal);
                        }
                        catch( IOException io){
```

```
                                System.out.println("Could not send answer
                                    to " + address );
                        }
if (VERBOSE){System.out.println("done.");}

                        try
                        {
                                closeSocket(mySock);
                        }catch (IOException io)
                        {
                                io.printStackTrace();
                        }
        }


  /**
   * Location of the policy.
   */
  private static final String POLICY_LOCATION = "./resources/policy.xml";

  /**
   * Directory containing the X509 ACs
   */
  private static final String REPOSITORY_LOCATION = "file://./resources/
      file-repo-root";

  /**
   * The requested action.
   */
  private static final String ACTION = "GET";

  /**
   * The requested target.
   */
  private static final String TARGET = "http://www.mysite.com/members";

  /**
   * First argument should be the distinguished name of the requesting
       entity.
   *
   * @param args
   */


        private String getPermisAuthz(String request){
//TODO stub:replace::
//      request = "cn=David,ou=staff,o=permisv5,c=gb";
try {
      CustomisePERMIS.configureX509Flavour();
      CustomisePERMIS.setAttributeCertificateAttribute("
          attributeCertificateAttribute");
    } catch (PbaException e) {
      e.printStackTrace();
      System.exit(-1);
    }

    /*
     * Use a policy in a text file.
     * No signature verification will be used.
     */
    PolicyFinder policyFinder = null;
    try {
```

```
  policyFinder = new SimplePERMISPolicyFinder(new File(POLICY_LOCATION
      ), new LDAPDNPrincipal(
      "cn=Policy␣Issuer"));
} catch (PbaException e) {
  e.printStackTrace();
  System.exit(-1);
} catch (RFC2253ParsingException e) {
  e.printStackTrace();
  System.exit(-1);
} catch (IOException e) {
  e.printStackTrace();
  System.exit(-1);
}

/*
 * Set up the engine.
 */
PermisRBAC engine = null;
try {
  engine = new PermisRBAC(policyFinder, REPOSITORY_LOCATION, null);
} catch (PbaException e) {
  e.printStackTrace();
  System.exit(-1);
}

/*
 * Get the credentials of the subject.
 */
Subject subject = null;
try {
  subject = engine.getCreds(new LDAPDNPrincipal(request));
} catch (PbaException e) {
  e.printStackTrace();
  System.exit(-1);
} catch (RFC2253ParsingException e) {
  e.printStackTrace();
  System.exit(-1);
}

/*
 * Set up action and target.
 */
Action action = new PermisAction(ACTION);
Target target = null;
try {
  target = new PermisTarget(TARGET);
} catch (BadURLException e) {
  e.printStackTrace();
  System.exit(-1);
}

/*
 * Compute the authorization decision.
 */
Response response = null;
try {
  response = engine.authzDecision(subject, action, target, null);
} catch (PbaException e) {
  System.out.println("Exception␣thrown␣during␣decision␣making" + e.
      getMessage());
  e.printStackTrace();
}
```

```
    if (response.isAuthorised()) {
      System.out.println("The request is granted.");
                return ("Granted");
    } else {
      System.out.println("The request is denied.");
                return ("Denied");
    }


        }

   // send a string down the socket
   public void send_string(String str, char terminal) throws IOException
   {
        /* convert our string into an array of bytes */
        ByteArrayOutputStream bytestream;
        bytestream = new ByteArrayOutputStream(str.length());

        DataOutputStream out;
        out = new DataOutputStream(bytestream);

        for (int i=0; i<str.length(); i++)
                out.write((byte) str.charAt(i));

        out.write((byte) terminal);

        output.write(bytestream.toByteArray(), 0, bytestream.size());
        output.flush();

        if (VERBOSE) System.out.println("Client: sending '" + str +"'");
   }

   // recv a string from the socket (terminates on terminal char)
   public String recv_string(char terminal) throws IOException
   {
        char c;
        String out;

        if (VERBOSE) System.out.println("Entered recv_string");
        // would have liked to use readUTF, but it didn't seem to work
        // when talking to the c++ server

        out = new String("");

        if (VERBOSE) System.out.print("Client: recv'd:");
        while ((c=(char) input.read())!=terminal){
                out = out + String.valueOf(c);
                if (VERBOSE) System.out.print(" " + String.valueOf(c)+",")
                    ;
        }
        if (VERBOSE) System.out.print("\n");
        if (VERBOSE) System.out.println("Client: recv'd '" + out +"'");

        return out;
   }

// shutdown the socket
        public void closeSocket(Socket sock) throws IOException
        {
                if (VERBOSE) System.out.println("Client: closing socket");
                sock.close();
```

```
    }


}
```

We also provide the client we used to jolt the system.

Listing E.3: TestServer.java

```
import java.io.*;
import java.net.*;
import java.awt.*;

//import java.io.File;
//import java.io.IOException;

public class TestServer {
        public static void main( String args[] )
        {
                if(args[0] == null || args[1] == null){
                        System.err.println( "Usage:␣java␣TestServer␣
                            server_name,␣port\n");
                        System.exit(1);
                }
                int port = Integer.parseInt(args[1]);

                PrintThread thread1, thread2, thread3, thread4, thread5,
                    thread6, thread7, thread8, thread9, thread10;

                thread1 = new PrintThread( "thread1", "cn=David,ou=staff,o
                    =permisv5,c=gb", args[0], port );
                thread2 = new PrintThread( "thread2", "CN=David,ou=staff,O
                    =permisv5,c=gb", args[0], port );
                thread3 = new PrintThread( "thread3", "cn=David,␣ou=staff,
                    ␣o=permisv5,c=gb", args[0], port );
                thread4 = new PrintThread( "thread4", "cn=David,Ou=staff,␣
                    o=permisv5,C=gb", args[0], port );
                thread5 = new PrintThread( "thread5", "/C=NL/O=WMC/CN=
                    hostA256", args[0], port );
                thread6 = new PrintThread( "thread6", "/C=NL/O=WMC/CN=
                    hostA384", args[0], port );
                thread7 = new PrintThread( "thread7", "cn=hostA256,o=WMC,C
                    =NL", args[0], port );
                thread8 = new PrintThread( "thread8", "cn=hostA256,o=WMC,c
                    =NL", args[0], port );
                thread9 = new PrintThread( "thread9", "/C=NL/O=WMC/CN=
                    hostB256", args[0], port );
                thread10 = new PrintThread( "thread10", "/C=NL/O=WMC/CN=
                    hostB384", args[0], port );

                thread1 = new PrintThread( "thread1", "/C=NL␣/O=WMC␣/CN=
                    hostB256", args[0], port );
                thread2 = new PrintThread( "thread2", "C=NL,O=WMC,CN=
                    hostB256", args[0], port );
                thread3 = new PrintThread( "thread3", "C=NL,␣O=WMC,␣CN=
                    hostB256", args[0], port );
                thread4 = new PrintThread( "thread4", "/O=WMC/C=NL/CN=
                    hostB256", args[0], port );
                thread5 = new PrintThread( "thread5", "C=NL,␣O=WMC,␣CN=
                    hostB256", args[0], port );
```

```
                thread6 = new PrintThread( "thread6", "CN=hostB256,␣C=NL/␣
                    O=WMC", args[0], port );
                thread7 = new PrintThread( "thread7", "/c=nl/o=wmc/cn=
                    hostb256", args[0], port );
                thread8 = new PrintThread( "thread8", "cn=hostA256,o=WMC,c
                    =NL", args[0], port );
                thread9 = new PrintThread( "thread9", "C=NL,O=WMC,CN=
                    hostB256", args[0], port );
                thread10 = new PrintThread( "thread10", "/C=NL/O=WMC/CN=
                    hostB384", args[0], port );




                System.err.println( "\nStarting␣threads" );

/*              thread1.start();
                thread2.start();
                thread3.start();
                thread4.start();
                thread5.start();
//              thread6.start();
*/              thread7.start();
                thread8.start();
                thread9.start();
//              thread10.start();
                System.err.println( "Threads␣started\n" );
        }
}

class PrintThread extends Thread {
        private int port;
        private String answer, req_str, server;
        private static final char terminal = (char) '\0';
        static int BUFFSIZE = 128000;         // how many bytes our
            incoming buffer can hold
        BufferedInputStream input;
        BufferedOutputStream output;
        // PrintThread constructor assigns name to thread
        // by calling Thread constructor
        public PrintThread( String name, String givenRequest, String
            servername, int serverport)
        {
                super( name );
                req_str = givenRequest;
                server = servername;
                port = serverport;
        }

   // execute the thread
        public void run()
        {
                try {
                Socket mySock = new Socket(server, port);
                input = new BufferedInputStream(mySock.getInputStream(),
                    BUFFSIZE);
                output = new BufferedOutputStream(mySock.getOutputStream()
                    ,BUFFSIZE);
                send_string(req_str, terminal);
                answer = recv_string(terminal);
                closeSocket(mySock);
```

```
                }
                catch (IOException e) {
                        System.err.println(e.toString());
                        e.printStackTrace();
                }
        }

   // send a string down the socket
   public void send_string(String str, char terminal) throws IOException
   {
        /* convert our string into an array of bytes */
        ByteArrayOutputStream bytestream;
        bytestream = new ByteArrayOutputStream(str.length());

        DataOutputStream out;
        out = new DataOutputStream(bytestream);

        for (int i=0; i<str.length(); i++)
                out.write((byte) str.charAt(i));

        out.write((byte) terminal);

        output.write(bytestream.toByteArray(), 0, bytestream.size());
        output.flush();

        System.out.println("Client:␣sending␣'" + str +"'");
   }

   // recv a string from the socket (terminates on terminal char)
   public String recv_string(char terminal) throws IOException
   {
        char c;
        String out;
        // would have liked to use readUTF, but it didn't seem to work
        // when talking to the c++ server

        out = new String("");
        while ((c=(char) input.read())!=terminal){
                out = out + String.valueOf(c);
        }
        System.out.println("Thread␣"+getName()+"␣Client:␣recv'd␣'" + out +
            "'");
        return out;
   }

// shutdown the socket
        public void closeSocket(Socket sock) throws IOException
        {
                System.out.println("Thread␣"+ getName()+"␣closing␣socket")
                    ;
                sock.close();
    }



}
```

E.3 Confidence Intervals

Our experiments generated lots of data to be used as input for calculation of Confidence Intervals (CIs). We created a program that can parse the files created by our experiments and do the calculations.

Listing E.4: RunAvarage.java

```
import java.io.*;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Vector;
import java.lang.Math;
import org.apache.commons.math.distribution.TDistribution;
import org.apache.commons.math.distribution.TDistributionImpl;
import org.apache.commons.math.MathException;

public class RunAvarage {
    public static int n = 10;

    public static boolean skipTimedOut = true;
    public static boolean outPutForLaTeX = false;
    public static boolean debug = false;
    public static double TIMEOUT = 47.0;
    public static double RUNTIME = 360.0;
    public static double WARMUP = 120.0;

    public static String perSecondString = "nanosec";
    public static double perSecondDouble = 1000000000.0;
    private HashMap<String, List<Double>>[] results;

    private double limit = RUNTIME * perSecondDouble;
    private double underlimit = WARMUP * perSecondDouble;
    /**
     * @param args
     */
    public static void main(String[] args) {
        /*
        new RunAvarage("/Users/beusink/test-results/log_exp2.1/results.
            exp1.sub1.256");
        */
        for (int i = 0; i < args.length; i++) {
            new RunAvarage(args[i]);
        }
    }

    public RunAvarage(String fileName) {
        results = new HashMap[n];

