NTNU
Innovation and Creativity

Using SIM for strong end-to-end Application Authentication

Lars Lunde
Audun Wangensteen

Master of Science in Communication Technology
Submission date: May 2006
Supervisor: Van Thanh Do, ITEM
Co-supervisor: Ivar Jørstad, ITEM

Norwegian University of Science and Technology
Department of Telematics

Problem Description

Till now, the most dominating applications on the Internet and the World Wide Web are either communication ones like mail, chat, IP telephony, etc. or information ones like Google, Yahoo, etc. The commercial and governmental applications have started to emerge but met security issues. These applications require strong authentication at the same time as privacy is preserved. The SIM card in the mobile phone is a smart card, a tamper-resistant device containing strong authentication mechanisms, which is most widely used due to high penetration of mobile phones. It is therefore very convenient and cost-efficient if the SIM card authentication can also be extended to be used for other applications. This thesis will build on the work previously done in two student projects, where general architectures utilizing SIM-based authentication were proposed.

The thesis work consists of the following tasks:
-Design of a strong SIM-based authentication system for end-user applications
-Implementation
-Testing
-The system consists of several components on heterogeneous platforms and the work will be distributed equally between the students.

Assignment given: 2006-01-16
Supervisor: Van Thanh Do, ITEM

Preface

This thesis is submitted to the Norwegian University of Science and Technology (NTNU) to fulfill the requirements for the Master of Technology degree. The thesis was carried out at Telenor R&D Fornebu in collaboration with Department of Telematics, NTNU.

The supervisor for this project has been PhD candidate Ivar Jørstad and the academic responsible has been professor Do van Thanh. We would like to thank them both for help and suggestions during this project and especially Ivar Jørstad who has been available for answering our questions at all times. We also would like to thank Telenor R&D for supporting us with Bluetooth dongles, phones, etc. for testing purposes.

Fornebu, May 30, 2006.

Lars Lunde Audun Wangensteen

Contents

List of Figures

List of Tables

Abbreviations

AAA	Authentication, Authorization and Accounting
AID	Application Identifier
AKA	Authentication and Key Agreement
APDU	Application Protocol Data Unit
API	Application Programming Interface
AT	Attention; Hayes Command Set (AT-Commands)
AuC	Authentication Center
BSS	Base Station Subsystem
BTS	Base Transceiver Station
CA	Certification Authority
CAD	Card Acceptance Device
CHV	Card Holder Verification
CLA	Class of instruction
CPU	Central Processing Unit
CSRC	Computer Security Resource Center
DF	Dedicated File; SIM
EAP	Extensible Authentication Protocol
EAP-AKA	EAP Method for 3. Generation AKA
EAPoL	EAP over LAN; often mentioned in same context as 802.1X
EAP-SIM	EAP Method for GSM SIM
EDR	Enhanced Data Rate
EEPROM	Electrically Erasable Programmable Read-Only Memory
EF	Elementary File; SIM
EIR	Equipment Identity Register
EMV	Europay, MasterCard, and Visa
FIPS	Federal Information Processing Standards
GAS	Generic SIM Authentication System
GPL	General Public License
GPRS	General Packet Radio Service
GSM	Global System for Mobile communication
HMAC	Keyed-Hash Message Authentication Code
HSS	Home Subscriber Service
HTTP	HyperText Transfer Protocol
HTTPS	Secure HTTP
ICC	Integrated Circuits Cards
IETF	The Internet Engineering Task Force
IMSI	International Mobile Subscriber Identity
INS	Instruction code
IP	Internet Protocol
IPSec	Internet Protocol Security
IS	Information System
IT	Information Technology
ITU	International Telecommunication Union
J2SE	Java 2 Platform, Standard Edition; from version 6 it will be called Java SE
JAR	Java ARchive
Java ME	Java Platform, Micro Edition; Formerly referred to as J2ME
JCRE	Java Card Runtime Environment
JSP	JavaServer Pages

JSR	Java Specification Requests
JVM	Java Virtual Machine
Kc	64-bits cipher key
Ki	128-bits authentication key
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
MAC	Message Authentication Code; used in cryptography
MAC	Media Access Control; address of a networking device
MD-5	Message Digest no. 5
ME	Mobile Equipment (mobile phone in GSM without SIM)
MF	Master File; SIM
MNC	Mobile Network Code
MNO	Mobile Network Operator
MS	Mobile Station (mobile phone in GSM with SIM)
MSC	Mobile services Switching Centre
NAS	Network Access Server
NFC	Near Field Communication
NIST	National Institute of Standards and Technology
NSS	Network Sub-System
NTNU	Norwegian University of Science and Technology
OCF	OpenCard Framework
OTP	One-Time Password
PC/SC	Personal Computer/Smart Card
PCT	Private Communication Technology
PDA	Personal Digital Assistant
PHP	PHP: Hypertext Preprocessor
PIN	Personal Identification Number
PKI	Public-Key Infrastructure
PPP	Point-to-Point Protocol
PSTN	Public Switched Telephone Network
PUK	PIN UnlocK
RA	Registration Authority
RADIUS	Remote Authentication Dial-In User Service; a AAA server
RAM	Random Access Memory
RAND	Random challenge number (128-bits)
RFID	Radio Frequency Identity
ROM	Read-Only Memory
SAP	SIM Access Profile; a Bluetooth profile
SAT	SIM Application Toolkit
SATSA	Security and Trust Services API
SCTP	Stream Control Transmission Protocol
SGSN	Serving GPRS Support Node
SHA-1	Secure Hash Algorithm no. 1
SIG	Special Interest Group
SIM	Subscriber Identity Module
SIP	Session Initiation Protocol
SMS	Short Messaging Service
SOAP	Simple Object Access Protocol
SP	Service Provider
SPP	Serial Port Profile; Bluetooth profile

SQN	Sequence Number
SRES	Signed RESponse (32 bits)
SS7	Signal System no. 7
SSL	Secure Socket Layer
SSO	Single Sign-On
SW	Status Word
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TMSI	Temporary Mobile Subscriber Identity
UDP	User Datagram Protocol
UE	User Equipment; analogous to MS in GSM
UICC	Universal Integrated Circuit Card
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunications System
USAT	USIM Application Toolkit
USB	Universal Serial Bus
USIM	Universal Subscriber Identity Module
VLR	Visitor Location Register
VPN	Virtual Private Network
WLAN	Wireless Local Area Network
WWW	World Wide Web
XML	eXtensible Markup Language

Terminology

Authentication	Authentication is any process, through which one proves and verifies certain information, i.e. determining whether someone or something is, in fact, who or what it is declared to be. Authentication requires that the information be checked for a single, previously identified, entity.
Authentication Server	An entity that provides an authentication service to an Authenticator. In GAS, it is the Authentication Server that authenticates the Supplicants on behalf of the Authenticator.
Authenticator	An Authenticator is an entity that requires authentication from the Supplicant. In GAS, the main functionality of the Authenticator is to locate a suitable Authentication Server, act as a translator from EAP to RADIUS and back, store information about authorized Supplicants, and keep track of identities when for instance temporary identities are used.
Authorization	Authorization is the process of giving someone permission to do or have something.
Client	A client is a system that accesses a (remote) service on another computer by some kind of network.
Credentials	A credential is defined as an object that is verified when presented to the verifier in an authentication transaction.
GSM Gateway	The GSM Gateway connects IP network and SS7 network. The Gateway communicates with the GSM Network using MTP3 over SS7.
Identification	Identification is a process through which one ascertains the identity of another person or entity. Identification must uniquely identify a given entity.
Java Applet	Java Applets are Java applications that run in a Web browser using a Java virtual machine
JNI	Java Native Interface; allows Java code to communicate with native applications and libraries written in other languages.
Mutual Authentication	The two communication entities are authenticated to each other.
Peer	Peer is any node able to initiate or complete any supported transaction. Peer nodes may differ in local configuration, processing speed, network bandwidth, and storage quantity.
RADIUS	RADIUS is a standardized protocol for AAA that is being used by the majority of remote dial-in users, routers, and VPNs to centralize network authentication of remote accesses.
Server	A computer software application that carries out some task (i.e. provides a service) on behalf of yet another piece of software called a client.
Service Provider	A service provider is an entity that provides services to other entities. Usually this refers to a business that provides subscription or metered service to other businesses or individuals. Examples of these services include Internet access, Mobile phone service, and web application hosting.
SIM	The SIM is a logical module that runs on an Integrated Circuit Card (ICC) type of Smart Card. The SIM is a tamper resistant device, which is used to store keys to authenticate mobile users, messages, and phone numbers.

Spoofing	A technique used to send messages to a computer with an IP address indicating that the message is coming from a trusted source. The man-in-the-middle attack is an example of a network attack using spoofing. The attacker sniffs on the link between two end points, and pretends to be one end of the connection by replying the messages.
Strong Authentication	Strong Authentication is a process controlling the authenticity of the users identity on the basis of at least two of the three following factors: Something you know, Something you have or Something you are.
Supplicant	A Supplicant is an entity that is being authenticated by an Authenticator. In GAS, the Supplicant is split in two; the SIM Supplicant and the Service Supplicant. The SIM Supplicant communicates with the SIM on behalf of the Service Supplicant. The Service Supplicant is the one sending messages to/from the Authenticator and handles the different authentication protocols.
Web Service	Web Service is a software system designed to support interoperable machine-to-machine interaction over a network.

Abstract

Today the Internet is mostly used for services that require low or none security. The commercial and governmental applications have started to emerge but met problems since they require strong authentication, which is both difficult and costly to realize. The SIM card used in mobile phones is a tamper resistant device that contains strong authentication mechanisms. It would be very convenient and cost-efficient if Internet services could use authentication methods based on the SIM.

This master thesis presents an analysis and a design of a generic authentication system based on SIM, together with a detailed description of an implemented prototype. The proposed system, called the Generic SIM Authentication System (GAS), provides a strong authentication mechanism. The GAS builds upon the existing GSM authentication infrastructure, thus allows re-use of GSM expertise from the mobile operators. New services can easily be supported, such that these can benefit from strong authentication. By gradually implementing more authentication mechanisms (e.g. OTP and PKI) on the SIM, it will be able to support several levels of security. This will result in a generic authentication system satisfying the security needs for nowadays and also for the future.

In order to design the GAS, the thesis starts by giving an overview of authentication and relevant technologies, before the requirements to the system, both functional and non-functional, are defined. Then different interaction diagrams, collaboration diagrams and sequence diagrams are presented, and the necessary components and interfaces in the system are outlined. This thesis builds on two student projects finished December 2005, where tentative high-level architectures for utilizing SIM-based authentication were proposed.

A Prototype has been developed in Java to demonstrate the GAS, and includes both a client (Supplicant) and a server (Authenticator) part. The communication between the Supplicant and the other components in the authentication system is based on EAP, which is a general authentication protocol supporting multiple authentication methods. When performing the GSM authentication the EAP-SIM protocol is used. The Prototype has been tested end-to-end, i.e. from the SIM to the Telenor GSM HLR/AuC, via IP-based network.

Three different services have been developed to demonstrate how easily the SIM authentication can be integrated. The first demo service shows how to integrate the authentication with JSP technology and Apache Tomcat. The second service, MyService, is another example of how the authentication service could be integrated into a web portal using PHP to demonstrate that the Prototype is independent of the service implementation language. MyService also illustrates how the service provider can control the registration of new users and link up with their SIM identity. The last service, GasSpot, shows how to integrate the GAS to authenticate users to a Captive Portal. The access is controlled by the gateway, which is implemented using ChilliSpot.

Based on the results of the master thesis, the authors have written the paper “A Generic Authentication System based on SIM”, which has been submitted and accepted for publication at the ICISP’06 Conference in Cap Esterel, Côte d’Azur, France, August 26-29, 2006.

1 Introduction

The emergence and growth of Internet usage has led to the need for security and enhanced privacy. With an increasing number of service providers requiring authentication of different strength, it becomes difficult for the user to manage all these identities. The number of user identities should therefore be kept at a minimum to accomplish user simplicity. Throughout this thesis, different protocols, interfaces, and security problems during an end-to-end authentication will be investigated to find a solution reducing the number of user identities.

The investigation is based on SIM usage and existing technologies. The idea is to combine existing technologies to meet the security requirements of today and at the same time keep the system user-friendly.

The goal of this thesis is to investigate and propose a solution allowing the usage of SIM authentication for a wide range of applications. This thesis will build on the work previously done in two student projects, where general architectures utilizing SIM-based authentication were proposed.

The thesis is done in cooperation with Telenor R&D at Fornebu.

1.1 Motivation

Today the Internet is mostly used for services that require low or none security. The commercial and governmental applications have started to emerge but met problems since they require strong authentication, which is both difficult and costly to realize.

"As demonstrated by the Entrust Internet Security Survey, the proliferation of online hazards such as ID theft and phishing scams is threatening the growth in e-commerce and Internet use that we have experienced over the past decade. Corporations and other organizations that offer online services to consumers are at risk of experiencing financial and reputation losses unless they act quickly to protect customers' identity"
Bill Conner, chairman and CEO of Entrust, Inc. [1]

The use of smart cards has proven to be a good solution to the authentication problem, but it requires the users to obtain new type of devices, the Smart Card and the Reader. The SIM card in the mobile phone is also a smart card, and if applications could use the strong authentication of the SIM card it would be very convenient and cost-efficient.

The GSM system has already many users and the administration procedures are well defined. The use of the SIM card will also give the user a much easier interface; he/she will only need one card and one account to use for many types of applications. SIM-based WLAN authentication is a good example of a new type of application that is starting to emerge.

The main motivational factors for using the SIM as an authentication token are:

1. Large existing customer base
2. Simple deployment and administration of tokens
3. User friendliness (uses an existing device (the mobile phone with a SIM))
4. Secure and tamper-resistant (same as smart cards)

1.2 Problem Definition

The main problem statement in this master thesis has been:

1. *How can the SIM be used for authentication of users towards a wide range of IP-based services?*

In addition, the following sub-statements have been investigated:

a) How can such services exchange credentials with the SIM?
b) How can such services communicate with an authentication server (where credentials are stored)?
c) How can an authentication server exchange credentials with the HLR/AuC?
d) How can a strong SIM-based authentication system be designed?
e) How can the SIM-based authentication system be implemented to a Prototype?
f) Which experiences can be learned from this Prototype?

1.3 Challenges

Wireless LANs are emerging all over the world at airports, cafés, hotels etc, and people are bringing their laptops with them. Operators should be able to earn money on this. Critical to success is the operators' ability to deliver low-cost and secure services. The most inexpensive way to do this is to use existing infrastructure and make sure they have:

- Authentication and accounting of customers for billing purposes
- Roaming between networks, including seamless handover between wireless access points
- Cooperation between different identity providers/countries

At the same time, generalization is important to be able to use the authentication method for other applications and services. It is in this context the SIM can be used, and especially if located within a mobile phone. But there are some challenges using SIM in an authentication procedure:

- Difficulties with exploiting SIM security mechanism because the SIM is operator property
- Access to SIM is limited
- The original credentials are stored in the operator network. How is it possible to communicate with the operator's network?
- Difficult to add new security mechanisms or additional applets (e.g. for generating OTP and handle PKI)

It will be a challenge to find a way to extract credentials from the SIM (especially via the mobile phone) and be able to use them in the authentication process.

1.4 Objectives

The primary objective in this master thesis is to find a solution where the SIM can be used for application authentication. Although there have been developed several solutions using SIM as an authentication mechanism for accessing WLAN, there are no general solution to the problem. The second objective is to be able to use the mobile phone to retrieve SIM credentials. This is important to make it user-friendly enough to conquer smart cards.

The task emphasizes on different aspects of Internet and Telecom architectures to achieve end-to-end authentication. This requires a study of the basic concepts of authentication technology, everything in relation with the SIM and its network, and communication protocols, e.g. between the mobile phone and computers.

Related technologies are studied to get a more complete understanding of how the different aspects of the architectures that is going to be designed could be fulfilled. In this context, related work on using SIM to authenticate users in Public WLAN is essential. The different solutions found in related work will be torn apart and analysed to find usable parts. These parts and more will be studied in a background chapter.

The next step is to analyse the problem by identifying use cases and requirements, and make interaction diagrams to complete a thorough analysis phase. The analysis will be used to come up with the requisite components and class diagrams in a design phase.

The summation of the background, analysis, and design will be implemented in a Prototype. The last step will be to have a discussion regarding the technological and security aspects of the experiences made from making the Prototype and the study towards the Prototype. This is to try to find potential weaknesses and alternatives for further work and similar solutions

1.5 Related Work

The use of SIM for authentication is starting to emerge, but most of the solutions developed so far are product specific and not scalable. One competitor to the SIM authentication is smart card authentication system. This section will briefly review some of the solutions that are available today.

Today there exist several companies that provide solutions for SIM-based WLAN authentication. Axalto is one of these companies; they provide a SIM card and an USB-dongle for connection to the computer. The SIM card incorporates the EAP-SIM [2] authentication protocol (discussed in Section 2.5.1) and the provided middleware ensures mutually secure authentication. Axalto also provides an additional PKI solution for the SIM card, as well as a One-Time-Password (OTP) solution. Gemplus is another company that offers a similar product, the GEMobileIT [3]. This product offers an additional VPN (Virtual Private Network) service specially intended for corporate users, the SIM card stores the certificates necessary for performing the VPN authentication. Once the user has gained access to the Internet the VPN can be launched automatically. Gemplus and Axalto have just merged into a new company called Gemalto.

There have also been done some research on the use of GSM to enhance the e-commerce security. An article [4] published in connection with WMC'02 proposed a payment protocol that utilizes the GSM system for user authentication. Other GSM-based payment systems either use the SMS messaging, which asks the e-consumers to open an e-wallet, or require the users to make or receive phone calls using a GSM phone.

The use of the GSM system for Single Sign-On (SSO) authentication is also a hot topic today. Ubilogin[1] is a commercial solution that supports authentication through GSM SMS messaging. In [5] is a proposal from the Information Security Group Royal Holloway, an SSO protocol using the SIM for authentication. This solution invents a new protocol and does

[1] Ubisecure Ubilogin, http://ubisecure.com/

not rely on existing protocols such as the Extensible Authentication Protocol (EAP); neither does it provide different security levels of authentication.

One competitor to the generic SIM authentication system is BankID[2], which is a standardized and coordinated electronic identification system developed by Bankenes BetalingsSentral (BBS) on request by the Norwegian banks. The system is based on a public key infrastructure (PKI) and can be used on the Internet for identification and digital signatures. To use the system the user needs a smart card and a smart card reader (offline). The BankID can be used as a standardized authentication mechanism for Norwegian Internet services. To be issued a BankID one needs to be customer of a bank that supports it. This solution differs from the one described in this thesis on many different points. The BankID needs two extra devices (smart card and offline reader), it has only one choice of security level and it is not designed for use in a single sign-on system.

A proof-of-concept service called "SIM strong authentication" was presented at the 3GSM World Congress in Barcelona, February 2006 [6]. The goal of this proof-of-concept was to demonstrate the possibility of implementing innovative services in a heterogeneous environment using Liberty Alliance Federation Standard. Telenor, Axalto, Linus and Oslo University College implemented the prototype in Oslo. The weakness with this proof-of-concept was that the implementation required new SIM cards with a preinstalled Java Card application to be distributed to the users, and Windows XP SP2/Internet Explorer with Active-X support enabled.

As far as the authors know, there exists no scalable generic SIM authentication system.

1.6 Methodology

Two paradigms characterize much of the research in the Information System (IS) discipline, natural- and design-science. The natural-science (also called behavioral-science) paradigm seeks to develop and verify theories explaining physical, biological, social, and behavioral domains. Such research is aimed at understanding reality. According to [7], natural science is often viewed as consisting of two activities, discovery and justification. Every law/theory (discovery) should be validated/proven (justification). Natural science is dominant in IS research.

The design-science paradigm seeks to create new and innovative things that serve human purposes, in comparison to understanding reality. In design science, the researcher "creates and evaluates" IT artefacts intended to solve identified organizational problems. Such artefacts are represented in a structured form that may vary from software, formal logic, and rigorous mathematics to informal natural language descriptions" [8]. Design-science consists of two basic activities; build and evaluate. Building is the process of constructing an artefact and evaluation is determining how well the artefact performs. In Figure 1, the activities have been split into five phases and show a general design-science research methodology. The outcome of the design-science research is an artefact. A design-science product, or IT artefact, is according to [8] of four types; constructs (vocabulary and symbols), models (abstractions and representations), methods (algorithms and practices), and instantiations (implemented and prototype systems). Hence, design-science designs knowledge.

[2] BankID, http://www.bankid.no

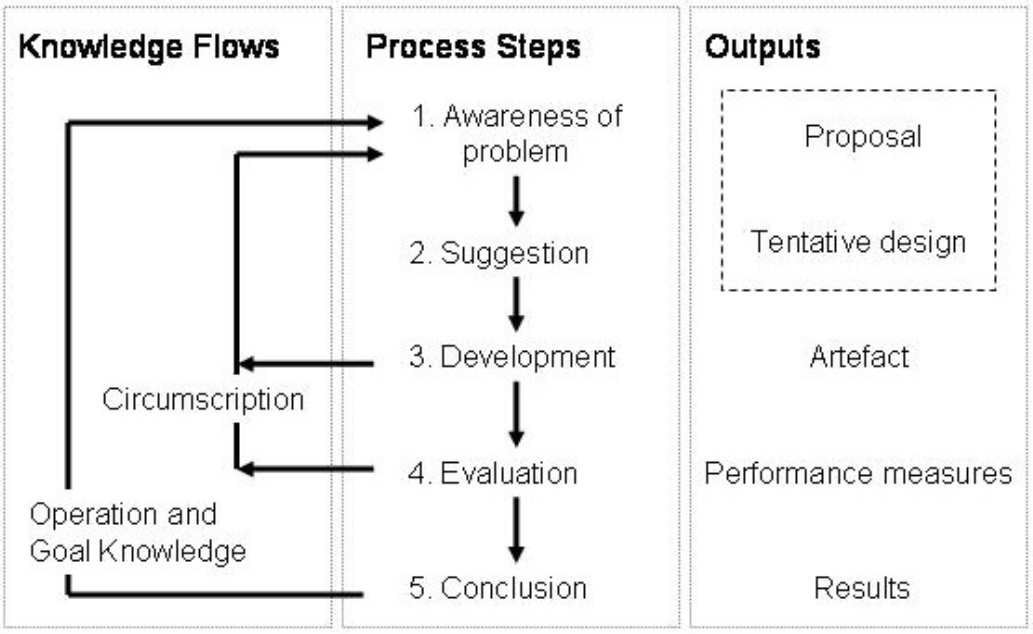


Figure 1: The General Methodology of Design Research

Figure 1 is redrawn from figure 5 in Vaishnavi's and Kuechler's design-science portal [9]. The different phases are:

1. Design science is either problem solving or performance improving, and phase 1 is meant to clarify the problem. Hence, the output of the problem phase is a proposal (formal or informal) for a new research effort.
2. The Suggestion phase is closely connected with the problem phase. Any formal proposal for design research should include a tentative design. Suggestion is a creative step that ends in a new configuration of new/existing elements.
3. An artefact is tried implemented according to the tentative design.
4. The artefact is evaluated according to functional criteria made implicit or explicit in the suggestion phase. Phase 3 and 4 gives additional information that is brought together and fed into the suggestion phase for another round with a new/modified design.
5. The conclusion phase terminates a specific design project, but there may be anomalies that serve as subject for a new project/research.

Natural- and design-science may occur similar because of the discovery/justification- and the build/evaluate-pair, respectively. The main difference is that natural science aims at understanding and explaining phenomena rather than developing ways to achieve human goals. According to [8], the danger of a design-science research paradigm is the failure to maintaining an adequate theory base, potentially resulting in "well-designed" artefacts that are useless in real organizational settings. The dangers of a natural-science research paradigm are overemphasis on theories, resulting in theories and principles addressing outdated or ineffective technologies.

A literature study including an investigation of existing authentication architectures is already performed in two earlier projects, [10] and [11]. In these projects, basic architectures were proposed. The goal of this thesis is to elaborate a more complete architecture and to implement a Prototype of the Generic SIM Authentication System.

1.7 Organization of the Thesis

This thesis is organized in 7 chapters, some appendixes, and a reference list. This is shown in Table 1, where the content of each chapter is described shortly.

Table 1: Overview of the Thesis' Content

Chapter	Content
Chapter 1	The Introduction provides motivation, problem definition, challenges, and objectives that forms the basis for this Master Thesis. It also includes the methodology used in this thesis and some related work.
Chapter 2	The Background starts with describing general concepts about authentication, before it describes specific technology relevant to authentication systems. There is also information on SIM and SIM related technology. This chapter is meant to support the analysis and design phase and be a reference for the rest of the thesis.
Chapter 3	The Analysis phase consists of Requirements, Use Cases, and Interaction diagrams.
Chapter 4	The Design phase consists of Component diagrams and Class diagrams.
Chapter 5	The Realization of the Prototype is an implementation of the designed system. The chapter also consists of a configuration of incorporated systems, services where the Prototype has been tested and an evaluation of the Prototype including the motivation for the technology choices.
Chapter 6	Critical Review provides the evaluation of the methodology used, the generality of the implemented prototype, and some security considerations.
Chapter 7	The Conclusion summarizes the results according to the problem definition presented in chapter 1. It also presents achievements and future work.
Appendixes	UML diagrams not embodied in the main chapter of the thesis, Use Case Templates, java code of the Cryptographic methods, list of mobile phones supporting SAP, configuration of JPCSC Mac OS X Port and ChilliSpot, content of the enclosed CD, and the approved paper based on the thesis.
References	The bibliography section shows all of the references used in the thesis.

When reading this thesis the readers will have several options in proportion to their level of knowledge. Advanced readers may for instance skip chapter 2, and only use it as a reference. The outline of the thesis is shown in Figure 2.