        String[] fn = fileName.split("\\.");

        if (outPutForLaTeX) {
            System.out
                .println(
```

```
            "\n\\begin{longtable}{|^c|^c|^c|^c|^c|}"
                + "\n\\caption{Experiment␣"
                + fn[fn.length - 3].substring(3)
                + "␣using␣\\acs{ECC}"
                + fn[fn.length - 1]
                + "␣certificates.\\label{table:"
                + fn[fn.length - 3]
                + "-"
                + fn[fn.length - 1]
                + "}}\\\\"
                + "\n\\hline"
                + "\nLoad␣&␣sample␣mean␣\\begin{math}\\bar{x}\\end{
                    math}"
                + "␣&␣\\begin{math}c\\frac{s}{\\sqrt{n}}\\end{math}␣&"
                + "\\begin{math}\\bar{x}-c\\frac{s}{\\sqrt{n}}\\end{
                    math}&"
                + "\\begin{math}\\bar{x}+c\\frac{s}{\\sqrt{n}}\\end{
                    math}"
                + "\\\\"
                + "\n\\hline\\hline"
                + "\n\\endfirsthead"

                + "\n\\caption[]{(continued)}\\\\"
                + "\n\\hline"
                + "\nLoad␣&␣sample␣mean␣\\begin{math}\\bar{x}\\end{
                    math}␣&"
                + "␣\\begin{math}c\\frac{s}{\\sqrt{n}}␣\\end{math}␣&"
                + "\\begin{math}\\bar{x}-c\\frac{s}{\\sqrt{n}}\\end{
                    math}&"
                + "\\begin{math}\\bar{x}+c\\frac{s}{\\sqrt{n}}\\end{
                    math}"
                + "\\\\"
                + "\n\\hline\\hline"
                + "\n\\endhead"

                + "\n\\hline"
                + "\n\\multicolumn{5}{|l|}{{Continued␣on␣Next␣Page\\
                    ldots}}"
                + "\\\\"
                + "\n\\hline"
                + "\n\\endfoot"

                + "\n\\endlastfoot");
        } else {
            System.out.println("\n" );
            /*+ fn[fn.length - 3] +
                               "\t" + fn[fn.length - 2] +
                               "\t" + fn[fn.length - 1]);
            */
        }
        String id = fn[fn.length - 3] + "."
        + fn[fn.length - 2].substring(fn[fn.length - 2].length()-1,fn[fn.
            length - 2].length())
        + "-" + fn[fn.length - 1];

        parse(fileName);
        PrintAvarages(id);
    }

    private void PrintAvarages() {
        PrintAvarages ("");
    }
```

```
    private void PrintAvarages(String id) {

        double threshold = TIMEOUT * perSecondDouble; // timeout threshold
            .
        int timeOutCount = 0, totalCount = 0;
        double[][] floatResults = new double[results[0].keySet().size()][n
            ];
        /** Determine average connection time per run. */
        for (int run = 0; run < n; run++) {
            int index = 0;
            Enumeration<String> e = Collections.enumeration(results[run]
                .keySet());
            while (e.hasMoreElements()) {
                String key = e.nextElement();
                List resultsList = results[run].get(key);

                double sum = 0;
                int successfull = 0;
                for (int j = 0; j < resultsList.size(); j++) {
                    double curValue = (Double)resultsList.get(j);
                    totalCount++;
                    if (curValue > threshold){
                        timeOutCount++;
                    }
                    if (!skipTimedOut || curValue < threshold) {
                        sum += curValue;
                        successfull++;
                    }
                }
                double avg;
                if (sum == 0 || successfull == 0) {
                    if (debug){
                        System.out

                        .println("Could␣not␣calculate,␣sum␣=␣" + sum + ",␣
                            successfull="
                                + successfull + "␣@␣run=" + run + ",␣
                                    index="
                                + index + ",␣load="
                                + results[run].keySet().toArray()[index])
                                    ;
                    }
                    avg=0;
                }
                else{
                    avg = sum / successfull;
                }
                floatResults[index][run] = avg;
                index++;
            }
        }

        if (!outPutForLaTeX) {
            System.out.println("\tsample-mean:\tdelta\tlower\tupper");
            System.out.println("load\t"+id+"\tdelta\tlower\tupper");
        }
        /**
         * Determine 95% Confidence interval based on the n runs.
         * Current implementation assumes n to be 10.
         */
        for (int load = 0; load < results[0].keySet().size(); load++) {
```

```
            double sampleMean = 0, s = 0, delta = 0, sum = 0, sigmaBase =
                0;
            int count = 0;

            for (int run = 0; run < n; run++) {
                if (floatResults[load][run] != 0){
                    sum += floatResults[load][run];
                    count++;
                }
            }

            if (count > 0){
                sampleMean = sum / (count * perSecondDouble);

                for (int run = 0; run < n; run++) {
                    if (floatResults[load][run] != 0){{
                        sigmaBase += Math
                        .pow((floatResults[load][run] / perSecondDouble)
                             - sampleMean, 2.0);
                    }
                }
                s = Math.sqrt(sigmaBase / (count - 1.0));
                delta = get_z(count) * s / Math.sqrt((double)count);
                }

            }
            else{
                System.out.println("found␣count␣=␣0␣for␣load␣" + load);
                sampleMean = delta = 0;
            }
            if (outPutForLaTeX) {
                System.out.println(results[0].keySet().toArray()[load] + "
                    \t&␣"
                                + rnd3(sampleMean) + "\t&␣" + rnd3(
                                    delta)
                                + "\t&␣" + rnd3(sampleMean - delta) + "
                                    \t&␣"
                                + rnd3(sampleMean + delta) + "\\hline")
                                    ;
            } else {
                System.out.println(results[0].keySet().toArray()[load] + "
                    \t"
                                + sampleMean + "\t" + delta + "\t"
                                + (sampleMean - delta) + "\t"
                                + (sampleMean + delta));
            }

        }


        if (outPutForLaTeX) {
            System.out.println("\\end{longtable}");
        }

        System.out.println("timed-out-connections␣" + timeOutCount +
                "␣total-connection-initiation-count-in-window␣" +
                    totalCount);
    }

    public void parse(String fileName) {
        String s;
        FileReader fr;
```

```
        BufferedReader br;

        try {
            fr = new FileReader(fileName);
            br = new BufferedReader(fr);

            while ((s = br.readLine()) != null) {
                if (s.contains(perSecondString)) {
                    parseLine(s);
                }
            }
            fr.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    private void parseLine(String line) {

        String[] lineArray = line.split("\\s+");
        int run = Integer.parseInt(lineArray[1].split(":")[1]);
        String load = lineArray[2].split(":")[1];

        //fix for modulo bug in test-script
        if (!lineArray[5].split(":")[1].startsWith("ecc")){
                return;
        }
        Double deltaT0 = Double.parseDouble(lineArray[8]);
        Double time = Double.parseDouble(lineArray[11]);
        if (deltaT0 > underlimit){
        //if > minimal warmup time
                //if not yet initiated, needed to cope with a dynamic of
                    runs.
                if (results[run - 1] == null) {
                    results[run - 1] = new LinkedHashMap();
                }
                //filtering out the connections that finish after we stop
                    generating
                //todo what to do with timeout items?
                if (deltaT0 + time < limit )
                {
                    if (results[run - 1].containsKey(load)) {
                        results[run - 1].get(load).add(time);
                    } else {
                        results[run - 1].put(load, new ArrayList<Double>()
                            );
                        results[run - 1].get(load).add(time);
                    }
                }
        }
    }

    private double rnd3(double d) {
        DecimalFormat threeDForm = new DecimalFormat("#.###");
        return Double.valueOf(threeDForm.format(d));
    }
    /**
     * returns the crititical point of a 95% two sided / 97.5% one sided
         for n degrees of freedom.
```

```
     * @param n number of degrees of freedom
     */
    private double get_z(int n){
        double z = 0.0;
        TDistributionImpl tDist =  new TDistributionImpl(n);
        try{
             z = tDist.inverseCumulativeProbability(0.025);
        }
        catch (MathException me){
            me.printStackTrace();
        }
        finally{
            if (debug){
                System.out.println("Returning z = " + z + " for n = " +n )
                    ;

            }
            return Math.abs(z);
        }

    }
}
```

Appendix F

Diff Files

This appendix contains the diff files used in our prototype.

F.1 Our Modifications

Listing F.1 lists the diff file that applies our modifications to strongSwan.

Listing F.1: 205-strongswan-conf-4.5.3-mod-beus-v04.patch

```
diff -Naur strongswan-4.5.3-orig/src/libcharon/encoding/message.c
    strongswan-4.5.3/src/libcharon/encoding/message.c
--- strongswan-4.5.3-orig/src/libcharon/encoding/message.c      2011-07-20
     16:16:31.000000000 +0200
+++ strongswan-4.5.3/src/libcharon/encoding/message.c   2011-10-20
    15:30:43.960372258 +0200
@@ -254,6 +254,7 @@
        {NOTIFY,
            ADDITIONAL_IP4_ADDRESS},
        {NOTIFY,
            ADDITIONAL_IP6_ADDRESS},
        {NOTIFY,
            NO_ADDITIONAL_ADDRESSES},
+       {NOTIFY,
    AUTHORIZATION_FAILED},
        {NOTIFY,                                                    0},
        {VENDOR_ID,                                                 0},
 };
diff -Naur strongswan-4.5.3-orig/src/libcharon/encoding/payloads/
    notify_payload.c strongswan-4.5.3/src/libcharon/encoding/payloads/
    notify_payload.c
--- strongswan-4.5.3-orig/src/libcharon/encoding/payloads/notify_payload.c
          2011-07-14 16:52:47.000000000 +0200
+++ strongswan-4.5.3/src/libcharon/encoding/payloads/notify_payload.c
    2011-10-20 15:31:40.390372195 +0200
@@ -54,9 +54,10 @@
        "USE_ASSIGNED_HoA",
        "TEMPORARY_FAILURE",
        "CHILD_SA_NOT_FOUND");
-ENUM_NEXT(notify_type_names, ME_CONNECT_FAILED, ME_CONNECT_FAILED,
    CHILD_SA_NOT_FOUND,
-       "ME_CONNECT_FAILED");
```

```
-ENUM_NEXT(notify_type_names, INITIAL_CONTACT, IPSEC_REPLAY_COUNTER_SYNC,
    ME_CONNECT_FAILED,
+ENUM_NEXT(notify_type_names, ME_CONNECT_FAILED, AUTHORIZATION_FAILED,
    CHILD_SA_NOT_FOUND,
+       "ME_CONNECT_FAILED",
+       "AUTHORIZATION_FAILED");
+ENUM_NEXT(notify_type_names, INITIAL_CONTACT, IPSEC_REPLAY_COUNTER_SYNC,
    AUTHORIZATION_FAILED,
        "INITIAL_CONTACT",
        "SET_WINDOW_SIZE",
        "ADDITIONAL_TS_POSSIBLE",
@@ -139,9 +140,10 @@
        "ASSIGNED_HoA",
        "TEMP_FAIL",
        "NO_CHILD_SA");
-ENUM_NEXT(notify_type_short_names, ME_CONNECT_FAILED, ME_CONNECT_FAILED,
    CHILD_SA_NOT_FOUND,
-       "ME_CONN_FAIL");
-ENUM_NEXT(notify_type_short_names, INITIAL_CONTACT,
    IPSEC_REPLAY_COUNTER_SYNC, ME_CONNECT_FAILED,
+ENUM_NEXT(notify_type_short_names, ME_CONNECT_FAILED,
    AUTHORIZATION_FAILED, CHILD_SA_NOT_FOUND,
+       "ME_CONN_FAIL",
+       "AUTHZ_FAIL");
+ENUM_NEXT(notify_type_short_names, INITIAL_CONTACT,
    IPSEC_REPLAY_COUNTER_SYNC, AUTHORIZATION_FAILED,
        "INIT_CONTACT",
        "SET_WINSIZE",
        "ADD_TS_POSS",
diff -Naur strongswan-4.5.3-orig/src/libcharon/encoding/payloads/
    notify_payload.h strongswan-4.5.3/src/libcharon/encoding/payloads/
    notify_payload.h
--- strongswan-4.5.3-orig/src/libcharon/encoding/payloads/notify_payload.h
          2011-07-14 16:52:47.000000000 +0200
+++ strongswan-4.5.3/src/libcharon/encoding/payloads/notify_payload.h
    2011-10-20 15:30:43.960372258 +0200
@@ -70,6 +70,7 @@