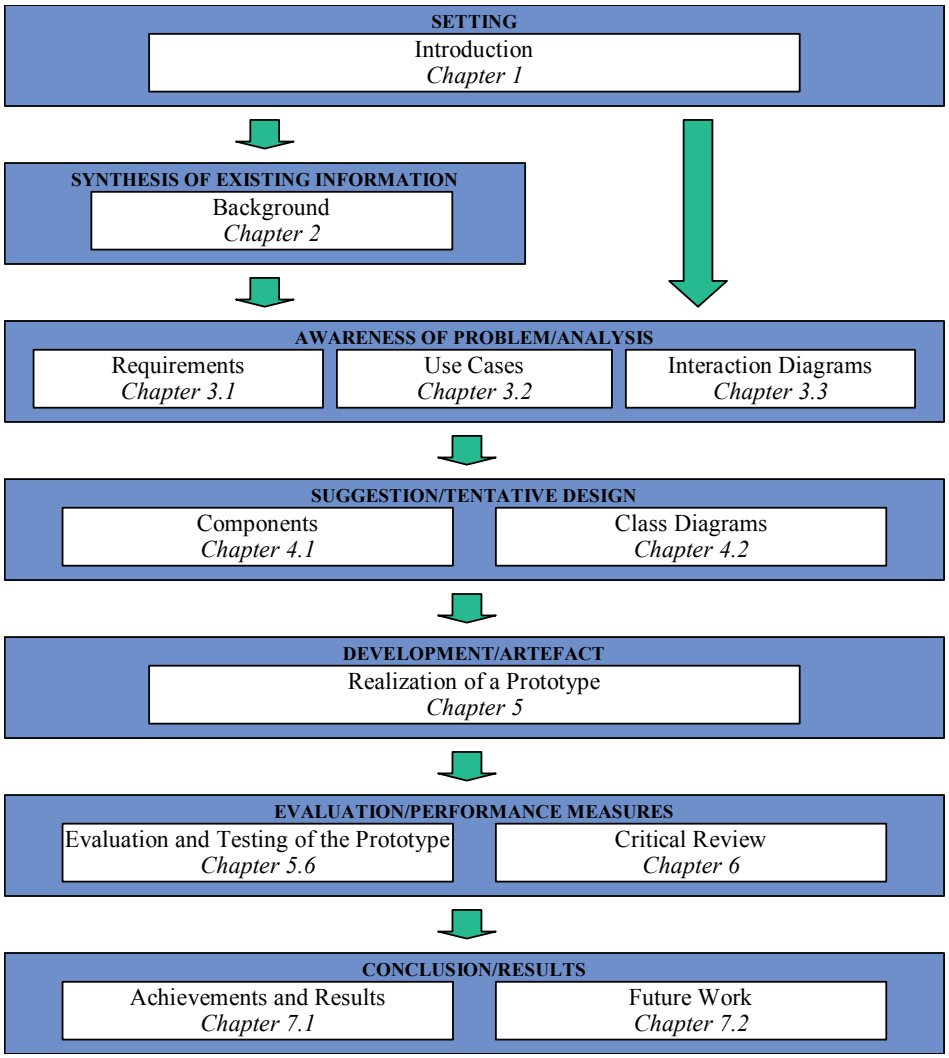


Figure 2: Outline of the Thesis

2 Background

This chapter provides background information that is crucial for the understanding of the thesis. An introduction to authentication systems and terms, cryptography, and protocols that are necessary to build a generic SIM authentication system is given.

2.1 Authentication Overview

"***Authentication*** *is the verification of the identity of a person or process. In a communication system, authentication verifies that messages really come from their stated source, like the signature on a (paper) letter*", defined by the Webster [3]dictionary. Authentication requires that the information be checked for a single, previously identified, entity. The most common authentication procedure in the computer world today is the username/password logon mechanism.

Authentication is used as the basis for:

- Authorization, determining whether a user or program is allowed access to some protected resource
- Privacy, keeping information from becoming known to other people
- Non-repudiation, ensuring that a contract cannot later be denied by one of the parties

A typical authentication procedure consists of a Supplicant, an Authenticator and an Authentication server. A Supplicant is defined as "*an entity that is being authenticated by an Authenticator*", while an Authenticator is defined as "*an entity that requires authentication from the Supplicant*". The definitions are found in RFC3580 [12].

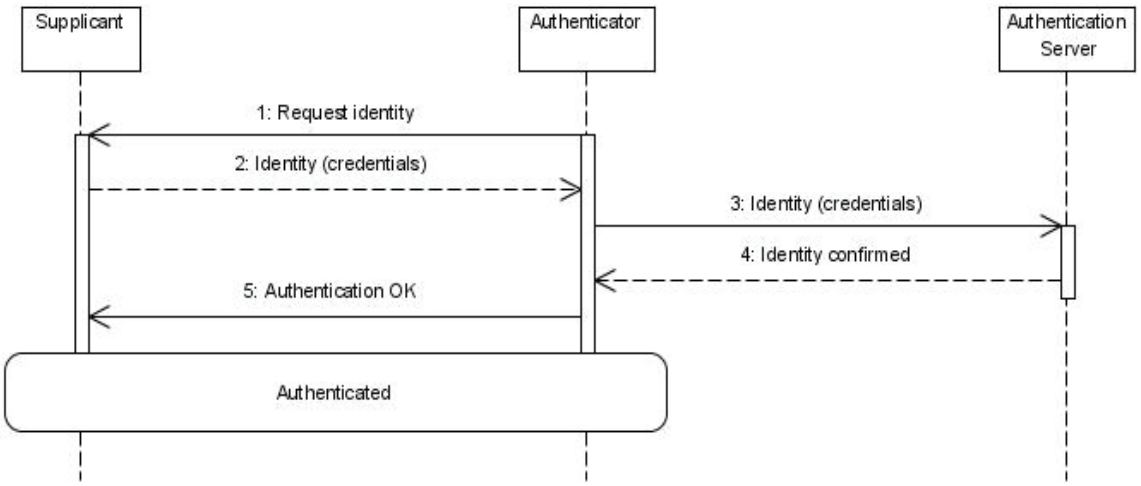


Figure 3: A General Authentication Scheme

A Supplicant will either be requested for an identity or present its identity to an Authenticator. The Authenticator will often use an Authentication Server to confirm the identity, i.e. the Authentication Server performs the authentication process on behalf of the Authenticator. The credentials are presented as the pre-identified entity and used to authenticate the user. The credentials are stored at the Authentication Server. The Authentication Server and the Authenticator can be located on the same unit. A credential is defined as "*an object that is verified when presented to the verifier in an authentication transaction*".

[3] Webster's online dictionary, http://www.websters-online-dictionary.org/

Identification

The term most similar to authentication (and often misunderstood) is identification, which is defined as "*any process through which one ascertains the identity of another person or entity, uniquely*". A common procedure to identify a person is by their uniquely shaped face.

Authentication and identification are different, but difficult to separate. Identification requires that the verifier check the information presented against all the entities it knows about, while authentication requires that the information be checked for a single, previously identified, entity. In addition, while identification must uniquely identify a given entity, authentication does not necessarily require uniqueness. For instance, someone logging into a shared account is not uniquely identified, but by knowing the shared password, he or she is authenticated as one of the users of the account. Furthermore, identification does not necessarily authenticate the user for a particular purpose.

2.1.1 Authentication Schemes

There are different authentication schemes that describe how you can use different authentication methods together to obtain greater strength. When designing a system, it is important to have in mind that different services require different security levels. An effective authentication process should have customer acceptance, be easy to use, have reliable performance, be scalable, and have interoperability with the existing systems.

This chapter describes two common authentication schemes, namely the one-factor and the two-factor authentication process.

These factors are based on:

- "What you know", e.g. password and pin code.
- "What you have", e.g. credit card.
- "What you are", e.g. fingerprint and facial features.

One-factor

One-factor authentication is typically based on "what you know" and is generally represented by the traditional password-based authentication process, where a user provides an identity with a memorized password.

Today, there is a trend to adopt Virtual Private Networks (VPNs) to allow remote users access to enterprise applications. Using only a one-factor password authentication method in such a situation constitutes a high security risk for enterprises. Although the security risk is most crucial for remote access, access to some in house applications showing sensitive information (e.g. patient data) may require more than a one-factor authentication.

A one-factor authentication method can also be based on "what you are" or "what you have", but this is unusual. Nevertheless, these methods will be elaborated in the next section and it shows how different methods may be mixed to obtain higher security.

Two-factor

The vulnerability of an authentication process increases when protected only by a password. Hackers and other criminals have an arsenal of methods to get hold of your password.

Criminals can find the password by, e.g.:

- Impersonating a person needing the password and get it just by asking
- Finding the password by the victim's bad habits:
  - Guessing easy-to-guess passwords
  - Obtaining the written password (e.g. a note attached to the monitor)
- Using software cracking the password
- Sniffing the password on network lines

Using a two-factor authentication process can solve this. Two-factor authentication is typically based on "what you have" (e.g. credit card, smart card or SIM) and "what you know" (e.g. password or PIN (Personal Identification Number)). Sometimes one of the factors could be based on "what you are", i.e. biometrics. Biometrics, such as fingerprint scanning, voiceprint, etc., could also be used in a **three-factor** authentication in conjunction with a card and a password. Two-factor authentication will drastically reduce Internet frauds, because the password would not be enough to give an adversary access to the victim's information.

Opponents to two-factor authentications argue that it is possible to bypass the physical authentication process, for instance if they get hold of your PC they can access it through safety mode. This reduces the security to a one-factor authentication process. But other schemes require two-factor authentication with the service itself. If you write down the password, it is important to separate the password from the other factor (e.g. a credit card). A common two-factor authentication process is the daily payment of groceries with your credit card where you have to enter a personal code.

No matter what type of two-factor authentication model is used, one should know that proper implementation is the key to reliability and security of the system. A poorly implemented two-factor authentication scheme may be less secure than a properly implemented one-factor scheme. This is so, because the human element is the weakest link in any security application or system.

Strong Authentication

The term "Strong Authentication" has many definitions; some of them are presented below:

NIST SP 800-63: Electronic Authentication Guideline, April 2006:

- *In Strong Authentication (Level 4), either public key or symmetric key technology may be used.*
- *Authentication requires that the claimant prove through a secure authentication protocol that he or she controls the token.*
- *Strong Authentication is based on proof of possession of a key through a cryptographic protocol.*
- *The token shall be a hardware cryptographic module validated at FIPS 140-2 Level 2 or higher*

U.S. Federal Government,

- *Strong authentication is a form of computer security in which the identities of networked users, clients and servers are verified without transmitting passwords over the network.*

Google Definition:

- *Strong Authentication is a process controlling the authenticity of the users identity on the basis of at least two of the three following factors: Something You Know, Something You Have or Something You Are.*

When using the term "Strong Authentication" through this thesis, the Google definition is chosen. An authentication system based on the SIM has two factors, the PIN code is something you know and the SIM card is something you have, hence supports strong authentication.

2.1.2 Authentication Mechanisms

Using the different kinds of authentication schemes, there are several possible compositions and implementations. This section will elaborate some of the most common authentication mechanisms. It is intended to give the reader a deeper understanding of what exists and what is possible to implement. Authentication mechanisms are generally based on data matching, and describe how to authenticate users and/or devices.

The authentication mechanisms are in principle listed from the weakest to the strongest.

Password

The password-based authentication is the oldest and most common authentication method. The password is a "what you know" mechanism and could be shared by different users. The password may be under some rule saying that for example the password should consist of 8 characters with at least one digit, one capital letter and one small letter. The password-based systems are vulnerable to different types of attacks, e.g. replay attacks[4] and dictionary attacks[5], and are not considered sufficiently secure. More information on attacks can be found in [13].

There is however different ways of implementing password-based systems; e.g. transfer (storage) of plaintext passwords and transfer (storage) of hashed passwords. The hash function takes a string (e.g. a password and a seed value) of any length and produces a one-way fixed length output (the hash value). The output is often called a message digest, and it should not be possible to produce two identical message digest from two different inputs. By using a hash function, you avoid sending or storing the password in plaintext, thus making the system a bit more secure than the plaintext system. This is because eavesdroppers cannot reconstruct the password from the hash.

OTP – One Time Password

A motivation for using OTP is to prevent attacks such as, eavesdropping on network connections to obtain authentication information that could be used at a later time in a replay attack to break into a system of some sort.

[4] Replay attack: an impersonation or other deception involving use of information from a single previous protocol execution, on the same or a different verifier. For stored files, the analogue of a replay attack is a restore attack, whereby a file is replaced by an earlier version. [13]

[5] An adversary, who tries all words in any on-line or available list of words, using a so-called *dictionary attack*, may uncover the password. [13]

The OTP mechanism uses a secret pass-phrase to generate a password allowed only once. Hence, a replay attack does not work. Since the OTP is generated locally, your secret pass-phrase is never sent throughout the network. In the standard for OTP systems [14], the generator produces the one-time password from a challenge from a server and the secret pass-phrase. Some Internet accounts are accessed using another OTP system scheme; the user has a code generator device containing one-time-passwords in a distinct sequence, only accessible by entering the correct PIN code. A server has the same sequence of passwords and during the authentication process the passwords are compared. There are several similar solutions to the scheme described in the text.

Challenge/Response

Challenge/Response authentication is based on a shared secret. The Authenticator sends a Challenge (e.g. random number), and the Supplicant uses for example a symmetric key-hashing algorithm (a one-way function) to encipher the random number. The hash value (Response) is sent back to the Authenticator and compared with its version of the enciphered Challenge. If identical, the Supplicant is authenticated. See Figure 4.

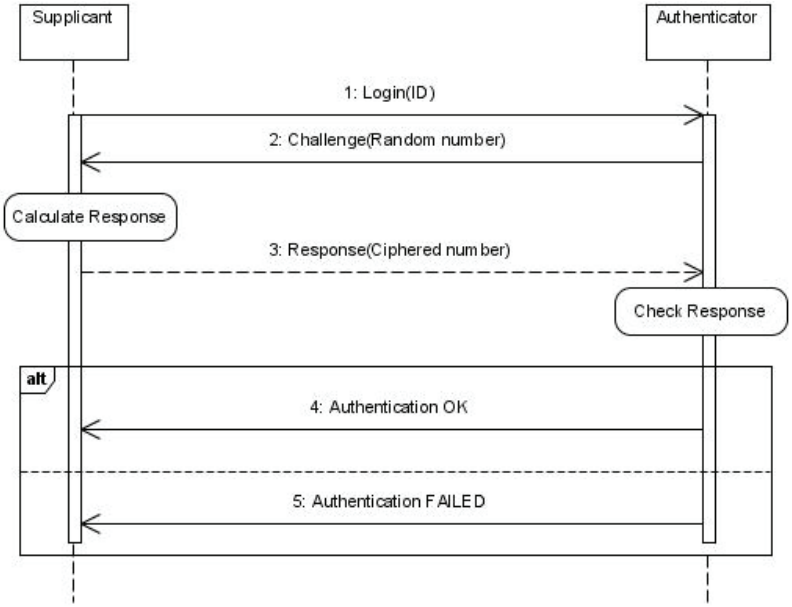


Figure 4: Challenge/Response Authentication Scheme

The GSM SIM authentication is a variant of the Challenge/Response-scheme.

PKI – Public Key Infrastructure

According to [15], the idea of Public Key Infrastructure (PKI) is: "*A PKI is the basis of a pervasive security infrastructure whose services are implemented and delivered using public-key concepts and techniques*". Hence, PKI is the sum of all the facilities and procedures needed to implement public key cryptography. Thus, PKI can be used as the underlying technology to support all of the security control objectives, namely authentication, confidentiality, data integrity, and non-repudiation. This is accomplished through a combination of symmetric and asymmetric cryptographic techniques used in a single infrastructure.

Public key cryptography consists of a private[6] and a public key. The private key is only known by its owner and is used to decrypt messages already encrypted with the public key. The private key is also used to sign messages to prove from who it is (only the corresponding public key can verify this signature). Symmetric cryptography, on the other hand, uses a symmetric key, and the same key is used to both encrypt and decrypt messages. The public key is put in a digital certificate and made available through the PKI system; only the private key needs to be kept secret. A public key infrastructure consists of many different services and technologies, the most important components are:

- The Certificate Authority (CA) that issues and verifies the digital certificate. When the CA has ensured that everything in the certificate is correct it digitally signs it.
- The Registration Authority (RA) is responsible for identifying the correctness of the identity of the person that is issued the certificate. The RA has close relationship with the CA, and can sometimes be the same authority.
- One or more directories where the certificates are stored.
- The certificate management system, which is the system where certificates are published, suspended, renewed or revoked.

A certificate is digitally signed by a CA and contains information about the owner of the private key, the key length, the algorithm used, the validity period and most importantly the public key. The most common certificate standard is the ITU-T X.509.

The infrastructure of a PKI will vary from place to place; Figure 5 shows the components in a simple setup of a PKI. First the CA will issue the certificate and private key to the user, when the authenticator wants to authenticate the user it sends a challenge. The user signs the challenge with his/her private key and sends the signed challenge back to the authenticator together with the certificate. The authenticator verifies that the signature corresponds with the certificate and contacts the CA to control that the certificate is real.

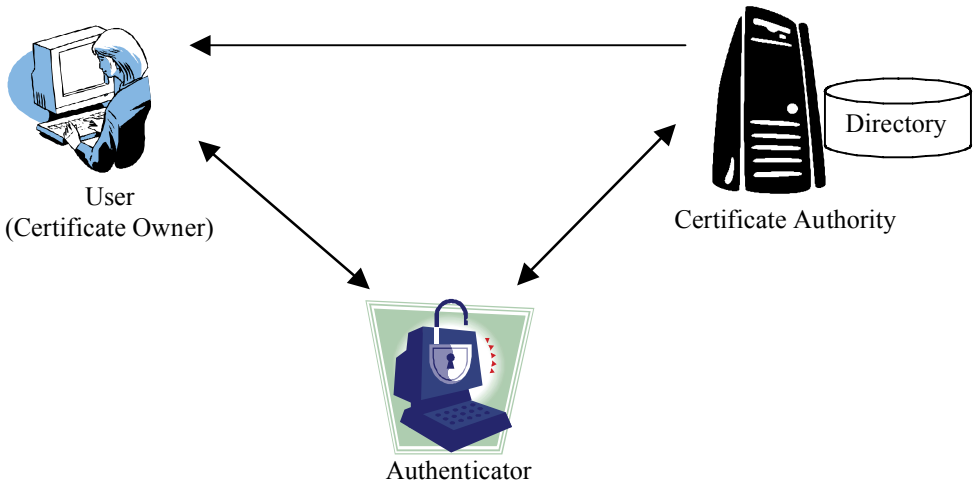


Figure 5: Components in a Simple PKI

In a typical public-key based authentication scheme, the Supplicant is authenticated without revealing any sensitive information. Nothing travels over the network that can be used by an eavesdropper to later impersonate the Supplicant. And there is no need for a costly process of pre-establishing a shared secret. The public key must of course still be distributed.

[6] The private key is a secret key, but the term, “secret key”, should only be used in a symmetric key cryptography context.

The private key and/or the digital certificate are often stored on a workstation, and are then subject to compromise by physical access to the machine or via unauthorized network access. To enhance security, at least private keys should be stored on tokens, secured by PINs. This increases security dramatically, and at the same time makes PKI a more mobile solution. Digital certificates can be stored on tokens to reduce the certificate management burden.

Nevertheless, PKI is one of the best-known security frameworks for open systems; e.g. it is a fundamental security component of all major Internet protocols for authentication and communication like: TLS (Transport Layer Security), IPSec (Internet Protocol Security), 802.1x, SIP (Session Initiation Protocol) [16]. Examples of PKI systems that use smart card in Norway today are BankID [17] and Buypass. The former system stores the client certificate in the network or at the client computer, and uses the smart card to generate a one-time password used in the authentication of the user. The latter system stores the certificate on the smart card and hence need to connect the card with the computer with a smart card reader.

Biometry

Biometric authentication is a "what you are"-factor authentication mechanism. It is based on physiological characteristics, like fingerprint scanning, iris image, facial image, voice print, etc. The advantages are that physical presence is ensured hence authorization cannot be transferred. The authentication mechanism is a comparison of a physiological characteristics and pre-stored data (which are a digital representation of the physiological characteristics). Thus, the security is dependent on how good this digital representation is. Biometrics should be used in a multi-factor authentication scheme.

2.2 Cryptography

This section elaborates how communication can be secured using hash functions and message authentication codes.

2.2.1 Communication Security

The origin of the transport level protocol was SSL (Secure Socket Layer) 1.0, proposed and implemented by Netscape for securing browser traffic. SSL 3.0 is the most common security protocol used today. The Internet Engineering Task Force (IETF) chartered a working group in 1996, accepted submissions from Netscape (responsible for SSL 3.0) and Microsoft (responsible for Private Communication Technology (PCT)), and delivered RFC 2246 with TLS 1.0. Hence, TLS is a standardized protocol. TLS is not directly compatible with SSL, but has a fallback modus to SSL 3.0. Some refer to TLS 1.0 as SSL 3.1, and when SSL is discussed or referred to, TLS is included.

The SSL protocol runs above the TCP/IP (Transmission Control Protocol/Internet Protocol) layer and below high-level layers like for instance HTTP (HyperText Transfer Protocol). It allows an SSL-enabled server to authenticate itself to an SSL-enabled client, and vice versa, and at the same time it allows both machines to enable an encrypted channel. In the authentication process (the SSL handshake) SSL uses private/public keys and certificates. A problem with this scheme is the protection of the private key. If someone gets hold on the private key it could impersonate the owner, and it is therefore normally protected encrypted under a key derived from a password.

While the private/public key pair is used during the authentication, it is not used when sending messages. During the handshake the client and the user agrees on a session key that is a secret key used in a symmetric encryption scheme. Symmetric encryption is used after the initial authentication due to higher efficiency and to minimize the exposure of messages encrypted with the private key. SSL can use different public-key cryptography, symmetric encryption, and key-exchange algorithms.

Secure HyperText Transfer Protocol (HTTPS) is the most common way to secure the communication on the Internet. HTTPS is not a separate protocol, but refers to the combination of a normal HTTP interaction over an SSL transport mechanism. HTTPS can be used in combination with SSL to ensure authentication and secure transportation when accessing a web page requiring high security. The default TCP port for HTTPS is 443 in comparison with normal HTTP where the default is 80.

2.2.2 Hash Function

A hash function is a way of creating a small value (often called hash value) of a fixed size of an arbitrary long message. It is often called a fingerprint and is used to make sure no one has tampered with the message. The message if often padded so the message is a multiple of a fixed block size (n*block size). Then the padded message is run through the algorithm n times producing the hash value.

A good hash function should be one-way and have few or no collisions. A one-way function is a function where it is not possible to deduce the original input which produced a given hash value. A collision is when two different messages give the same hash value.

2.2.3 Message Authentication Code

A Message Authentication Code (MAC) is an authentication technique that involves the use of a secret key to produce a small block of data. MAC is actually pretty similar to the hash function described above, where the difference is the secret key. The secret key is used to authenticate the message as well (see Figure 6). It is not possible to compute the MAC without the key. Hence, the two communicating parties must share a secret key (k).

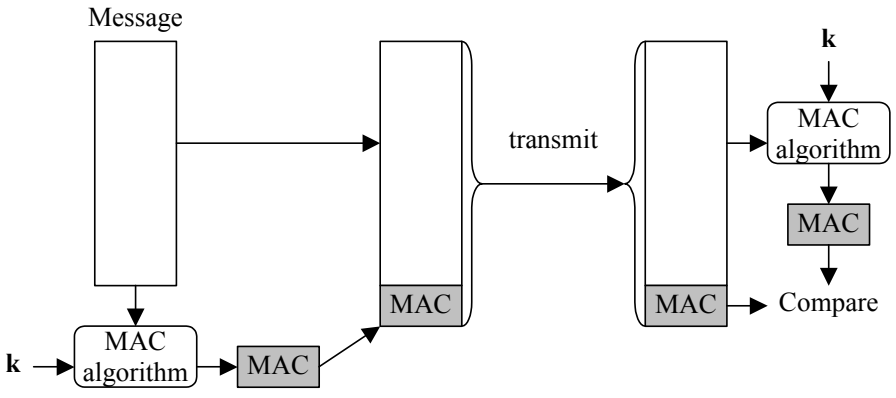


Figure 6: Usage of a Message Authentication Code

MAC is often used in conjunction with a cryptographic hash function, for instance Hashed MAC (HMAC).

2.3 Smart Card

There are two categories of Smart Cards or Integrated Circuits Cards (ICCs): memory cards and microprocessor cards. All later referrals to Smart Cards are microprocessor cards. To promote interoperability among Smart Cards and readers, the ISO 7816 standard was developed, which later was adopted by GSM and EMV (Europay, MasterCard, and Visa). EMV has some additional data types and encoding rules for use by the financial services industry.

The Smart Card consists of (see Figure 7: Smart Card Components):

- A Central Processing Unit (CPU)
- Random Access Memory (RAM)
- Read-Only Memory (ROM)
- Electrically Erasable Programmable Read-Only Memory (EEPROM)

The operating system is stored on the ROM and customized applications and data are typically stored on the EEPROM. All smart cards rely on the reader for its energy.

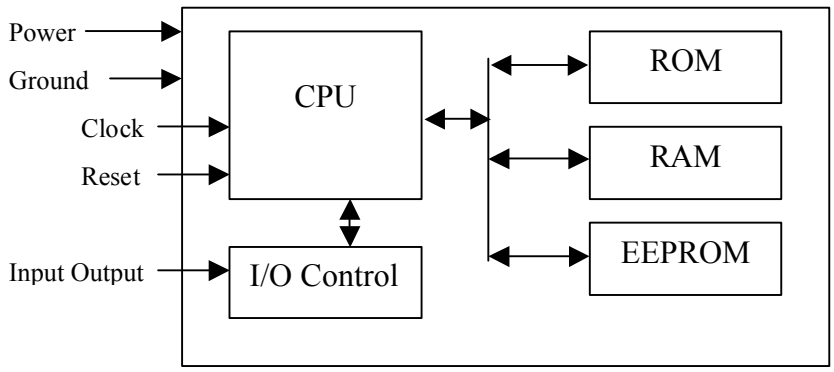


Figure 7: Smart Card Components

Smart Cards are mostly used as credit cards or SIMs, but they are also used as identification (secure authentication), pay television, ticket control, electronic money and so on. Different protocols protect these applications.

2.3.1 SIM – Subscriber Identity Module

The SIM is a logical module that runs on an Integrated Circuit Card (ICC) type of Smart Card. It is used to store keys to authenticate mobile users, messages, and phone numbers. Since a subscriber's identity stays with the SIM, a subscriber may move their phone number from one ME to another (personal mobility). The specification of the SIM and the handset was considered part of the GSM standard. For example GSM 11.11 describes the interface ME to SIM, and GSM 11.14 describes the interface SIM to ME. The newest versions of the specifications are standardized by the 3GPP as TS 51.011 [18] and TS 51.014 [19].

The content and use of the SIM can be protected by different codes, referred to as PIN[7] codes. PIN1 protects normal usage (like authentication) while PIN2 protects special services or blocks special numbers. Normally, the PINs will be blocked after a number of invalid

[7] PIN is an obsolete term for the Card Holder Verification information (CHV)

attempts (normally 3). The PIN UnlocK (PUK) is used to unlock the PIN. PUK1 and PUK2 reset PIN1 and PIN2, respectively. If the PUK is entered invalid a numerous times (normally 10), the SIM will block local access to privileged information permanently.

International Mobile Subscriber Identity (IMSI)
The International Mobile Subscriber Identity (IMSI) is a unique number for every subscriber in the world. It usually consists of 15 decimal digits. The number follows the ITU E.212 standard, "The international identification plan for mobile terminals and mobile users". The three first digits are the country code, the next two or three digits are the network code. The remaining part of the number is the unique number within the specific network. An IMSI for a Norwegian subscriber at Telenor Mobil could be [20]:

IMSI: 242011234567891
Country code (Norway): 242
Network code (Telenor): 01

Cryptographic functions
The most secret value on the SIM is the 128-bit randomly generated secret key, Ki. The key is securely protected on both the SIM and in the AuC and they are the only places the key is stored; the phone, for example, never learns the key.