        /* IKE-ME, private use */
        ME_CONNECT_FAILED = 8192,
+       AUTHORIZATION_FAILED = 8193,

        /* notify status messages */
        INITIAL_CONTACT = 16384,
diff -Naur strongswan-4.5.3-orig/src/libcharon/sa/tasks/ike_auth.c
    strongswan-4.5.3/src/libcharon/sa/tasks/ike_auth.c
--- strongswan-4.5.3-orig/src/libcharon/sa/tasks/ike_auth.c     2011-02-05
     07:40:34.000000000 +0100
+++ strongswan-4.5.3/src/libcharon/sa/tasks/ike_auth.c  2011-10-20
    15:37:56.240372801 +0200
@@ -14,10 +14,17 @@
  * for more details
  */

+#define BUFFSIZE 65536
+#define MAX_PERMIS_REQUEST_LENGTH 2048
+#define MAX_AUTHZ_REPLY_LENGTH 1024
+#define TERMINALCHAR '\0'
+
 #include "ike_auth.h"

 #include <string.h>
```

```
+#include <netdb.h>
+
 #include <daemon.h>
 #include <encoding/payloads/id_payload.h>
 #include <encoding/payloads/auth_payload.h>
@@ -111,6 +118,42 @@
          * received an INITIAL_CONTACT?
          */
         bool initial_contact;
+
+       /**
+        * should we do authorization after authentication?
+        */
+       bool do_authz;
+
+       /**
+        * should we send a AUTHORIZATION_FAILED notify?
+        */
+       bool authorization_failed;
+
+       /**
+        * should we do webdav based authentication?
+        */
+       bool do_webdav;
+
+       /**
+        * Is the WebDAV retrieved certificate identical to the one given
    in the IKE exchange?
+        */
+       cert_validation_t webdav_validation;
+
+       /**
+        * The local PERMIS server FQDN or IP address
+       */
+/*      * RFC 1034: To simplify implementations, the total number of
    octets that
+        *  represent a domain name (i.e., the sum of all label octets and
     label
+        *  lengths) is limited to 255.
+        * So including the trailing terminating character we need 256
    bytes.*/
+
+       char *permis_server_name;
+
+       /**
+        * On what port is the PERMIS server listening?
+        */
+       int permis_port;
+
 };

 /**
@@ -489,6 +532,16 @@
         id_payload_t *id_payload;
         identification_t *id;

+
+               /** authz_req can not be longer than the DN identication
    printf_hook */
+               char authz_req[MAX_PERMIS_REQUEST_LENGTH];
```

```
+               char authz_respons[MAX_AUTHZ_REPLY_LENGTH], buffer[
    BUFFSIZE];
+               auth_payload_t *auth_payload;
+               int permis_fd, numbytes, i, j, end;
+               bool authz_result = NULL;
+               struct hostent *other_hostent;
+               struct sockaddr_in other_addr;
+
        if (message->get_exchange_type(message) == IKE_SA_INIT)
        {
                return collect_other_init_data(this, message);
@@ -629,6 +682,216 @@
                return NEED_MORE;
        }

+DBG1(DBG_IKE, "DEBUG␣POINT␣4,␣do_authz=%c",this->do_authz);
+       if (this->do_authz)
+       {
+                       DBG1(DBG_IKE, "Processing␣authorization");
+       //precheck for authz creating
+                       auth_payload = (auth_payload_t*)message->
    get_payload(message, AUTHENTICATION);
+                       if (auth_payload == NULL)
+                       { // TODO check if this is eap only error,
    apparently no auth payload means EAP
+       DBG1(DBG_IKE, "DEBUG␣POINT␣5");
+                               DBG0(DBG_IKE, "Error:␣EAP␣is␣an␣invalid␣
    authentication␣type␣for␣use"
+                                       "␣with␣PERMIS␣server;␣expected␣a␣
    certificate␣based␣authentication␣"
+                                       "type.␣Please␣configure␣strongSwan
    ␣for␣certificate␣only␣when␣"
+                                       "combining␣with␣PERMIS");
+                               this->authorization_failed = TRUE;
+                               return NEED_MORE;
+                       }
+
+                       /**
+                       * If and only if this is a certificate containing
    a DN we can process
+                       * PERMIS authorization
+                       */
+                       if (    (uintptr_t)cfg->get(cfg,
    AUTH_RULE_AUTH_CLASS) ==
+                                       AUTH_CLASS_PUBKEY && id->get_type(
    id) == ID_DER_ASN1_DN )
+                       {
+
+                               if (snprintf(authz_req,
    MAX_PERMIS_REQUEST_LENGTH-1, "%Y",id) == -1)
+                               {
+                                       //TODO report error msg.
+                               }
+
+                               //authz_req = "cn=David,ou=staff,o=
    permisv5,c=gb";
+       DBG1(DBG_IKE, "DEBUG␣POINT␣6:␣Found␣id:␣'%Y'.", id);
+//BEUS: this is were we set the request to ask the permis enginge
+
+// this needs to be extended.
+                       }
+                       else{
```

```
+       DBG1(DBG_IKE, "DEBUG␣POINT␣7");
+                               DBG1(DBG_IKE, "Found␣authentication␣method
    :␣'%N'.",
+                                       auth_method_names, auth_payload->
    get_auth_method(auth_payload));
+                               DBG1(DBG_IKE, "Found␣authentication␣class:
    ␣'%N'.", auth_class_names,
+                                       ((uintptr_t)cfg->get(cfg,
    AUTH_RULE_AUTH_CLASS)));
+                               DBG1(DBG_IKE, "Found␣id␣type␣name:␣'%N'.",
     id_type_names,
+                                       id->get_type(id));
+
+                               DBG0(DBG_IKE, "Could␣not␣get␣distinguished
    ␣name␣from␣certificate;␣"
+                                       "can␣not␣perform␣PERMIS␣
    authorization␣on␣'%Y'",id);
+                               this->authorization_failed = TRUE;
+                               return NEED_MORE;
+                       }
+
+
+DBG1(DBG_IKE, "DEBUG␣POINT␣8:␣do␣permis␣authorization");
+//BEUS// BEGIN: This should not be in this class, but passed onto this
    class
+               /** TODO The functionality of communicating with PERMIS
    should be done
+               * in a seperate Class. Setting up a connection an tearing
    it down for
+               * every authorization request is not efficient when under
    a lot of load.
+               * Rather the connection should be constructed on forehand
    or upon first
+               * request and and remain openened (until a specified time
    of inactivity)
+               * allowing multiple requests to me made with the PERMIS
    server. Thus
+               * allowing for better scalability. After a timeout the
    connection should
+               * be initialized again upon first request.
+               */
+
+               other_hostent = gethostbyname(this->permis_server_name);
+               if (other_hostent==NULL)
+               {
+                       DBG0(DBG_IKE, "Error␣resolving␣PERMIS␣hostname␣\"%
    s\"␣can␣not␣"
+                               "connect␣to␣PERMIS;␣can␣not␣authorize.␣
    Giving␣up␣on␣"
+                               "authorization.", this->permis_server_name
    );
+                       this->authorization_failed = TRUE;
+                       return NEED_MORE;
+               }
+
+DBG1(DBG_IKE, "DEBUG␣POINT␣9");
+               //mysocket = socket(int socket_family, int socket_type,
    int protocol);
+               if ((permis_fd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
+               {
+                       DBG0(DBG_IKE, "Error:␣could␣not␣create␣socket;␣can
    ␣not␣connect␣to␣"
```

```
+                               "PERMIS; can not authorize. Giving up on
    authorization.");
+                       this->authorization_failed = TRUE;
+                       return NEED_MORE;
+               }
+               other_addr.sin_family = AF_INET;
+               other_addr.sin_port = htons(this->permis_port);
+               other_addr.sin_addr = *((struct in_addr *) other_hostent->
    h_addr);
+               bzero(&(other_addr.sin_zero), 8);
+               if (connect(permis_fd, (struct sockaddr *)&other_addr,
    sizeof(struct
+               sockaddr)) == -1)
+               {
+DBG1(DBG_IKE, "DEBUG POINT 10");
+                       DBG0(DBG_IKE, "Error: could not connect to PERMIS
    server; can not "
+                               "authorize. Giving up on authorization.");
+                       this->authorization_failed = TRUE;
+                       return NEED_MORE;
+               }
+
+
+               //send authz request
+               DBG1(DBG_IKE, "authz_req='%s', length = %d",authz_req,
    strlen(authz_req) );
+//TODO check wether the terminal character needs to be send after
+               if (send(permis_fd, (char *) authz_req, strlen(authz_req)
    +1, 0) == -1)
+               {
+DBG1(DBG_IKE, "DEBUG POINT 12");
+                       DBG0(DBG_IKE, "Error sending authorization request
    : '%s' Can not: "
+                               "authorize.", authz_req);
+//                     free(authz_req);
+                       close(permis_fd);
+                       this->authorization_failed = TRUE;
+                       return NEED_MORE;
+               }
+//             free(authz_req); /* no further use for this dynamic char[]
     */ // TODO Remove me, no longer char* but char[]
+DBG1(DBG_IKE, "DEBUG POINT 13");
+
+       //receive and set authz_respons[]
+               /**
+                * After the following while loop, char authz_response[]
    contains the
+                * response received, if more than MAX_AUTHZ_REPLY_LENGTH
    -1 bytes are
+                * received only the first MAX_AUTHZ_REPLY_LENGTH-1 bytes
    are stored.
+                *
+                * The java end is sending a char at a time, so we recv
    some chars
+                * (probably 1), append it to our str string, then carry
    on until we see
+                * the terminal character.
+                */
+
+               numbytes = end = j = 0;
+               while (!end)
+               {       //recv blocks until a message is received
```