Because the SIM is an "intelligent" device with a Central Processing Unit (CPU), the authentication and the key generation can be performed inside the SIM. The authentication algorithm used in the GSM system is known as the A3 algorithm and the key generation algorithm is known as the A8 algorithm. The A3 algorithm's task is to generate the 32-bit Signed Response (SRES) needed in the SIM authentication process. The algorithm takes as input the 128-bit Random Challenge Number (RAND) generated by the Home Location Register (HLR) in the GSM system and the 128-bit authentication key (Ki) stored in the SIM and in the AuC. The A8 algorithms takes the same input as the A3 algorithm, but the task of the algorithm is to generate the 64-bit Cipher key (Kc) used in the GSM symmetric cryptography algorithm A5. The purpose and the parameters of the A3 and A8 algorithm are specified in the 3GPP specification TS 03.30 [21]

Since both algorithms take the same input, A3/A8 are usually implemented together. The operator chooses which type of algorithm the SIM actually implements. The most common implementation for the A3/A8 algorithm is the COMP128 (version 1 and 2) algorithm. This algorithm actually performs both the A3 and A8 algorithm in the same stage.

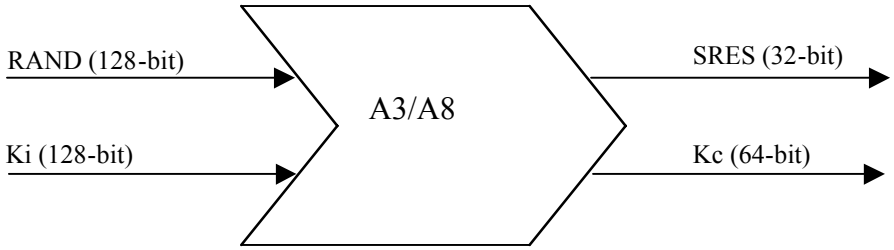


Figure 8: Authentication (A3) and Key Generation (A8) Algorithm Input and Output

SIM Application Toolkit (SAT)
The SIM Application Toolkit defines how the card should interact with the outside world and extends the functionality of the SIM card. It is abbreviated in three different ways: SAT, SIM AT or STK (SimToolKit). SAT will be used throughout this document.

SAT is defined as an optional feature to implement on the SIM card, and it was first specified in the GSM 11.14 standard. With SAT operators can create additional menus on the phone screen that provide user-friendly access to important services. Such menus would otherwise be hard to deploy on different types of handset all with different software. The only requirement is that the mobile phone supports SAT.

The different SAT mechanisms are listed in Table 2, and are defined in GSM 11.14 [19].

Table 2: SIM Application Toolkit – Mechanisms

Mechanism	Explanation
Profile download	Mechanism for the Mobile Equipment (ME) to tell its capabilities to the SIM.
Proactive SIM	Mechanism whereby the SIM can initiate actions to be taken by the ME, e.g. displaying text, sending short message etc. This does not conflict with the T=0 protocol, but introduces a new TERMINAL RESPONSE, SW1, which tells the ME that the SIM has a message waiting. The ME then requests the message.
Data download to SIM	Data downloading to the SIM uses either dedicated commands (the transport mechanisms of SMS point-to-point and Cell Broadcast) or the Bearer independent protocol. Transferral of information over the SIM-ME interface uses the ENVELOPE command.
Menu Selection	Used to transfer the SIM application menu item, which has been selected by the user to the SIM. Possible menu entries are supplied by the SIM in a proactive SIM command.
Call control by SIM	When activated, all dialled digit strings are first passed to the SIM before setting up the call. The SIM may deny/allow/modify/replace the call request.
MO Short Message control by SIM	All MO Short Messages are first passed to the SIM. The SIM may deny/allow sending the message or modify the destination address.
Event download	A set of events to monitor is supplied by the SIM in a proactive SIM command. The event download mechanism is used to transfer the details to the SIM.
Security	The applications using SIM Application Toolkit may require data confidentiality, integrity, and validation.
Multiple card	One event and a set of proactive commands are supplied to monitor and control Card x behaviour. (Only supported by some implementations)
Timer Expiration	Used to inform the SIM when a timer expires. (SIM is able to mange timers running in ME with proactive commands)
Bearer Independent Protocol	A set of proactive commands and events allows the SIM to establish a data channel to a network server. The protocol allows SIM and the server to communicate transparently. (Only supported by some implementations)

A SIM application session is the execution of a sequence of commands internal to the SIM that can result in the performance of one or several proactive SIM sessions. The SIM application session can be started by any event in the card session, and can execute for the duration of the card session. Processing of the SIM application session will not interfere with normal GSM operation.

SIM API – SIM Application Programming Interface
The SIM API allows application programmers to gain easy access to functions and data described in 3GPP TS 11.11 (transport functions) and 3GPP TS 11.14 (pro-active functions), such that SIM based services can be developed and loaded easily onto the SIM. 3GPP TS 03.48 [22] describes the procedure to remotely load applications onto SIMs. It specifies secure structured packets, and uses Short Message Service Point to Point (SMS-PP) and Short Message Service Cell Broadcast (SMS-CB). This opens the possibility of secure remote management of files on the SIM in conjunction with SMS and the SIM Data Download feature of the SIM Application Toolkit. A SIM API requirement is that an applet should not interfere with the basic GSM services.

The development of a SIM application is as follows:
1. An application programmer will write the code in a standard development environment e.g. C, Java, Visual Basic, etc.
2. The source code including the libraries is compiled into byte code, called a Toolkit Applet File and optionally optimised.
3. The card issuer or a trusted party gets the file and loads the code into the SIM's EEPROM (or another smart card application platform).
4. The file is installed and eventually executed.

SIM API for Java Card
3GPP TS 03.19 [23] defines the SIM API for Java Card, which allows applets written in the Java language to be executed on a smart card. The applets run within a Java Card Runtime Environment (JCRE), which provides classes and methods to help developers create applets. Each applet is identified by its Application Identifier (AID). The Java Card is inserted into a Card Acceptance Device (CAD), which selects an applet to execute. The CAD is the SIM in GSM. Commands (e.g. ENVELOPE) to be executed are formatted and transmitted in the form of Application Protocol Data Units (APDUs). The applets reply with a Status Word (SW) or data.

2.3.2 USIM – Universal Subscriber Identity Module

The USIM is the name of the module that runs on a Universal Integrated Circuit Card (UICC) type of smart card. It is important to note that a UICC may contain a SIM application in addition to a USIM application. "I'M Technologies" has a good explanation of the confusing naming [24]:
"In the 3G environment it becomes more important to distinguish clearly. Talking about USIM means to talk about the data and functions to be provided by a device (e.g. UICC). This means, USIM is better described as an application in order to clearly distinguish it from the carrier of the application. This becomes even more apparent when we talk about a single UICC being able to support several USIMs. And it is most obvious when one understands that a UICC may not only support one or more USIM in a UMTS environment, but may even be able to carry data and functions for a RUIM, USAT and even a WIM in parallel. This enables a user to change from UMTS to CDMA2000 when traveling".

The authentication mechanism used in Universal Mobile Telecommunications System (UMTS) has improved from what is used in GSM. UMTS still checks the subscriber's identity by the challenge-response mechanism; the biggest improvement is that it uses mutual authentication and have a new Authentication and Key Agreement (AKA) technique. The 3GPP specifies "Characteristics of the USIM application" in TS 31.102 [25] and the USIM security in TS 33.102[26].

The USIM like the SIM stores the 128-bit subscriber authentication key denoted K. In total, five one-way functions are used in the USIM authentication computation. The functions are called f1, f2, f3, f4 and f5. The function f1 takes four input parameters: the key K, the random number RAND, the sequence number (SQN) and the Authentication Management Field (AMF). The other functions only take K and RAND as inputs. The Figure 9 below illustrates the authentication handling in the USIM.

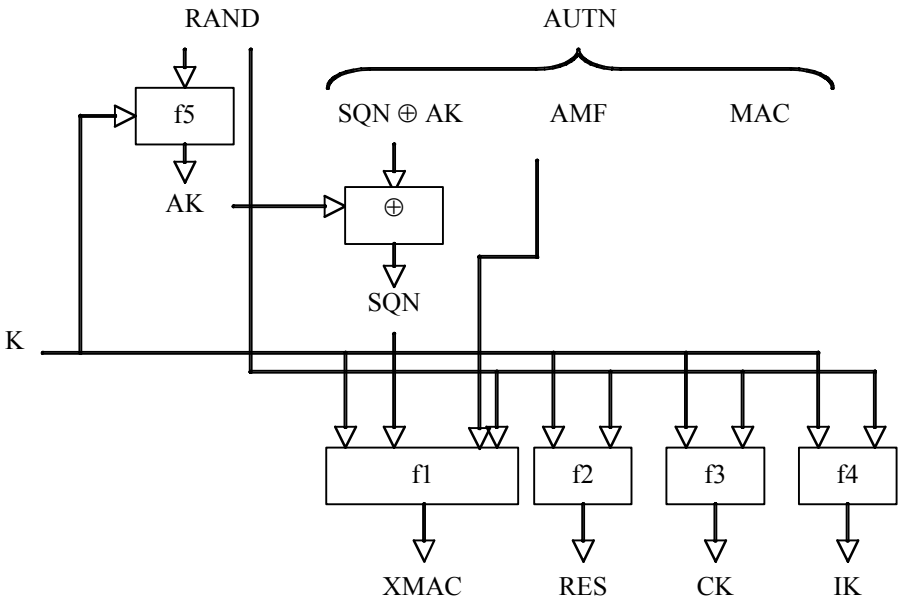


Figure 9: Authentication Handling in USIM [26]

The USIM first computes the anonymity key (AK); this is used to conceal the SQN. Next the USIM computes XMAC and compares this with MAC, which is included in AUTN. If something is wrong an error message is sent back to the system. In the positive case the computed RES parameter is sent back to the system.

The purpose of the SQN is to provide the USIM with proof that the generated authentication vector has not been used in an earlier run of authentication. This will protect against an attacker that has recorded an earlier authentication event. The SQN management is operator specific and there are two basic strategies. Each user may have an individual SQN or the SQN could follow a global counter (e.g. time).

The choice of algorithm (f1-f5) is in principle operator-specific just like the A3/A8 in the GSM. The 3GPP has created an example set of algorithms called MILENAGE (based on AES); they are specified in TS 35.206 [27]. When a UMTS terminal visits a network that only supports GSM authentication and encryption, the terminal must authenticate itself using the A3/A8 interface. The MILENAGE algorithms support conversion rules to make this work; you can actually use these rules to implement a GSM SIM card, instead of using proprietary algorithms [28].

Just like the SIM card the USIM has an application toolkit, USAT. The new thing on the UICC cards is something called *USAT Interpreter* (USAT-I). USAT-I is a byte code interpreter; in comparison with USAT this is an application within UICC (and not part of the USIM).

2.3.3 Readers

In the GSM system the SIM reader resides inside the mobile equipment. For a personal computer there are 3 main choices of reading the existing SIM/USIM card. Also, the user could have a dual-SIM subscription, relieving him/her of the task of moving the SIM between various SIM readers:

USB or PCMCIA (PC Card) reader: With this type of reader the user has to physically remove the SIM from the handset and insert it into the reader (unless the user has a dual-SIM subscription). The reader is connected to the computer through either a USB or a PCMCIA port. The USB reader is the most common method today for accessing a SIM Card. A standard USB smart card reader can also be used to read the SIM card, to use this type of reader the SIM card needs to be attached to a smart card size frame.

Figure 10: SIM Readers

Hardwired SIM Reader/Smart Card Slot: This requires that the computer have a built in reader, which a number of laptops has today. A hardwired approach provides secure access for the client device.

Using Handset as remote reader: With this method the phone will act as a remote SIM reader to the computer. This requires the handset to have a SIM access server and the computer must have a SIM access client. An emerging solution is to connect to the handset via a Bluetooth connection, but for this to work the phone needs to implement the Bluetooth SIM Access Profile (SAP) [29]. Alternative access solutions to a remote SIM reader could be over an Infrared connection, through cable or with the new Near Field Communication (NFC) technology.

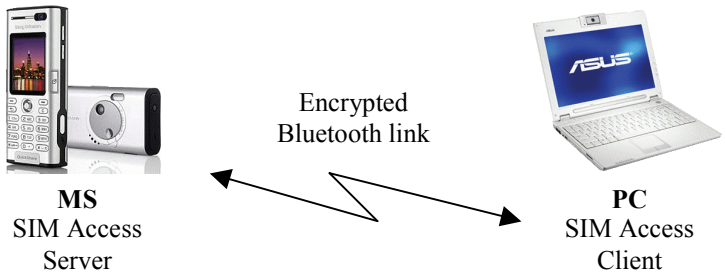


Figure 11: Bluetooth SIM Access

There exist also solutions today where we have SIM cards built into the external device; examples of this are GPRS PCMCIA cards and USB dongles.

It is important that the choice of SIM reader makes the use easy for the user. The biggest problem with the USB reader that is most common today is that you need to remove the SIM card from your phone or need a second SIM card. The user also has to buy another device.

2.3.4 Communication Interfaces

Applications that are using Smart Cards are in two parts: one part is running in a Smart Card and the other is running on a terminal or server. The communication between the SIM card and the SIM reader follows the T=0 protocol which is defined in [30]. The step in the application protocol consists of sending a command, processing it in the receiving entity and sending back the response. Therefore a specific response corresponds to a specific command, referred to as a command-response pair. An application protocol data unit (APDU) contains either a command message or a response message.

An important property with the SIM interface is the exchange of files. The SIM maintains information in a series of "files" that are organized hierarchically, much like the operating system of a personal computer. There are three different types of files on a SIM: a master file (MF), dedicated files (DF), and elementary files (EF). A two-byte hexadecimal number uniquely identifies the different files. There is one master file on a SIM, which holds all the other files in a hierarchical structure. GSM defines two dedicated files immediately under the MF, DF-GSM containing GSM application files and DF-Telecom containing the application service features. Data in these files are accessed through APDU commands sent to the operating system. The message format is defined in [30] and will be described next.

APDU – Application Protocol Data Unit

A number of commands are defined for GSM SIM cards, including functions to read and write data, confirm security features, and run the GSM authentication algorithm. Completing an entire GSM procedure may require a series of APDU command-response pairs.

There are five fields in an APDU command (see Figure 12). The class of instruction (CLA) is always A0 for GSM. The instruction code (INS) indicates the particular command to be performed. Figure 12 shows the instruction codes used in the SIM authentication. P1 and P2 are parameters for the specific command, and Lc contains the length of the Data field if any. Le specifies the maximum number of bytes expected in the data field of the response to the command.

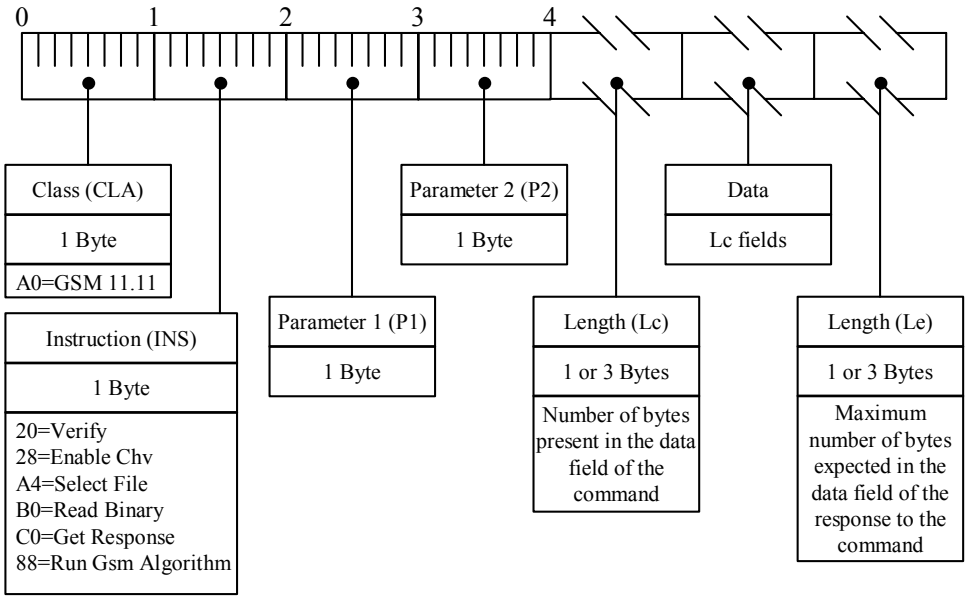


Figure 12: Command APDU Format

The response to a command is returned in the response APDU, which contains three fields (see Figure 13). The Data field, if any, contains information requested in the command. SW1 and SW2 are status bytes indicating the success or failure of the command.

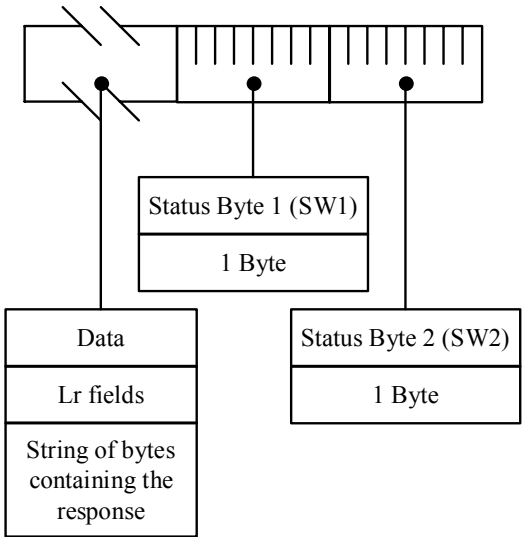


Figure 13: Response APDU Format

There are also developed different mechanisms to support the programmers with a higher-level API than the low-level T=0 protocol. Popular frameworks are the PC/SC and OpenCard Framework (OCF), which will be described next.

PC/SC – Personal Computer/Smart Card

The PC/SC Workgroup was formed to address limitations in existing standards, from a personal computer/smart card application perspective. They have developed a specification PC/SC API [31] that is completely platform independent. Microsoft has implemented the PC/SC API as part of the Win32 API, which is the fundamental toolset for building Windows applications. A competitive approach to the Microsoft implementation of the PC/SC API is the MUSCLE project [32], which coordinates the development of smart cards and applications under Linux. They have developed a tool called PCSC Lite that implements the PC/SC API. According to a Smart Card Alliance article [33], the PCSC Lite is stable, fast, and easy to use, and some Windows deployments of smart cards have opted to use PCSC Lite, rather than the native Windows PC/SC implementation, because of the transparency and flexibility of PCSC Lite. Because of the popularity, PCSC Lite has been ported to many different platforms, including Linux, Solaris, FreeBSD, NetBSD, OpenBSD, Mac OS X and Microsoft Windows.

OCF – OpenCard Framework

The OCF is a standard Java Framework for working with Smart Cards. Based on Java technology, OCF enhances portability and interoperability. One of the main aspects of OCF's architecture is a clearly structured model that divides the functionality provided by the Application and Service Developers and the Card and Terminal Providers. This is to reduce interdependencies and is done by introducing two main interfaces:

- A high-level application-programming interface that hides the characteristics of a particular provider component.
- A common provider interface that enables the seamless integration of Smart Cards building blocks from different vendors.

The OCF and PC/SC are complementary rather than competing, which one to choose all depends on the programming language and which type of standards the reader is compatible with. In fact, it is possible to access PC/SC through the OCF interfaces also (using the OCF API com.ibm.opencard.terminal.pcsc10 package).

2.4 GSM Authentication

GSM is an acronym for the "Global System for Mobile communication". It is a common standard, and combined with good roaming agreements between the Mobile Network Operators (MNOs), results in the most successful and widespread mobile standard. GSM is a Second Generation (2G) mobile telephone system, i.e. digital speech and signalling channels.

A GSM network is composed of several entities (see Figure 14); the parts involved when being authenticated will be elaborated. A GSM mobile phone, called Mobile Station (MS), is composed of a Mobile Equipment (ME) and a Subscriber Identity Module (SIM). The Base Station Subsystem (BSS) controls the radio link with the MS. The BSS is composed of several Base Transceiver Stations (BTSs) and a Base Station Controller (BSC), and communicates with the MS over an Um-interface. The BSS communicates with the Mobile services Switching Centre (MSC) via the A-interface.

The MSC is a part of the GSM Network Sub-System (NSS) along with the Home Location Register (HLR), Authentication Centre (AuC), Visitor Location Register (VLR), and Equipment Identity Register (EIR). The VLR has information about the active subscribers in its network. The MSC is coordinating and setting up calls to and from Mobile Stations. The information for this figure is collected from [34], chapter 3 and 5 of [35]. The GPRS (General Packet Radio Service) Core Network and the GSM network's connection to the PSTN (Public Switched Telephone Network) are not relevant in the context of this thesis. The communication is over SS7 (Signal System no. 7) between the entities in the BSS and NSS, while the Um interface is a radio interface.

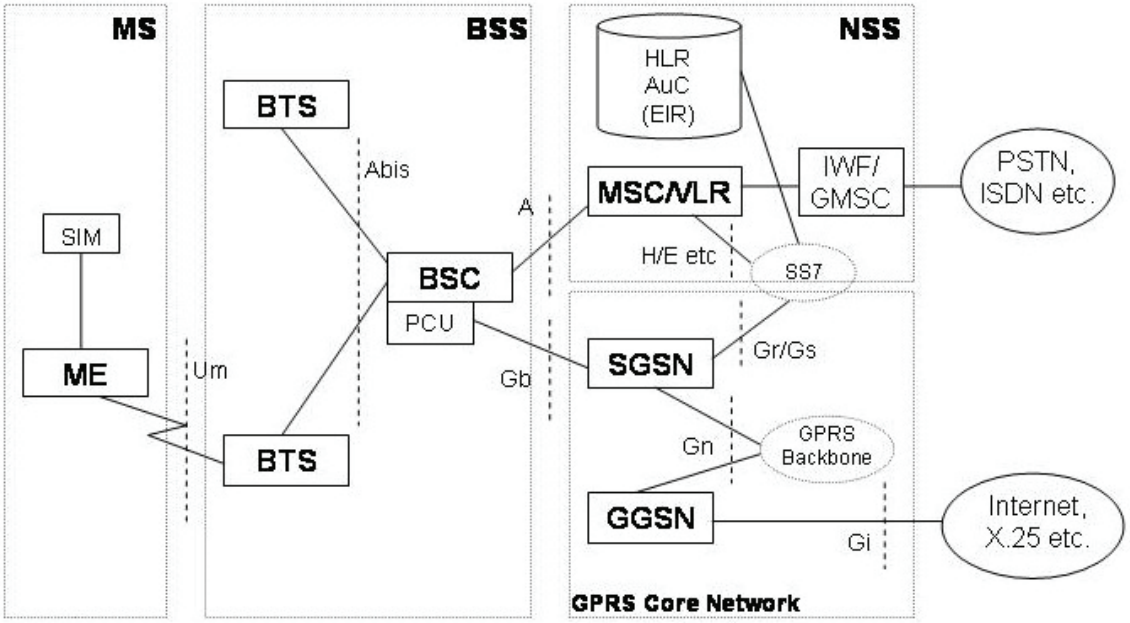


Figure 14: GSM/GPRS Network Architecture

During a GSM Authentication the most important parts of the GSM Network is the HLR/AuC and the MSC/VLR in the NSS and the SIM in the MS. The BSS and the ME is only used for communication.

HLR – Home Location Register
The HLR is a database used for storage and management of subscriptions. The HLR is considered the most important database, as it stores semi-permanent data about an MNO's entire subscriber base. HLR subscriber information includes the International Mobile Subscriber Identity (IMSI), service subscription information, location information (the identity of the currently serving Visitor Location Register (VLR) to enable the routing of mobile-terminated calls), service restrictions and supplementary services information.

The HLR handles transactions with Mobile Switching Centres (MSCs) and VLR nodes, which either request information from the HLR or update the information contained within the HLR. The HLR also initiates transactions with VLRs to complete incoming calls and to update subscriber data.

There is logically one HLR per GSM network although it may be implemented as a distributed database. The HLR data is stored for as long as a subscriber remains with the MNO.

VLR – Visitor Location Register
The VLR is a database attached to one or several MSCs. It holds much of the same information as the HLR but in addition it temporarily stores subscriber data for those subscribers that are in the service area. The VLR also keeps a Temporary Mobile Subscriber Identity (TMSI) of the subscribers connected. In most networks the MSC and the VLR are one and the same piece of equipment.

AuC – Authentication Centre
The Authentication Centre (AuC) is a function that provides authentication of each subscriber that tries to connect to the GSM core network. The AuC contains information about subscribers to a GSM network that must be kept secure, such as the 128-bit permanent key (Ki) of the subscriber's SIM card. This information is never released to other parts of the GSM network. Instead, the AuC releases derived security information (triplets) that cannot compromise the security of the Ki. A triplet consists of:

- RAND – a 128 bits RANDom number.
- SRES – the 32 bits Signed RESponse to the RAND.
- Kc – the 64 bits encryption key.

The triplets are sent to the MSC/VLR, where the authentication process takes place. To establish the encrypted communication over the wireless interface, the BSC receives a Kc from the MSC.

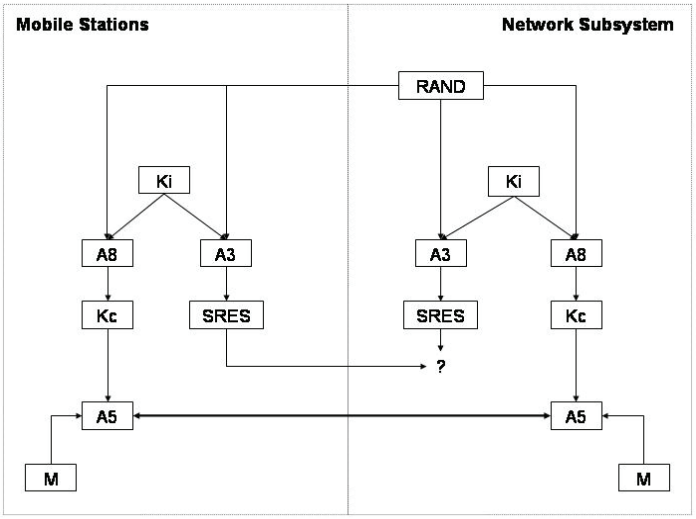


Figure 15: Generation of Triplets in GSM

Figure 15 shows the key generation, validation and encryption process. For each IMSI, the AuC stores the secret key (Ki) and an algorithm id. A3 and A8 are the generic names for the algorithms that are part of the GSM terminology, but the implementations are left to the operator. Nevertheless, there are suggested default implementations that will be described later. When the MSC/VLR requests a new triplet, the AuC gets a RAND from the HLR and combined with the Ki they are fed into A3 and A8. The algorithms calculate the SRES and the Kc. This is done several times to obtain an array of values (triplets). The MS gets a RAND from the MSC and generates the similar Kc and SRES. After authentication, the Kc is fed into the A5-algorithm together with the Message (M) to encrypt the message before transmitting over the wireless interface. The information is found in [36].

GSM Authentication step-by-step

The goal of the GSM security design is that it has to be as good as that for wire line systems. The authentication is a standard challenge-response mechanism that is based on symmetric cryptography, where the challenge is a RAND and the response is the equivalent SRES. The authentication procedure is described in Figure 16.