```
+DBG1(DBG_IKE, "DEBUG␣POINT␣14␣(loop)");
+                       if ((numbytes=recv(permis_fd, buffer, BUFFSIZE, 0)
    )==-1)
+                       {
+                               DBG0(DBG_IKE, "Error␣receiving␣
    authorization␣request.");
+                       }
+                       for (i=0; i<numbytes; i++)
+                       {
+                               authz_respons[j] = buffer[i];
+                               j++;
+                       }
+DBG1(DBG_IKE, "DEBUG␣POINT␣14␣(loop):␣received␣%d␣bytes:'%s'",numbytes,
    buffer);
+                       if ((buffer[i-1]==TERMINALCHAR) || (j==
    MAX_AUTHZ_REPLY_LENGTH-1))
+                       {
+                               end = 1;
+                       }
+               }
+DBG1(DBG_IKE, "DEBUG␣POINT␣15");
+               authz_respons[j] = '\0'; //append terminal char just to be
     sure
+
+//TODO closing the socket, including opening it should be done in calling
     class
+               close(permis_fd);
+
+       /**
+        * From Implementing Role Based Access Controls using X.509
    Privilege
+        * Management - the PERMIS Authorisation Infrastructure.
+        * DW Chadwick and A Otenko
+        * In Borka Jerman-Blazic, Wolfgang Schneider, and Tomaz Klobucar,
     editors,
+        * Security and Privacy in Advanced Networking Technologies, NATO
    Science
+        * Series, pages 26-39. IOS Press, 2004 Proceedings of the NATO
    Advanced
+        * Networking Workshop on Advanced Security Technologies in
    Networking,
+        * Bled, Slovenia, 15-18 September 2003.
+        * http://www.cs.kent.ac.uk/pubs/2004/2279
+        *
+        * Once the user has been successfully authenticated he will
    attempt to
+        * perform actions on the target. At each attempt, the AEF passes
    the
+        * subject object, the target name, and the attempted action along
     with its
+        * parameters, to the ADF via a call to Decision. Decision checks
    if the
+        * action is allowed for the roles that the user has, taking into
    account
+        * all the conditions specified in the TargetAccessPolicy. If the
    action is
+        * allowed, Decision returns Granted, if it is not allowed it
    returns Denied
+        * .
+        */
+DBG1(DBG_IKE, "DEBUG␣POINT␣16");
+               if (strcmp(authz_respons,"Granted")==0)
```

```
+               {
+                       authz_result = TRUE;
+                       DBG1(DBG_IKE, "Received␣an␣Authorization␣granted␣
    message␣from␣"
+                               "PERMIS␣server.");
+               }
+               else if (strcmp(authz_respons, "Denied")==0)
+               {
+                       authz_result = FALSE;
+                       DBG1(DBG_IKE, "Received␣an␣Authorization␣Denied␣
    message␣from␣PERMIS"
+                               "␣server.");
+               }
+               else
+               {
+                       authz_result = FALSE;
+                       DBG0(DBG_IKE, "Received␣an␣unexpected␣answer␣from␣
    PERMIS␣server:␣"
+                               "'%s'.␣Can␣not␣perform␣authorization",
    authz_respons);
+               }
+DBG1(DBG_IKE, "DEBUG␣POINT␣17");
+               switch (authz_result)
+               {
+                       case TRUE:
+                               DBG1(DBG_IKE, "Authorization␣granted.");
+                               this->authorization_failed = FALSE;
+                               break;
+                       case FALSE:
+                               DBG1(DBG_IKE, "Authorization␣denied.");
+                               this->authorization_failed = TRUE;
+                               return NEED_MORE;
+                       default:
+                               DBG0(DBG_IKE, "Authorization␣process␣
    produced␣unexpected␣reply."
+                                       "␣Authorization␣failed.");
+                               return FAILED;
+               }
+       }
+
+
+DBG1(DBG_IKE, "DEBUG␣POINT␣20,␣beus'␣code␣ends␣here");
+
        /* store authentication information */
        cfg = auth_cfg_create();
        cfg->merge(cfg, this->ike_sa->get_auth_cfg(this->ike_sa, FALSE),
            FALSE);
@@ -673,6 +936,13 @@
                return FAILED;
        }

+/* todo when is peer_cfg set? should this message be inserted here? */
+       if (this->authorization_failed)
+       {
+               message->add_notify(message, TRUE, AUTHORIZATION_FAILED,
    chunk_empty);
+               return FAILED;
+       }
+
        if (this->my_auth == NULL && this->do_another_auth)
        {
                identification_t *id, *id_cfg;
```

```
@@ -1047,6 +1317,8 @@
        this->do_another_auth = TRUE;
        this->expect_another_auth = TRUE;
        this->authentication_failed = FALSE;
+/* todo; check if a this->authorization_failed = ... is needed;*/
+/* todo; check if a this->do_authz = ... is needed;*/
        this->candidates = linked_list_create();
 }

@@ -1064,6 +1336,9 @@
        free(this);
 }

+
+
+
 /*
  * Described in header.
  */
@@ -1086,6 +1361,11 @@
                .candidates = linked_list_create(),
                .do_another_auth = TRUE,
                .expect_another_auth = TRUE,
+               .do_webdav = lib->settings->get_bool(lib->settings, "
    charon.plugins.webdav.enable", FALSE),
+               .do_authz = lib->settings->get_bool(lib->settings,   "
    charon.plugins.permis.enable", FALSE),
+               .permis_server_name = lib->settings->get_str(lib->settings
    , "charon.plugins.permis.servername","127.0.0.1"),
+               .permis_port = lib->settings->get_int(lib->settings,"
    charon.plugins.permis.port",5010),
+
        );
        if (initiator)
        {
diff -Naur strongswan-4.5.3-orig/src/libstrongswan/plugins/curl/
    curl_fetcher.c strongswan-4.5.3/src/libstrongswan/plugins/curl/
    curl_fetcher.c
--- strongswan-4.5.3-orig/src/libstrongswan/plugins/curl/curl_fetcher.c
    2011-04-08 07:50:20.000000000 +0200
+++ strongswan-4.5.3/src/libstrongswan/plugins/curl/curl_fetcher.c
    2011-10-20 15:30:43.960372258 +0200
@@ -111,6 +111,9 @@
                case CURLE_OK:
                        status = SUCCESS;
                        break;
+               case CURLE_REMOTE_FILE_NOT_FOUND:
+                       status = NOT_FOUND;
+                       break;
                default:
                        DBG1(DBG_LIB, "libcurl␣http␣request␣failed:␣%s",
                             error);
                        status = FAILED;
diff -Naur strongswan-4.5.3-orig/src/libstrongswan/plugins/revocation/
    revocation_validator.c strongswan-4.5.3/src/libstrongswan/plugins/
    revocation/revocation_validator.c
--- strongswan-4.5.3-orig/src/libstrongswan/plugins/revocation/
    revocation_validator.c   2011-02-10 15:54:26.000000000 +0100
+++ strongswan-4.5.3/src/libstrongswan/plugins/revocation/
    revocation_validator.c        2011-10-20 15:30:43.960372258 +0200
@@ -25,6 +25,10 @@
 #include <credentials/sets/ocsp_response_wrapper.h>
```

```
 #include <selectors/traffic_selector.h>

+#include <ctype.h>
+#define DEFAULT_TIMEOUT 10
+#define MAX_WEBDAV_REQUEST_LENGTH 2048
+
 typedef struct private_revocation_validator_t
     private_revocation_validator_t;

 /**
@@ -206,8 +210,259 @@
        return best;
 }

+
+
+/* Converts an integer value to its hex character*/
+static char to_hex(char code) {
+  static char hex[] = "0123456789abcdef";
+  return hex[code & 15];
+}
+
+/* Returns a url-encoded version of str */
+/* IMPORTANT: be sure to free() the returned string after use */
+static char *url_encode(char *str) {
+  char *pstr = str, *buf = malloc(strlen(str) * 3 + 1), *pbuf = buf;
+  while (*pstr) {
+    if (isalnum(*pstr) || *pstr == '-' || *pstr == '_' || *pstr == '.' ||
     *pstr == '~')
+      *pbuf++ = *pstr;
+    else if (*pstr == ' ')
+      *pbuf++ = '+';
+    else
+      *pbuf++ = '%', *pbuf++ = to_hex(*pstr >> 4), *pbuf++ = to_hex(*pstr
     & 15);
+    pstr++;
+  }
+  *pbuf = '\0';
+  return buf;
+}
+
+
+
+/**
+ * Returns a *char[] containg the full URL-encoded URL for retrieving id'
    s
+ * certificate(s) from a WebDAV server or NULL if the request including
    the
+ * trailing null character exceeded MAX_WEBDAV_REQUEST_LENGTH
+ * Be sure to free the returned *char after use.
+ */
+static char *get_webdav_url(char *base_url, identification_t *subject)
+{
+//TODO this function is overflow save, but does not yet report it. report
     overflow.
+
+       char *result, dn[MAX_WEBDAV_REQUEST_LENGTH], *dn_ptr, *encoded;
+
+       if (snprintf(dn, MAX_WEBDAV_REQUEST_LENGTH-1, "%Y", subject) ==
    -1)
+       {
+               return NULL;
```

```
+       }
+       /* replace ", " with '/'  */
+       dn_ptr = dn;
+       while (dn_ptr)
+       {
+               dn_ptr = strstr(dn_ptr, ",␣");
+               if (dn_ptr)
+               {
+                       *dn_ptr++ = '/';
+                       /* remove space */
+                       strcpy(dn_ptr, ++dn_ptr);
+               }
+       }
+       /* replace ',' with '/'  */
+       dn_ptr = dn;
+       while (dn_ptr)
+       {
+               dn_ptr = strchr(dn_ptr,',');
+               if (dn_ptr)
+               {
+                       *dn_ptr++ = '/';
+               }
+       }
+
+       result = malloc(MAX_WEBDAV_REQUEST_LENGTH);
+       if(strlen(result)+strlen(base_url) < MAX_WEBDAV_REQUEST_LENGTH -1)
+       {
+               strncpy(result, base_url, MAX_WEBDAV_REQUEST_LENGTH-1);
+       }
+       else
+       {
+               return NULL;
+       }
+       /* removing possible double / */
+
+       if ( (result[strlen(result)] == dn[0]) == '/')
+       {
+               /* remove traling '/' of result by replacing it with a
    null char */
+               result[strlen(result)] = '\0';
+       }
+       if(strlen(result)+strlen(dn) < MAX_WEBDAV_REQUEST_LENGTH -1)
+       {
+               strncat(result, dn, MAX_WEBDAV_REQUEST_LENGTH - strlen(
    result) -1);
+       }
+       else
+       {
+               return NULL;
+       }
+
+       /* Make it URL encoded */
+       encoded = url_encode(result);
+       if(strlen(result)+strlen(encoded) < MAX_WEBDAV_REQUEST_LENGTH -1)
+       {
+               strncpy (result, encoded, MAX_WEBDAV_REQUEST_LENGTH-1);
+       }
+       else
+       {
+               result = NULL;
+       }
+       free(encoded);
```