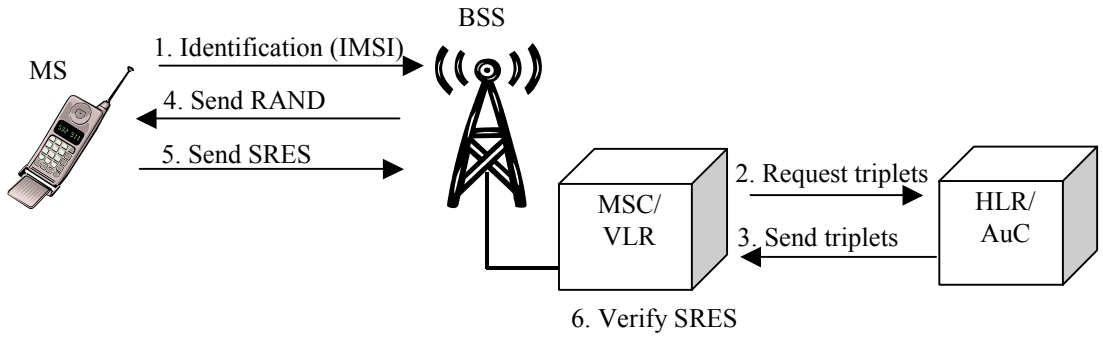


Figure 16: GSM Authentication

1. The authentication procedure begins when the user is identified in the serving network. Identification occurs when the identity of the user (IMSI or TMSI) has been transmitted to the MSC/VLR. The authentication to the circuit switched domain is performed by the MSC/VLR.
2. In order for the MSC/VLR to make this decision they will contact the HLR/AuC and request authentication triplets.

3. The HLR/AuC will calculate new authentication triplets and send them back to the MSC/VLR (between 1 to 5 triplets).
4. The MSC/VLR will choose one of the triplets, and send the corresponding RAND number to the MS as a challenge.
5. When the MS receives the RAND number it will tell the SIM to perform the authentication algorithms that generates the SRES and Kc. This SRES is sent back to the MSC/VLR via the base station.
6. The MSC/VLR compares the SRES received from the MS to the SRES contained in the chosen triplet. If the two match, the MSC/VLR can safely allow the MS access to the network. If the two SRESs do not match, the MS will not get access to the network.

2.4.1 UMTS Authentication

The initial standards for UMTS were completed by 3GPP in April of 1999 and termed Release 1999 (R99). These standards are the basis for a majority of the current commercially deployed UMTS systems. Since R99 the 3GPP has followed up with release 4, 5 and 6, where the last one was completed in 2005. The latest releases have defined features to provide significant network efficiency, performance and functionality advantages. The growing demands for greater speed and capacity have been the focus that has driven the evolution.

The UMTS authentication procedure is very similar to the GSM authentication, the most significant difference is that UMTS provides mutual authentication. In R99 we still have two core networks, the CS domain and the PS domain, but in the implementation of the latest releases the networks will converge towards all IP and only a PS domain. Nevertheless the access security methods have not changed in the new specifications, so the R99 access security will still be utilized. The procedure is described below, and is shown in Figure 17.

The HLR is in UMTS called Home Subscriber Service (HSS), the Serving GPRS Support Node (SGSN) together with the VLR performs the authentication, and the MS is called User Equipment (UE).

1. The authentication procedure begins with the identification of the user. Identification occurs in the same way as in GSM, the user sends the identification to the base station.
2. The base station will contact the SGSN/VLR to decide if the UE should get access to the network. The SGSN will contact the HSS/AuC to get the authentication vectors. An authentication vector is the equivalent of the authentication triplet used in GSM, however the vector consist of five numbers:
   - The random number, RAND
   - The expected signed response, XRES
   - The encryption key, CK
   - The integrity key, IK
   - The network authentication token, AUTN
3. When the SGSN/VLR receives the set of authentication vectors, it selects the first one and stores the rest for later use.
4. The SGSN then sends the RAND and AUTN from the chosen vector to the MS.
5. When the UE receives these, it will tell the USIM to perform the authentication algorithm (shown in section 2.3.2). The result of the algorithms gives the USIM the ability to verify whether the AUTN parameter authenticates the network.
6. In the positive case, the computed RES parameter is sent back to the SGSN.

7. The SGSN will compare the RES with the XRES from the corresponding authentication vector. If the two match, the SGSN allows the MS access to the network.

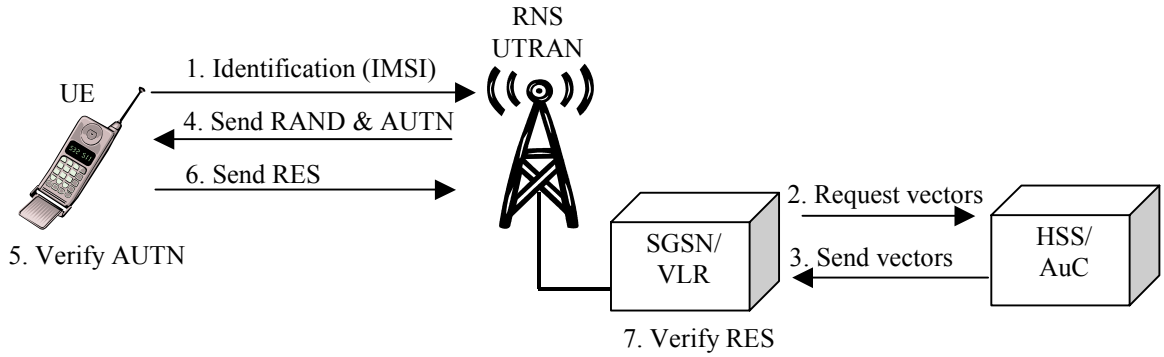


Figure 17: UMTS Authentication

The AUTN parameter gives the MS the ability to check that the home network has authorized the serving network. This verification is essentially based on the value of SQN. For this purpose, two counters are maintained synchronized in the AuC and in the USIM.

In addition to the authentication, the authentication procedure has established keys to provide confidentiality (CK), integrity (IK) and anonymity (AK). After the authentication the MS and the network may begin secure communication. Before the encryption can begin, the communicating parties will agree on the encryption algorithm.

Several known weaknesses in the GSM authentication seem now to be fixed in UMTS. The example set of AKA algorithms provided by the 3GPP is commonly denoted MILENAGE, and is fully open to the public. The main cryptographic strength of MILENAGE is due to its chosen kernel algorithm (AES), which has been tested extensively by many cryptanalysts.

2.5 EAP – Extensible Authentication Protocol

EAP is a general protocol for authentication that supports multiple authentication methods. This section is based on the specification RFC 3748 [37], which obsoletes the old specification in RFC 2284. EAP can be seen as a transport protocol (an envelope) that can accommodate many different authentication methods, e.g. SIM card, password, and public-key authentication. EAP does not require IP, but runs directly over data link layers such as Point-to-Point Protocol (PPP) or IEEE 802. It was designed for use in network access authentication, where IP layer connectivity may not be available. EAP was originally designed for PPP, so the encapsulation of EAP over IEEE 802 has a special specification, the IEEE 802.1X (also referred to as EAP over LAN (EAPoL)).

EAP is a somewhat independent exchange layer. Independent in the context that it does not interfere in how the authentication process is fulfilled, but only gives a way to exchange the needed messages. EAP runs over all 802-protocols, and can be wrapped in all kinds of transport and network protocols.

The EAP framework is peer-to-peer and is based on Requests and Responses. As Figure 18 shows, it is possible to send Success- (when Supplicant is authenticated) or Failure-messages as well. In the data field, method specific messages will be put. Success and failure packets have no data, i.e. the length equals 4. Request/Response packets are explained later in this section. Figure 18 is found in [2].

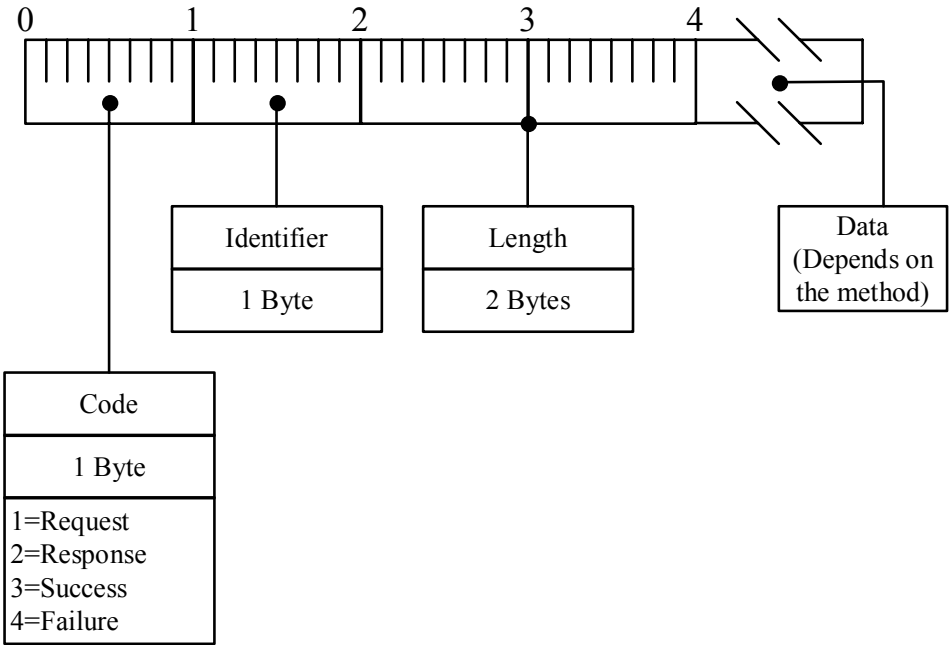


Figure 18: EAP Frame Format

One of the advantages of the EAP architecture is its flexibility. The protocol does not need to be updated to support changes and new versions of the authentication methods. A typical EAP architecture will use an authenticator and a backend authentication server. The authenticator will then act as a pass-through for the authentication with the server, which will only require the authenticator to implement the EAP protocol and not all the different supported methods.

In addition to EAP, there are different similar protocols competing for securely transporting authentication data:

- Lightweight Extensible Authentication Protocol (LEAP) is a proprietary protocol developed by Cisco.
- Extensible Authentication Protocol Transport Layer Security (EAP-TLS) was created by Microsoft and accepted as an IETF standard (RFC2716). EAP-TLS is the *de facto* standard for authentication in 802.11i wireless LANs.
- Extensible Authentication Protocol over LAN (EAPoL or EAPOL) defines the encapsulation techniques used to carry EAP packets between a Supplicant's Port Access Entity (PAE) and an Authenticator's PAE in a LAN environment. I.e., it transports the EAP packets directly by a LAN MAC service.
- Protected Extensible Authentication Protocol (PEAP) is a proprietary protocol, which was developed by Microsoft, Cisco, and RSA Security. Cisco is phasing out LEAP in favour of PEAP.
- Extensible Authentication Protocol Tunnelled Transport Layer Security (EAP-TTLS) is a proprietary protocol developed by Funk Software and Certicom. IETF is considering EAP-TTLS as a new standard.

PEAP and EAP-TTLS both utilize TLS to set up an end-to-end tunnel to transfer user credentials without having to set up a certificate on the client.

EAP Request/Response packets

Almost every EAP-packet is wrapped in a request or response packet. Therefore, these messages have got a special frame format. The request/response frame format is put inside the

data field of the EAP frame format. As you can see in Figure 19, every EAP-method has its own type value and puts the data inside the type-data field. The type-data field may yet again contain special formats.

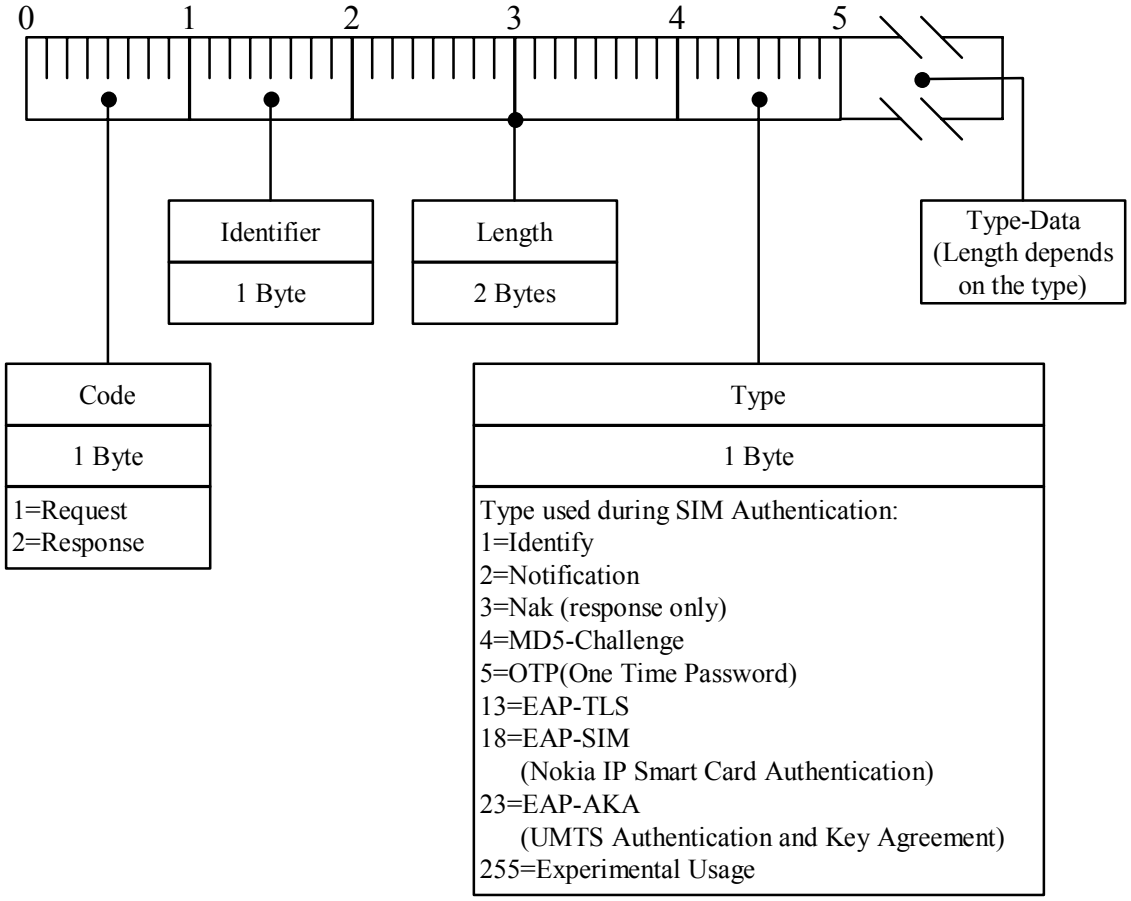


Figure 19: EAP Request/Response Frame Format

Important EAP-methods that can be used with SIM cards will be discussed in the remaining part of this section.