```
+       return result;
+}
+
+
+/**
+ * Do a WebDAV request
+ */
+static status_t fetch_webdav(char *url,        certificate_t *subject,
    certificate_t *issuer, chunk_t *receive)
+{
+       char *full_url;
+       status_t response;
+       int timeout;
+
+       full_url = get_webdav_url(url, subject->get_subject(subject));
+       timeout = lib->settings->get_int(
+               lib->settings, "charon.plugins.webdav.timeout",
    DEFAULT_TIMEOUT);
+       if (!full_url){
+               DBG0(DBG_CFG, "␣␣could␣not␣fetch␣WebDAV␣status␣for␣'%Y'␣
    from␣'%s';␣The␣"
+                                                "request␣size␣is␣too␣large
    .",
+                                                subject->get_subject(
    subject), url);
+       }
+
+DBG0(DBG_CFG, "␣␣requesting␣WebDAV␣status␣for␣'%Y'␣at␣'%s'␣using␣URI:'%s
    '...",
+        subject->get_subject(subject), url, full_url);
+
+       DBG3(DBG_CFG, "␣␣requesting␣WebDAV␣status␣from␣'%s'␣...", url);
+       response = lib->fetcher->fetch(lib->fetcher, full_url, receive,
+                                                       FETCH_REQUEST_TYPE
    , "application/http-request",
+                                                       FETCH_TIMEOUT,
    timeout, FETCH_END);
+       if (response != SUCCESS)
+       {
+               DBG3(DBG_CFG, "␣␣fetcher␣returned␣status␣type␣%N␣when␣"
+                                                "requesting␣WebDAV␣status␣
    from␣'%s'␣...",
+                                                status_names ,response ,
    url);
+       }
+       free(full_url);
+
+       if (!response)
+       {
+               DBG1(DBG_CFG, "parsing␣WebDAV␣response␣failed");
+               return FAILED;
+       }
+       return response;
+}
+
+
+
+/**
+ * validate a x509 certificate using WebDAV
+ */
+static cert_validation_t check_webdav(x509_t *subject, x509_t *issuer,
+                auth_cfg_t *auth)
```

```
+{
+       enumerator_t *enumerator;
+       cert_validation_t valid = VALIDATION_SKIPPED;
+       status_t result;
+       char *uri = NULL;
+       chunk_t *retrieved_chunk = NULL;
+       x509_t *retrieved_cert;
+
+
+       /* lookup cache for valid WebDAV responses */
+       //TODO
+       //      if (cached_as_revoked)
+       //      {
+       //              valid = VALIDATION_REVOKED;
+       //      }
+
+       /* fallback to URL fetching from subject certificate's URIs */
+       if (valid != VALIDATION_REVOKED)
+       {
+               enumerator = subject->create_ocsp_uri_enumerator(subject);
+               while (enumerator->enumerate(enumerator, &uri))
+               {
+                       result = fetch_webdav( uri, &subject->interface,
+                                       &issuer->interface,
    retrieved_chunk);
+                       DBG3(DBG_CFG, "  fetcher returned status type %N
    when requesting "
+                                                       "WebDAV status
    from '%s' ...", status_names,
+                                                       result, uri);
+                       if (result == SUCCESS)
+                       {
+                               retrieved_cert = lib->creds->create(lib->
    creds, CRED_CERTIFICATE,
+                                CERT_X509, BUILD_BLOB_ASN1_DER,
    retrieved_chunk, BUILD_END);
+
+                               /**
+                                * if the certificate retrieved from the
    uri protected in the
+                                * subject certificate is a bitwise match
    it is validated,
+                                * otherwise validation check failed.
+                                */
+DBG0(DBG_CFG, "About to compare certificates webdav style");
+                               valid = ( (&subject->interface)->equals(&
    subject->interface, &retrieved_cert->interface) ? VALIDATION_GOOD :
    VALIDATION_FAILED);
+DBG0(DBG_CFG, "Comparing certificates found: %N",cert_validation_names ,
    valid);
+                               break; /* no need to check other URI's */
+                       }
+                       if (result == NOT_FOUND)
+                       {
+                               /**
+                                * If the certificate could not be
    retrieved from the uri
+                                * protected in the subject certificate
    but connection to uri
+                                * succeeded, i.e. 404 error, validation
    is considered revoked.
+                                */
```

```
+                               valid = VALIDATION_REVOKED;
+                               //TODO cache the revocation
+                               break; /* no need to check other URI's */
+                       }
+//                     if(result == PARSE_ERROR)
+//                     {
+//                             /**
+//                              * Apparently the certificate could not be
     correctly retrieved
+//                              * from the uri in the subject certificate
     but connection to uri
+//                              * succeeded, assuming bad configured
    server of malformed
+//                              * subject certificate. Validation is
    considered failed.
+//                              */
+//                             valid = VALIDATION_FAILED;
+//                     }
+                       if(result == NOT_SUPPORTED || FAILED )
+                       {
+                               DBG0(DBG_CFG, "␣␣fetcher␣returned␣status␣
    type␣%N␣when␣"
+                                                               "requesting␣WebDAV
    ␣status␣from␣'%s'␣...␣check␣"
+                                                               "configuration␣",
    status_names , result, uri);
+                               valid = VALIDATION_STALE;
+                               continue;
+                       }
+               }
+               enumerator->destroy(enumerator);
+       }
+
+
+       /* an uri was found, but no result. switch validation state to
    failed */
+       if (valid == VALIDATION_SKIPPED && uri)
+       {
+               valid = VALIDATION_FAILED;
+       }
+//TODO check
+//     if (auth)
+//     {
+//             auth->add(auth, AUTH_RULE_OCSP_VALIDATION, valid);
+//             if (valid == VALIDATION_GOOD)
+//             {       /* successful OCSP check fulfills also CRL
    constraint */
+//                     auth->add(auth, AUTH_RULE_CRL_VALIDATION,
    VALIDATION_GOOD);
+//             }
+//     }
+//     DESTROY_IF(best);
+
+       return valid;
+}
+
 /**
  * validate a x509 certificate using OCSP
+ * Currently WebDAV is implemented using a OCSP related field in the
+ * certificate. If WebDAV is enabled return WebDAV validation status
    instead.
  */
```

```
 static cert_validation_t check_ocsp(x509_t *subject, x509_t *issuer,
                                                                    auth_cfg_t
                                                                        *
                                                                        auth
                                                                        )

@@ -220,6 +475,13 @@
         chunk_t chunk;
         char *uri = NULL;

+       if (lib->settings->get_bool(lib->settings,
+                "charon.plugins.webdav.enabled",FALSE))
+       {
+               DBG3(DBG_CFG, "Performing␣WebDAV␣validation␣instead␣of␣
    normal␣OCSP");
+               return check_webdav(subject, issuer, auth);
+       }
+
         /** lookup cache for valid OCSP responses */
         enumerator = lib->credmgr->create_cert_enumerator(lib->credmgr,
                                                                CERT_X509_OCSP_RESPONSE
                                                                    ,
                                                                    KEY_ANY
                                                                    , NULL
                                                                    ,
                                                                    FALSE)
                                                                    ;
```

F.2 Tobias' Patch

Listing F.2 and F.3 list the patch supplied by Tobias Brunner [19, 20].

Listing F.2: tobias-patch1.diff

```
--- a/src/libcharon/plugins/stroke/stroke_socket.c
+++ b/src/libcharon/plugins/stroke/stroke_socket.c
@@ -1,4 +1,5 @@
 /*
+ * Copyright (C) 2011 Tobias Brunner
  * Copyright (C) 2008 Martin Willi
  * Hochschule fuer Technik Rapperswil
  *
@@ -25,7 +26,10 @@

 #include <hydra.h>
 #include <daemon.h>
+#include <threading/mutex.h>
 #include <threading/thread.h>
+#include <threading/condvar.h>
+#include <utils/linked_list.h>
 #include <processing/jobs/callback_job.h>

 #include "stroke_config.h"
@@ -35,6 +39,12 @@
 #include "stroke_attribute.h"
 #include "stroke_list.h"

+/**
```

```
+ * To avoid clogging the thread pool with (blocking) jobs, we limit the
    number
+ * of concurrently handled stroke commands.
+ */
+#define MAX_CONCURRENT 4
+
 typedef struct stroke_job_context_t stroke_job_context_t;
 typedef struct private_stroke_socket_t private_stroke_socket_t;

@@ -56,7 +66,32 @@ struct private_stroke_socket_t {
         /**
          * job accepting stroke messages
          */
-        callback_job_t *job;
+        callback_job_t *receiver;
+
+        /**
+         * job handling stroke messages
+         */
+        callback_job_t *handler;
+
+        /**
+         * queued stroke commands
+         */
+        linked_list_t *commands;
+
+        /**
+         * lock for command list
+         */
+        mutex_t *mutex;
+
+        /**
+         * condvar to signal the arrival or completion of commands
+         */
+        condvar_t *condvar;
+
+        /**
+         * the number of currently handled commands
+         */
+        u_int handling;

         /**
          * configuration backend
@@ -84,7 +119,7 @@ struct private_stroke_socket_t {
         stroke_ca_t *ca;

         /**
-         * Status information logging
+         * status information logging
          */
         stroke_list_t *list;
 };
@@ -489,6 +524,18 @@ static void stroke_job_context_destroy(
     stroke_job_context_t *this)
 }

 /**
+ * called to signal the completion of a command
+ */
+static inline job_requeue_t job_processed(private_stroke_socket_t *this)
+{
+        this->mutex->lock(this->mutex);
```

```
+       this->handling--;
+       this->condvar->signal(this->condvar);
+       this->mutex->unlock(this->mutex);
+       return JOB_REQUEUE_NONE;
+}
+
+/**
  * process a stroke request from the socket pointed by "fd"
  */
 static job_requeue_t process(stroke_job_context_t *ctx)
@@ -506,7 +553,7 @@ static job_requeue_t process(stroke_job_context_t *ctx
     )
         {
                 DBG1(DBG_CFG, "reading␣length␣of␣stroke␣message␣failed:␣%s
                     ",
                         strerror(errno));
-               return JOB_REQUEUE_NONE;
+               return job_processed(this);
         }

         /* read message */
@@ -515,14 +562,14 @@ static job_requeue_t process(stroke_job_context_t *
    ctx)
         if (bytes_read != msg_length)
         {
                 DBG1(DBG_CFG, "reading␣stroke␣message␣failed:␣%s",
                     strerror(errno));
-               return JOB_REQUEUE_NONE;
+               return job_processed(this);
         }

         out = fdopen(strokefd, "w+");
         if (out == NULL)
         {
                 DBG1(DBG_CFG, "opening␣stroke␣output␣channel␣failed:␣%s",
                     strerror(errno));
-               return JOB_REQUEUE_NONE;
+               return job_processed(this);
         }

         DBG3(DBG_CFG, "stroke␣message␣%b", (void*)msg, msg_length);
@@ -599,11 +646,38 @@ static job_requeue_t process(stroke_job_context_t *
    ctx)
         fclose(out);
         /* fclose() closes underlying FD */
         ctx->fd = 0;
-       return JOB_REQUEUE_NONE;
+       return job_processed(this);
 }

 /**
- * Implementation of private_stroke_socket_t.stroke_receive.
+ * Handle queued stroke commands
+ */
+static job_requeue_t handle(private_stroke_socket_t *this)
+{
+       stroke_job_context_t *ctx;
+       callback_job_t *job;
+       bool oldstate;
+
+       this->mutex->lock(this->mutex);
```

```
+       thread_cleanup_push((thread_cleanup_t)this->mutex->unlock, this->
    mutex);
+       oldstate = thread_cancelability(TRUE);
+       while (this->commands->get_count(this->commands) == 0 ||
+                   this->handling >= MAX_CONCURRENT)
+       {
+               this->condvar->wait(this->condvar, this->mutex);
+       }
+       thread_cancelability(oldstate);
+       this->commands->remove_first(this->commands, (void**)&ctx);
+       this->handling++;
+       thread_cleanup_pop(TRUE);
+       job = callback_job_create_with_prio((callback_job_cb_t)process,
    ctx,
+                       (void*)stroke_job_context_destroy, this->handler,
    JOB_PRIO_HIGH);
+       lib->processor->queue_job(lib->processor, (job_t*)job);
+       return JOB_REQUEUE_DIRECT;
+}
+
+/**
+ * Accept stroke commands and queue them to be handled
  */
 static job_requeue_t receive(private_stroke_socket_t *this)
 {
@@ -611,7 +685,6 @@ static job_requeue_t receive(private_stroke_socket_t *
    this)
        int strokeaddrlen = sizeof(strokeaddr);
        int strokefd;
        bool oldstate;
-       callback_job_t *job;
        stroke_job_context_t *ctx;

        oldstate = thread_cancelability(TRUE);
@@ -624,17 +697,18 @@ static job_requeue_t receive(private_stroke_socket_t
     *this)
                return JOB_REQUEUE_FAIR;
        }

-       ctx = malloc_thing(stroke_job_context_t);
-       ctx->fd = strokefd;
-       ctx->this = this;
-       job = callback_job_create_with_prio((callback_job_cb_t)process,
-                       ctx, (void*)stroke_job_context_destroy, this->job,
     JOB_PRIO_HIGH);
-       lib->processor->queue_job(lib->processor, (job_t*)job);
+       INIT(ctx,
+               .fd = strokefd,
+               .this = this,
+       );
+       this->mutex->lock(this->mutex);
+       this->commands->insert_last(this->commands, ctx);
+       this->condvar->signal(this->condvar);
+       this->mutex->unlock(this->mutex);

        return JOB_REQUEUE_FAIR;
 }

-
 /**
  * initialize and open stroke socket
  */
```

```
@@ -682,7 +756,11 @@ static bool open_socket(private_stroke_socket_t *this
    )
 METHOD(stroke_socket_t, destroy, void,
        private_stroke_socket_t *this)
 {
-       this->job->cancel(this->job);
+       this->handler->cancel(this->handler);
+       this->receiver->cancel(this->receiver);
+       this->commands->destroy_function(this->commands, (void*)
    stroke_job_context_destroy);
+       this->condvar->destroy(this->condvar);
+       this->mutex->destroy(this->mutex);
        lib->credmgr->remove_set(lib->credmgr, &this->ca->set);
        lib->credmgr->remove_set(lib->credmgr, &this->cred->set);
        charon->backends->remove_backend(charon->backends, &this->config->
            backend);
@@ -722,14 +800,22 @@ stroke_socket_t *stroke_socket_create()
        this->control = stroke_control_create();
        this->list = stroke_list_create(this->attribute);

+       this->mutex = mutex_create(MUTEX_TYPE_DEFAULT);
+       this->condvar = condvar_create(CONDVAR_TYPE_DEFAULT);
+       this->commands = linked_list_create();
+
        lib->credmgr->add_set(lib->credmgr, &this->ca->set);
        lib->credmgr->add_set(lib->credmgr, &this->cred->set);
        charon->backends->add_backend(charon->backends, &this->config->
            backend);
        hydra->attributes->add_provider(hydra->attributes, &this->
            attribute->provider);

-       this->job = callback_job_create_with_prio((callback_job_cb_t)
    receive,
+       this->receiver = callback_job_create_with_prio((callback_job_cb_t)
    receive,
+
    this, NULL, NULL, JOB_PRIO_CRITICAL);
+       lib->processor->queue_job(lib->processor, (job_t*)this->receiver);
+
+       this->handler = callback_job_create_with_prio((callback_job_cb_t)
    handle,
                                                                    this
                                                                        ,
                                                                        NULL
                                                                        ,
                                                                        NULL
                                                                        ,
                                                                        JOB_PRIO_CRITICAL
                                                                        )
                                                                        ;

-       lib->processor->queue_job(lib->processor, (job_t*)this->job);
+       lib->processor->queue_job(lib->processor, (job_t*)this->handler);

        return &this->public;
 }
```

Listing F.3: tobias-patch2.diff

```
--- a/src/libcharon/plugins/stroke/stroke_socket.c
+++ b/src/libcharon/plugins/stroke/stroke_socket.c
@@ -43,7 +43,7 @@
  * To avoid clogging the thread pool with (blocking) jobs, we limit the
     number
  * of concurrently handled stroke commands.
  */
-#define MAX_CONCURRENT 4
+#define MAX_CONCURRENT_DEFAULT 4

 typedef struct stroke_job_context_t stroke_job_context_t;
 typedef struct private_stroke_socket_t private_stroke_socket_t;
@@ -94,6 +94,11 @@ struct private_stroke_socket_t {
        u_int handling;

        /**
+        * the maximum number of concurrently handled commands
+        */
+       u_int max_concurrent;
+
+       /**
         * configuration backend
         */
        stroke_config_t *config;
@@ -662,7 +667,7 @@ static job_requeue_t handle(private_stroke_socket_t *
    this)
        thread_cleanup_push((thread_cleanup_t)this->mutex->unlock, this->
            mutex);
        oldstate = thread_cancelability(TRUE);
        while (this->commands->get_count(this->commands) == 0 ||
-                  this->handling >= MAX_CONCURRENT)
+                  this->handling >= this->max_concurrent)
        {
                this->condvar->wait(this->condvar, this->mutex);
        }
@@ -803,6 +808,8 @@ stroke_socket_t *stroke_socket_create()
        this->mutex = mutex_create(MUTEX_TYPE_DEFAULT);
        this->condvar = condvar_create(CONDVAR_TYPE_DEFAULT);
        this->commands = linked_list_create();
+       this->max_concurrent = lib->settings->get_int(lib->settings,
+                               "charon.plugins.stroke.max_concurrent",
    MAX_CONCURRENT_DEFAULT);

        lib->credmgr->add_set(lib->credmgr, &this->ca->set);
        lib->credmgr->add_set(lib->credmgr, &this->cred->set);
```

Bibliography

[1] Shiboleth. http://shibboleth.internet2.edu/ retrieved May 05, 2009.

[2] strongSwan, the OpenSource IPsec-based VPN Solution. Website. Available at http://strongswan.org/. Retrieved November 05, 2009. Last modified September 27, 2009.

[3] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz. Extensible Authentication Protocol (EAP). RFC 3748 (Proposed Standard), June 2004. Updated by RFC 5247.

[4] B. Aboba and D. Simon. PPP EAP TLS Authentication Protocol. RFC 2716 (Experimental), October 1999. Obsoleted by RFC 5216.

[5] B. Aboba, D. Simon, and P. Eronen. Extensible Authentication Protocol (EAP) Key Management Framework. RFC 5247 (Proposed Standard), August 2008.

[6] B. Aboba and J. Wood. Authentication, Authorization and Accounting (AAA) Transport Profile. RFC 3539 (Proposed Standard), June 2003.

[7] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307, London, UK, 1999. Springer-Verlag.

[8] Puri N. Anggraeni, Neeli R. Prasad, and R. Prasad. Secure personal network. In *PIMRC*, pages 1–5. IEEE, 2008.

[9] Claudio A. Ardagna, Marco Cremonini, Ernesto Damiani, Sabrina De Capitani di Vimercati, and Pierangela Samarati. Supporting location-based conditions in access control policies. In *ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, computer*

and communications security, pages 212–222, New York, NY, USA, 2006. ACM.

[10] J. Arkko and H. Haverinen. Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA). RFC 4187 (Informational), January 2006.

[11] Giuseppe Ateniese, Marina Blanton, and Jonathan Kirsch. Secret handshakes with dynamic and fuzzy matching. In *Network and Distributed System Security Symposuim*, pages 159–177. The Internet Society, February 2007.

[12] Dirk Balfanz, Glenn Durfee, Narendar Shankar, Diana Smetters, Jessica Staddon, and Hao-Chi Wong. Secret handshakes from pairing-based key agreements. In *SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy*, page 180, Washington, DC, USA, 2003. IEEE Computer Society.

[13] Romain Berrendonner and Herve Chabanne. Ppp eap make mutual authentication protocol, November 2001. draft-berrendo-chabanne-pppext-eapmake-01.txt.

[14] F. Bersani and H. Tschofenig. The EAP-PSK Protocol: A Pre-Shared Key Extensible Authentication Protocol (EAP) Method. RFC 4764 (Experimental), January 2007.

[15] Jan Willem Christiaan Beusink. Investigation and selection of a suitable security architecture that can be applied in federated personal networks. Deliverable of the course Research Topics, 2009.

[16] Wen bi Rao and Quan Gan. The performance analysis of two digital signature schemes based on secure charging protocol. In *Wireless Communications, Networking and Mobile Computing, 2005. Proceedings. 2005 International Conference on*, volume 2, pages 1180–1182, Sept. 2005.

[17] A. Bogdanov, L. R. Knudsen, G. Le, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. Present: An ultra-lightweight block cipher. In *the proceedings of CHES 2007*. Springer, 2007.

[18] Carolyn A. Brodie, Clare-Marie Karat, and John Karat. An empirical study of natural language parsing of privacy policy rules using the sparcle policy workbench. In *SOUPS '06: Proceedings of the second symposium on Usable privacy and security*, pages 8–19, New York, NY, USA, 2006. ACM.

[19] Tobias Brunner. Make number of concurrently handled stroke messages configurable. Website. Available at `http://git.strongswan.org/?p=strongswan.git;a=commitdiff;h=7c0c2349`. Retrieved January 02, 2012. Last modified December 29, 2011.

[20] Tobias Brunner. This avoids clogging the thread pool with potentially blocking jobs. Website. Available at `http://git.strongswan.org/?p=strongswan.git;a=commitdiff;h=8ff513a8`. Retrieved January 02, 2012. Last modified December 29, 2011.

[21] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko. Diameter Base Protocol. RFC 3588 (Proposed Standard), September 2003.

[22] N. Cam-Winget, D. McGrew, J. Salowey, and H. Zhou. The Flexible Authentication via Secure Tunneling Extensible Authentication Protocol Method (EAP-FAST). RFC 4851 (Informational), May 2007.

[23] Xiang Cao and Lee Iverson. Intentional access management: making access control usable for end-users. In *SOUPS '06: Proceedings of the second symposium on Usable privacy and security*, pages 20–31, New York, NY, USA, 2006. ACM.

[24] David Chadwick and Sean Anthony. Using WebDAV for Improved Certificate Revocation and Publication. In *LNCS 4582. Public Key Infrastructure, Proceedings of 4th European PKI Workshop, Palma de Mallorca, Spain*, pages 265–279, June 2007.

[25] David W Chadwick, Gansen Zhao, Sassa Otenko, Romain Laborde, Linying Su, and Tuan Anh Nguyen. PERMIS: a modular authorization infrastructure. *Concurrency and Computation: Practice and Experience*, 20(11):1341–1357, August 2008. Online ISSN: 1532-0634.

[26] P. Congdon, B. Aboba, A. Smith, G. Zorn, and J. Roese. IEEE 802.1X Remote Authentication Dial In User Service (RADIUS) Usage Guidelines. RFC 3580 (Informational), September 2003.

[27] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008.

[28] Lorrie Faith Cranor and Lawrence Lessig. *Web Privacy with P3P*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

[29] C. de Laat, G. Gross, L. Gommans, J. Vollbrecht, and D. Spence. Generic AAA Architecture. RFC 2903 (Experimental), August 2000.

[30] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard), January 1999. Obsoleted by RFC 4346, updated by RFC 3546.

[31] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346 (Proposed Standard), April 2006. Obsoleted by RFC 5246, updated by RFCs 4366, 4680, 4681.

[32] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008.

[33] L. Dusseault. HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV). RFC 4918 (Proposed Standard), June 2007.

[34] ECRYPT. Ecrypt final report on algorithms and key lengths (2008), July 2008. Yearly Report on Algorithms and Keysizes (2007-2008), D.SPA.28 Rev. 1.1, IST-2002-507932 ECRYPT.

[35] Stephan Eichler and Bernd Muller-Rathgeber. Performance analysis of scalable certificate revocation schemes for ad hoc networks. In *LCN '05: Proceedings of the The IEEE Conference on Local Computer Networks 30th Anniversary*, pages 382–391, Washington, DC, USA, 2005. IEEE Computer Society.

[36] P. Eronen. IKEv2 Mobility and Multihoming Protocol (MOBIKE). RFC 4555 (Proposed Standard), June 2006.

[37] P. Eronen and J. Korhonen. Multiple Authentication Exchanges in the Internet Key Exchange (IKEv2) Protocol. RFC 4739 (Experimental), November 2006.

[38] S. Cantor et al. Assertions and protocols for the oasis security assertion markup language (saml) v2.0. Standard 2.0, OASIS (the Organization for the Advancement of Structured Information Standards), March 2005. Obtainable from `http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf`.

[39] 3g security; access security for ip-based services. Technical Report 3GPP TS 33.203 version 8.5.0 release 8, European Telecommunications Standards Institute (ETSI), 2009.

[40] S. Farrell and R. Housley. An Internet Attribute Certificate Profile for Authorization. RFC 3281 (Proposed Standard), April 2002.

[41] David Ferraiolo, Janet Cugini, and Richard Kuhn. Role-based access control (rbac): Features and motivations. In *In Proceedings of 11th Annual Computer Security Application Conference*, pages 241–248, 1995.

[42] David Ferraiolo and Richard Kuhn. Role-based access control. In *In 15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[43] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFC 2817.

[44] E. R. Fledderus, O. D. Rietkerk, F. T. Hartog, I. G. Niemegeers, and S. M. Groot. Towards viable personal networks and fednets – a value-web perspective. *Wirel. Pers. Commun.*, 38(1):103–115, 2006.

[45] P. Funk and S. Blake-Wilson. Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0). RFC 5281 (Informational), August 2008.

[46] Paul Funk and Simon Blake-Wilson. Eap tunneled tls authentication protocol version 1 (eap-ttlsv1), March 2006. `http://tools.ietf.org/html/draft-funk-eap-ttls-v1-01`.

[47] Ed Gerck. Overview of certification systems: X.509, pkix, ca, pgp & skip. do you understand digital certificates? do you know what they warrant? published by the MCG 1997-2000, July 2000. at `http://www.thebell.net/papers/certover.pdf`.

[48] Vipul Goyal. Certificate revocation lists or online mechanisms. In Eduardo Fernández-Medina, Julio César Hernández Castro, and L. Javier García-Villalba, editors, *WOSIS*, pages 261–268. INSTICC Press, 2004.

[49] Vipul Goyal. Certificate revocation using fine grained certificate space partitioning. In Sven Dietrich and Rachna Dhamija, editors, *Financial Cryptography*, volume 4886 of *Lecture Notes in Computer Science*, pages 247–259. Springer, 2007.

[50] Yanying Gu, Weidong Lu, R. V. Prasad, and Ignas Niemegeers. Clustering in ad hoc personal network formation. In *ICCS '07: Proceedings of the 7th international conference on Computational Science, Part IV*, pages 312–319, Berlin, Heidelberg, 2007. Springer-Verlag.

[51] Vipul Gupta, Sumit Gupta, Sheueling Chang, and Douglas Stebila. Performance analysis of elliptic curve cryptography for ssl. In *WiSE '02: Proceedings of the 1st ACM workshop on Wireless security*, pages 87–94, New York, NY, USA, 2002. ACM.

[52] Jeroen Hoebeke, Gerry Holderbeke, Ingrid Moerman, Martin Jacobsson, Venkatesha Prasad, N.I. Cempaka Wangi, Ignas Niemegeers, and

Sonia Heemstra de Groot. Personal network federations. In *proceedings of the 15th IST Mobile & Wireless Communications Summit*, June 4-8, 2006.

[53] Extensible authentication protocol (eap) registry. Retrieved from `http://www.iana.org/assignments/eap-numbers` on the 28th of May 2009, at which time it was last updated 2009-05-22.

[54] Malohat Ibrohimovna and Sonia Heemstra de Groot. Sharing resources in group-oriented networks fednet and related paradigms. In *Mobile Ubiquitous Computing, Systems, Services and Technologies, 2008. UBICOMM '08. The Second International Conference on*, pages 430–437, Washington, DC, USA, 29 2008-Oct. 4 2008. IEEE Computer Society.

[55] Malohat Ibrohimovna and Sonia Heemstra de Groot. Proxy-based fednets for sharing personal services in distributed environments. In *Wireless and Mobile Communications, 2008. ICWMC '08. The Fourth International Conference on*, pages 150–157, 27 2008-Aug. 1 2008.

[56] Malohat Ibrohimovna, Sonia Heemstra de Groot, Jasper Goseling, Hartmut Benz, and Jan Stoter. *Federations of Personal Networks*. Freeband, v1.0 edition, 19 December 2008. Project reference: PNP2008 Delivarable A.2.5.

[57] Ieee standard for information technology – telecommunications and information exchange between systems – local and metropolitan area networks – specific requirements; part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications; amendment 6: Medium access control (mac) security enhancements. Technical report, June 2004.

[58] Ieee standard for local and metropolitan area networks – port-based network access control. *IEEE Std 802.1X-2004 (Revision of IEEE Std 802.1X-2001)*, pages 1–169, November 2004.

[59] ECRYPT II. Yearly report on algorithms and keysizes (2008-2009), July 2009. Yearly Report on Algorithms and Keysizes (2008-2009), D.SPA.7 Rev. 1.0, ICT-2007-216676 ECRYPT II, 07/2009.

[60] Philip Inglesant, M. Angela Sasse, David Chadwick, and Lei Lei Shi. Expressions of expertness: the virtuous circle of natural language for access control policy specification. In *SOUPS '08: Proceedings of the 4th symposium on Usable privacy and security*, pages 77–88, New York, NY, USA, 2008. ACM.

[61] IOP GenCom. *VITRUVIUS – Versatile Interface for TRUstworthy VItal User (oriented) Services*, April 2008. projectplan.

[62] International Telecommunication Union (ITU). Security architecture for open systems interconnection for ccitt applications, 1991. ITU-T Recommendation X.800/ISO 7498-2, obtainable from http://www.itu.int/rec/T-REC-X.800/en.

[63] ITU-T. Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). Recommendation X.690, International Telecommunication Union, June 1994.

[64] David P. Jablon. Strong password-only authenticated key exchange. *SIGCOMM Comput. Commun. Rev.*, 26:5–26, October 1996.

[65] Martin Jacobsson, Weidong Lu, Anthony Lo, Venkatesha Prasad, Malohat Kamilova, Ertan Onur, Edsger Jager, Rieks Joosten, Martin Werf, van der, Ben Hillen, Sonia Heemstra de Groot, Jan Stoter, Neill Whillians, and Jasper Goseling. *PN Architectures Final Version*. Freeband, v1.0 edition, December 2008. Project reference: PNP2008 Delivarable A.1.3.

[66] Martin E. Jacobsson. *Personal Networks – An Architecture for Self-Organized Personal Wireless Communications*. PhD thesis, Delft University of Technology, 2008. available on http://www.wmc.ewi.tudelft.nl/~martin/thesis/.

[67] Usman Javaid, Djamal-Eddine Meddour, Tinku Mohamed Rasheed, and Toufik Ahmed. Personal network routing protocol (pnrp) for personal ubiquitous environments. In *ICC*, pages 3100–3107. IEEE, 2007.

[68] Assed Jehangir and Sonia M. Heemstra de Groot. A security architecture for personal networks. *Mobile and Ubiquitous Systems - Workshops, 2006. 3rd Annual International Conference on*, pages 1–8, July 2006.

[69] Assed Jehangir and Sonia M. Heemstra de Groot. Securing personal network clusters. In *Security and Privacy in Communications Networks and the Workshops, 2007. SecureComm 2007. Third International Conference on*, pages 320–329, Sept. 2007.

[70] Xiaodong Jiang and James A. Landay. Modeling privacy control in context-aware systems. *IEEE Pervasive Computing*, 1(3):59–63, 2002.

[71] Jon_Senior. Jamvm + gnu classpath on kamikaze (+ wiki question). Webforum. Available at https://forum.openwrt.org/viewtopic.php?pid=71500#p71500. Retrieved November 30, 2010. Last modified July 21, 2008.

[72] S. Josefsson. Extended Kerberos Version 5 Key Distribution Center (KDC) Exchanges over TCP. RFC 5021 (Proposed Standard), August 2007.

[73] C. Kaufman. Internet Key Exchange (IKEv2) Protocol. RFC 4306 (Proposed Standard), December 2005. Updated by RFC 5282.

[74] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[75] D. M. Kyriazanos, H. Olesen, A. Hammershøj, E. Heinze, S. Bessler, J. Zeiss, C. Patrikakis, G. Nikolakopoulos, S. Amundsen, H. Thuvesson, A. Cimmino, P. Novelli, R. Olsen, N. Prasad, M. Bauer, F. Armknecht, A. Pashalidis, K. Masmoudi, M. Girod Genet, and I. Moerman. Specification of user profile, identity and role management for PNs and integration to the PN platform. Technical report, EU FP6 IST Project My Personal Adaptive Global Net (MAGNET) Beyond, March 2007. Document number: MAGNET/B/WP4.3/NTUA/D4.3.2/R/PU/001/28.03.2007.

[76] L. Law and J. Solinas. Suite B Cryptographic Suites for IPsec. RFC 4869 (Informational), May 2007.

[77] List of ldap software. Wikipedia online encyclopedia. Available at from `http://en.wikipedia.org/wiki/List_of_LDAP_software`, last modified on 7 June 2009, retrieved June 22, 2009.

[78] Jun Lei, Xiaoming Fu, Dieter Hogrefe, and Jianrong Tan. Comparative studies on authentication and key exchange methods for 802.11 wireless lan. *Computers & Security*, 26(5):401–409, 2007.

[79] A. Levi and E. Savas. Performance evaluation of public-key cryptosystem operations in wtls protocol. In *Computers and Communication, 2003. (ISCC 2003). Proceedings. Eighth IEEE International Symposium on*, pages 1245–1250 vol.2, June-3 July 2003.

[80] Albert Levi and Erkay Savas. Performance evaluation of public-key cryptosystem operations in wtls protocol. In *ISCC '03: Proceedings of the Eighth IEEE International Symposium on Computers and Communications*, page 1245, Washington, DC, USA, 2003. IEEE Computer Society.

[81] Gabriel López, Oscar Cánovas, Antonio F. Gómez, Jesús D. Jiménez, and Rafael Marín. A network access control approach based on the aaa architecture and authorization attributes. *J. Netw. Comput. Appl.*, 30(3):900–919, 2007.

[82] Magnet, my personal adaptive global net. http://www.telecom.ece.ntua.gr/magnet/index.html retrieved March 13, 2009.

[83] Magnet, my personal adaptive global net. http://magnet.aau.dk retrieved 13-3-09.

[84] Khamish Malhotra, Stephen Gardner, and Will Mepham. A novel implementation of signature, encryption and authentication (sea) protocol on mobile patient monitoring devices. *Technol. Health Care*, 16(4):261–272, 2008.

[85] Cameron McDonald, Philip Hawkes, and Josef Pieprzyk. Differential path for sha-1 with complexity $o(2^{52})$. Cryptology ePrint Archive, Report 2009/259, 2009. http://eprint.iacr.org/.

[86] Silvio Micali. Efficient certificate revocation. Technical report, Cambridge, MA, USA, 1996.

[87] Marco Casassa Mont and Robert Thyne. A systemic approach to automate privacy policy enforcement in enterprises. In George Danezis and Philippe Golle, editors, *Privacy Enhancing Technologies*, volume 4258 of *Lecture Notes in Computer Science*, pages 118–134. Springer, 2006.

[88] R L B Morgan. Federated security: The shibboleth approach. *EDUCAUSE Quarterly*, (2004):12–17, 2004.

[89] Ms-peap: Protected extensible authentication protocol (peap) specification. Technical report, Microsoft Corporation, May 2009.

[90] Darren Mundy and David W. Chadwick. An xml alternative for performance and security: Asn.1. *IT Professional*, 6(1):30–36, 2004.

[91] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 2560 (Proposed Standard), June 1999.

[92] Nanoeap™ – mocana's open standards based, full featured rfc compliant embedded eap solution. Product data sheet, February 2009. Retrieved from http://www.mocana.com/NanoEAP.html on July 06, 2009.

[93] National Institute of Standards and Technology. *SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES*, May 2001. FIPS PUB 140-2.

[94] National Institute of Standards and Technology. *Recommendation for Key Management – Part 1: General (Revised)*, March 2007. FIPS PUB Special Publication 800-57, part 1.

[95] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos Network Authentication Service (V5). RFC 4120 (Proposed Standard), July 2005. Updated by RFCs 4537, 5021.

[96] I. G. Niemegeers and S. M. Groot. Fednets: Context-aware ad-hoc network federations. *Wirel. Pers. Commun.*, 33(3-4):305–318, 2005.

[97] I. G. Niemegeers and S. M. Heemstra De Groot. From personal area networks to personal networks: A user oriented approach. *Wirel. Pers. Commun.*, 22(2):175–186, 2002.

[98] Y. Nir. Repeated Authentication in Internet Key Exchange (IKEv2) Protocol. RFC 4478 (Experimental), April 2006.

[99] NSA suite B cryptography. Website. Available at `http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml`. Retrieved Jun 04, 2009. Last modified Jan 15, 2009.

[100] Protected eap protocol (peap) version 2. IETF internet-draft. available at `http://tools.ietf.org/html/draft-josefsson-pppext-eap-tls-eap-10`.

[101] PERMIS. Website. Available at `http://sec.cs.kent.ac.uk/permis/`. Retrieved November 06, 2009. Last modified Feb 19, 2007.

[102] Personal network pilot 2008. `http://www.freeband.nl/`.

[103] Dubravko Priselac and Miljenko Mikuc. Security risks of pre-ims aka access security solutions, 2008. retrieved from `http://www.ericsson.com/hr/about/events/mipro_2008/papers.shtml`.

[104] J. Quittek, T. Zseby, B. Claise, and S. Zander. Requirements for IP Flow Information Export (IPFIX). RFC 3917 (Informational), October 2004.

[105] James Randall and Michael Szydlo. Collisions for sha0, md5, haval, md4, and ripemd, but sha1 still secure, August 2004.

[106] Maxim Raya, Daniel Jungels, Panos Papadimitratos, Imad Aad, and Jean-Pierre Hubaux. Certificate Revocation in Vehicular Networks. Technical report, Laboratory for computer Communications and Applications, 2006.

[107] C. Rigney. RADIUS Accounting. RFC 2866 (Informational), June 2000. Updated by RFCs 2867, 5080.

[108] C. Rigney, S. Willens, A. Rubens, and W. Simpson. Remote Authentication Dial In User Service (RADIUS). RFC 2865 (Draft Standard), June 2000. Updated by RFCs 2868, 3575, 5080.

[109] Ronald Rivest. Can we eliminate certificate revocation lists? In *In Financial Cryptography*, pages 178–183. Springer-Verlag, 1998.

[110] Konrad Roeder. Obtained from `http://en.wikipedia.org/wiki/RADIUS` under the Creative Commons Attribution-Share Alike 3.0 Unported License.

[111] J. Rosenberg, R. Mahy, and J. Mathews. Traversal using relays around nat (turn): Relay extensions to session traversal utilities for nat (stun), February 2009.

[112] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393.

[113] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489 (Proposed Standard), March 2003. Obsoleted by RFC 5389.

[114] PPP EAP RSA Public Key Authentication Protocol. Intellectual Property Rigths declaration. available at `https://datatracker.ietf.org/ipr/15` retrieved June 09, 2009.

[115] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[116] J. Sermersheim. Lightweight Directory Access Protocol (LDAP): The Protocol. RFC 4511 (Proposed Standard), June 2006.

[117] D. Simon, B. Aboba, and R. Hurst. The EAP-TLS Authentication Protocol. RFC 5216 (Proposed Standard), March 2008.

[118] W. Simpson. The Point-to-Point Protocol (PPP). RFC 1661 (Standard), July 1994. Updated by RFC 2153.

[119] A. Smailagic and D. Kogan. Location sensing and privacy in a context-aware computing environment. *Wireless Communications, IEEE*, 9(5):10–17, Oct. 2002.

[120] Asim Smailagic, Daniel P. Siewiorek, Joshua Anhalt, David Kogan, and Yang Wang. Location sensing and privacy in a context aware computing environment. *IEEE Wireless Communications*, 9:10–17, 2001.

[121] William Stallings. *Network Security Essentials: Applications and Standards*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 3rd edition, 2006.

[122] Antonietta Stango, Neeli Prasad, and Jordi Jaen Pallares. Analysis, verification and evaluation. Technical report, EU FP6 IST Project My Personal Adaptive Global Net (MAGNET) Beyond, July 2008. Document number: MAGNET_B_WP4_D4_4_2_PU_R_1.0.doc.

[123] D. Stanley, J. Walker, and B. Aboba. Extensible Authentication Protocol (EAP) Method Requirements for Wireless LANs. RFC 4017 (Informational), March 2005.

[124] Andreas Steffen and Tobias Brunner. strongswan.conf. Website. Available at `http://wiki.strongswan.org/projects/strongswan/wiki/StrongswanConf`. Retrieved November 30, 2010. Last modified October 11, 2010.

[125] Andreas Steffen, Martin Willi, and Tobias Brunner. Autoconf options for strongSwan 4.3 releases. Website. Available at `http://wiki.strongswan.org/projects/strongswan/wiki/Autoconf`. Retrieved October 11, 2010. Last modified September 02, 2010.

[126] E. Stokes, D. Byrne, B. Blakley, and P. Behera. Access Control Requirements for LDAP. RFC 2820 (Informational), May 2000.

[127] Clifford Stoll. Stalking the wily hacker. *Commun. ACM*, 31(5):484–497, 1988.

[128] Lars Strand. Obtained from `http://en.wikipedia.org/wiki/802.1x` under the terms of the GNU Free Documentation License.

[129] Falko Timme. How To Set Up WebDAV With Apache2 On Ubuntu 9.04. Website. Available at `http://www.howtoforge.com/how-to-set-up-webdav-with-apache2-on-ubuntu-9.04`. Retrieved October 11, 2010. Last modified October 19, 2010.

[130] P. Trakadas, T. Zahariadis, H.C. Leligou, S. Voliotis, and K. Papadopoulos. Analyzing energy and time overhead of security mechanisms in wireless sensor networks. In *Systems, Signals and Image Processing, 2008. IWSSIP 2008. 15th International Conference on*, pages 137–140, June 2008.

[131] Axel Tressel and Jorg Keller. A system for secure ip telephone conferences. In *NCA '06: Proceedings of the Fifth IEEE International Symposium on Network Computing and Applications*, pages 231–234, Washington, DC, USA, 2006. IEEE Computer Society.

[132] H. Tschofenig, D. Kroeselberg, A. Pashalidis, Y. Ohba, and F. Bersani. The Extensible Authentication Protocol-Internet Key Exchange Protocol version 2 (EAP-IKEv2) Method. RFC 5106 (Experimental), February 2008.

[133] Vitruvius project. `www.vitruvius-project.com` retrieved February 15, 2009.

[134] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence. AAA Authorization Framework. RFC 2904 (Informational), August 2000.

[135] M. Wahl, A. Coulbeck, T. Howes, and S. Kille. Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions. RFC 2252 (Proposed Standard), December 1997. Obsoleted by RFCs 4510, 4517, 4523, 4512, updated by RFC 3377.

[136] Arvinderpal S. Wander, Nils Gura, Hans Eberle, Vipul Gupta, and Sheueling Chang Shantz. Energy analysis of public-key cryptography for wireless sensor networks. In *PERCOM '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, pages 324–328, Washington, DC, USA, 2005. IEEE Computer Society.

[137] Haodong Wang, Bo Sheng, and Qun Li. Elliptic curve cryptography-based access control in sensor networks. *Int. J. Secur. Netw.*, 1(3/4):127–137, 2006.

[138] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full sha-1. In *Advances in Cryptology – CRYPTO 2005*, volume 3621/2005 of *Lecture Notes in Computer Science*, pages 17–36. Springer Berlin / Heidelberg, 2005.

[139] N.I.C. Wangi, R.V. Prasad, I. Niemegeers, and S.H. de Groot. Ad hoc federation of networks (fednets) - mechanisms and requirements. *Communication Systems Software and Middleware, 2007. COMSWARE 2007. 2nd International Conference on*, pages 1–6, January 2007.

[140] Report on war driving survey 2008, February 2009. obtainable at `http://www.safewifi.hk/security.html`.

[141] William T. Whelan. Ppp eap rsa public key authentication protocol, February 1997. retrieved from `http://tools.ietf.org/html/draft-ietf-pppext-eaprsa-04`.

[142] Petra Wohlmacher. Digital certificates: a survey of revocation methods. In *MULTIMEDIA '00: Proceedings of the 2000 ACM workshops on Multimedia*, pages 111–114, New York, NY, USA, 2000. ACM.

[143] Timothy Wright, Pubudu Chandrasiri, Ozgur Gurleyen, Micael da Costa, Christian Gehrmann, András Méhes, Stefan Goeman, Thomas Kuhn, Christian Günter, Kaisa Nyberg, Sampo Sovio, and Peter Windirsch. *Final technical report – Specification of a security architecture for distributed terminals.* Information Society Technologies, v1.0 edition, November 2002.

[144] Hanbing Yao, Heping Hu, Baohua Huang, and Ruixuan Li. Dynamic role and context-based access control for grid applications. In *PDCAT '05: Proceedings of the Sixth International Conference on Parallel and Distributed Computing Applications and Technologies*, pages 404–406, Washington, DC, USA, 2005. IEEE Computer Society.

[145] Yuanyuan Zhang and Dawu Gu. Performance analysis of pair-wise key establishment protocols for sensor networks. In *Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on*, pages 2637–2641, Sept. 2007.

[146] Peifang Zheng. Tradeoffs in certificate revocation schemes. *SIGCOMM Comput. Commun. Rev.*, 33(2):103–112, 2003.

[147] L. Zhu, K. Jaganathan, and K. Lauter. Elliptic Curve Cryptography (ECC) Support for Public Key Cryptography for Initial Authentication in Kerberos (PKINIT). RFC 5349 (Informational), September 2008.

[148] L. Zhu, P. Leach, and K. Jaganathan. Kerberos Cryptosystem Negotiation Extension. RFC 4537 (Proposed Standard), June 2006.