2.5.1 EAP-SIM

EAP-SIM specifies an EAP-based mechanism for mutual authentication and session key agreement using the SIM card. The method is described in RFC 4186 [38], which is the basis for this section.

The EAP-SIM method includes enhancements to the GSM authentication, such as mutual authentication and larger random numbers and ciphering keys. To provide mutual authentication, the EAP-SIM client issues a random number to the network that is used to verify correct identity of the authentication server. To provide a larger ciphering key, the EAP-SIM method uses several authentication triplets to generate several ciphering keys, which are combined into one large key.

EAP-SIM also includes two optional features, the fast re-authentication procedure and identity protection. The fast re-authentication do not make use of the GSM authentication system, hence it offers a more inexpensive authentication. The identity protection uses the same concept as GSM, where temporary identifiers are used to protect the privacy of the user.

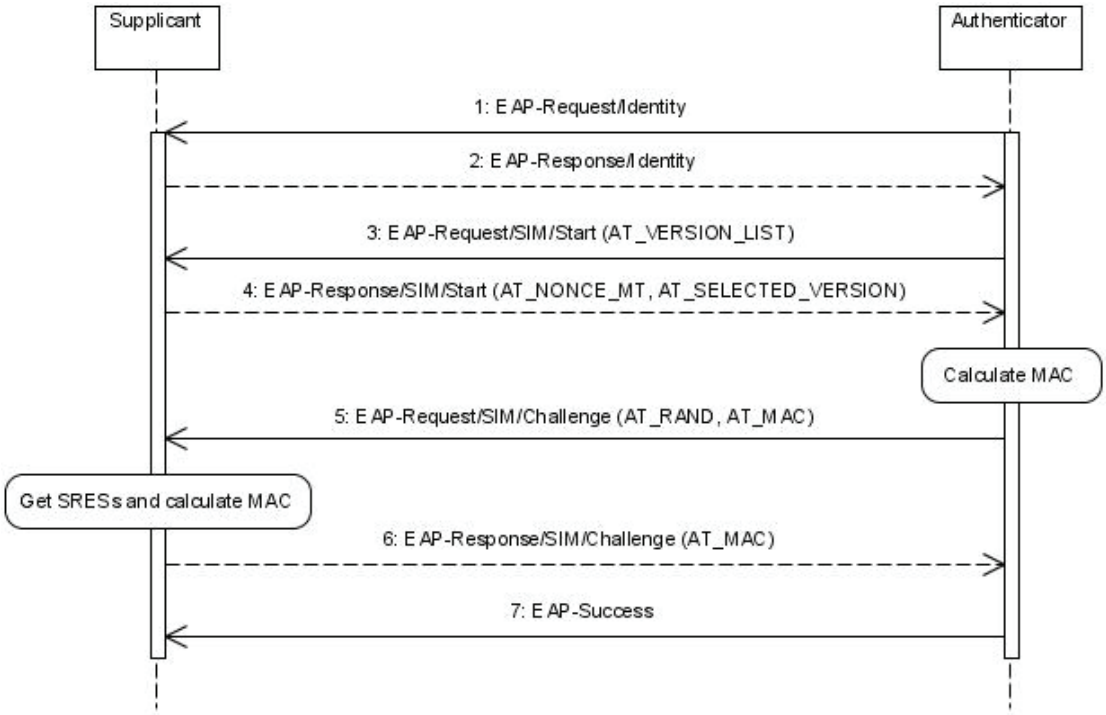


Figure 20: Sequence Diagram – EAP-SIM Full Authentication

Figure 20 shows an overview of the EAP-SIM authentication procedure, the Authenticator typically communicates with an authentication server (not shown). The EAP-SIM authentication procedure starts with an EAP-Request/Identity message issued by the Authenticator. The Supplicant's response includes the identity, either the IMSI or a temporary identity if identity privacy is in effect. Next, the Authenticator sends an EAP-Request/SIM packet containing the EAP-SIM versions supported, the Supplicant respond with a selected version and a random number (NONCE) used in the mutual authentication. The next EAP-Request contains n (n=2 or 3) GSM RANDs, the data field of the attribute AT_RAND, and a message authentication code (MAC) that covers the NONCE-challenge sent from the Supplicant. The Supplicant will run the GSM authentication algorithms (n times) and check the MAC, if something is wrong the Supplicant responds with an error message. If all checks are ok, the Supplicant responds with a new MAC calculated from the n SRESs. The procedure ends with an EAP-Success message or an EAP-Failure message.

Several messages have an attribute (starting with AT_), which is used when performing the mutual authentication. The attributes and the attribute frame format are explained below.

The current EAP-SIM claims to provide 128-bit security, but two successful 64-bit attacks are described in [2], hence EAP-SIM only provides 64-bit protection. The paper gives a couple of solutions to this problem, but the improvements are currently not incorporated to the protocol. S. Patel [2] also shows that EAP-SIM does not provide session independence between different sessions, which will make it possible to re-use the credentials in any communication.

EAP-SIM Attributes

Several EAP-SIM messages have one or more attributes, which have a special format shown in Figure 21. The first byte is the attribute type. Next there is a length field where the value is a multiple of four. Hence, the attribute data should always be a multiple of four bytes. The data filed should be padded with zeroes if less than a multiple of four bytes. Byte number two and three are only used by some attributes, if not used the bytes are reserved for later use and

should be filled with zeroes. Some attributes only have a simple value in byte 2 and byte 3 and no attribute data at all. The attribute data and their structure are explained in [2], but some attributes will be investigated below.

Figure 21 is modified from the specification where there is a value field from byte number two and forward. The modification is done to show more of the normal attribute's field for easier understanding and usage of the specification.

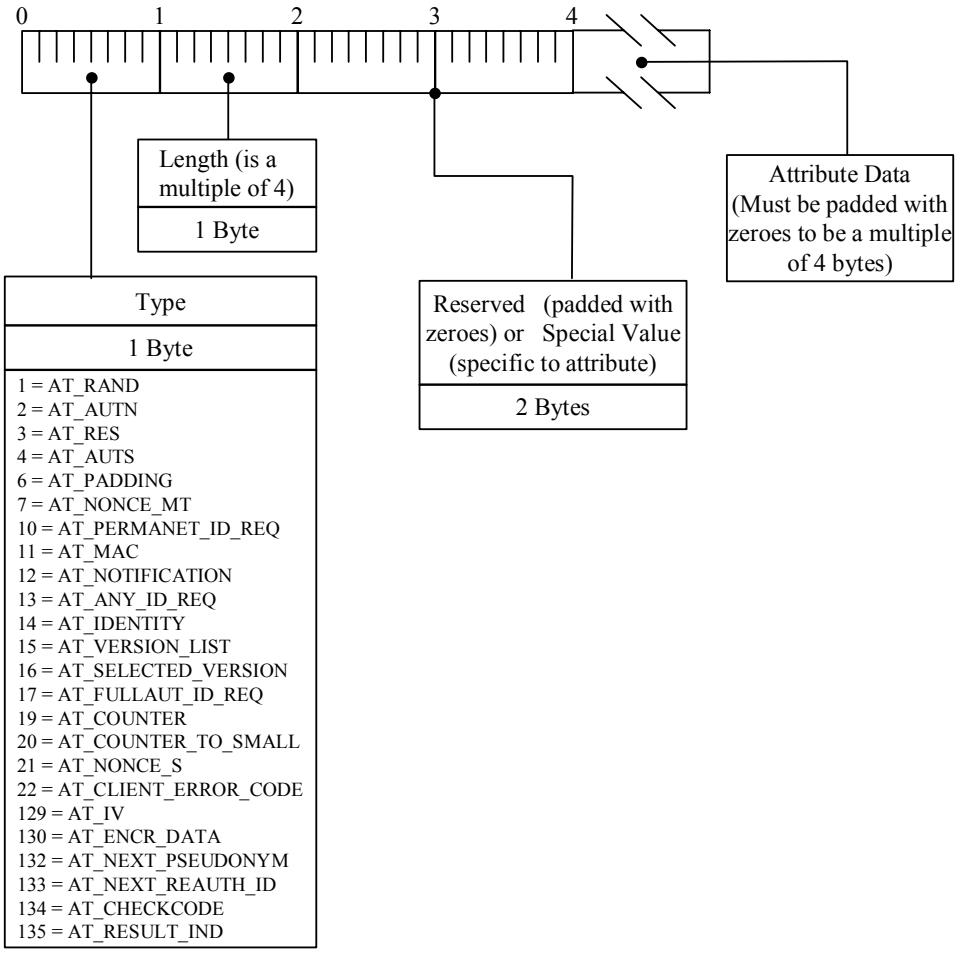


Figure 21: EAP-SIM Attribute Frame Format

AT_IDENTITY

The identity is on the format username@realm (e.g. 123456789@telenor.no). The username is normally the IMSI (permanent username), but pseudonyms or fast re-authentication user names can be used. Fast re-authentication user names are only used during fast re-authentications and are one-time user names. If one of the two alternative user names are used and not recognized, the server will send an AT_PERMANENT_ID_REQ to get the permanent username. The attribute uses the reserved bytes to give the actual length of the identity, which is put in the data field.

AT_VERSION_LIST

This attribute is used in version negotiation. The reserved bytes contain the actual Version List Length (necessary because of potential padding), while the attribute data contains the supported versions (each of length 2 bytes). The supported versions should be listed in the order of preference.

AT_SELECTED_VERSION

This is the response in the version negotiation. Here, the reserved bytes contain the selected version. This indicates the EAP-SIM version that the Supplicant wants to use.

AT_NONCE_MT

The data field of this attribute contains NONCE_MT. NONCE_MT is a 16 bytes long random number generated by the Supplicant, and is used in the process of authenticating the Authenticator. The NONCE_MT should not be re-used. If there are several EAP/SIM/Start rounds, then the Supplicant should use the same NONCE_MT. The reserved bytes are set to zero.

AT_RAND

The Authenticator will send two or three GSM RANDs, each 16 bytes long. The EAP server must obtain new RANDs for each full EAP-SIM Authentication. A RAND value should be used only once. The reserved bytes are set to zero.

AT_MAC

AT_MAC has 16 bytes long MAC value put in the data field. The MAC is an HMAC-SHA1-128 (see section 2.2.3 and 5.2.3) keyed hash value, which uses a key, K_aut (see section 7 of [38]), as the secret key put into the algorithm. The MAC is calculated over the whole EAP packet concatenated with optional message specific data, with the exception that the data field of the AT_MAC is set to zero. The reserved bytes are set to zero.

Example of MAC calculated in response of n*RANDs:
MAC = HMAC-SHA1-128 (EAP-packet concatenated with n*SRESs)

2.5.2 EAP-AKA

Similar to EAP-SIM, EAP-AKA specifies an EAP-based mechanism for mutual authentication and session key agreement using the Authentication and Key Agreement (AKA) mechanism used in the Third Generation (3G) mobile networks (UMTS and CDMA2000). AKA is based on symmetric keys and runs on any type of smart cards, e.g. USIM card. The method is described in the Internet-Draft [39].

The EAP-AKA authentication procedure message exchange (see Figure 22) is shorter than the EAP-SIM procedure shown in Figure 20. Because the UMTS authentication already incorporates mutual authentication, the procedure do not need the extra request and response to exchange the random number from the client. Other differences are in the authentication vector and that the user runs an AKA algorithm instead of the GSM algorithms. The EAP-AKA includes three optional features, fast re-authentication, identity protection and protected result-indication.

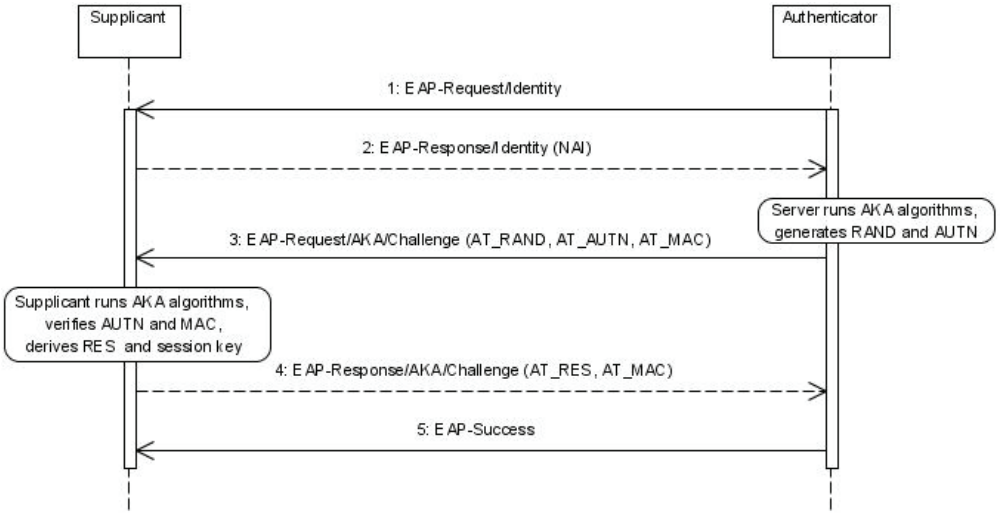


Figure 22: Sequence Diagram – EAP-AKA Full Authentication

Just as EAP-SIM, EAP-AKA has its own attributes, but here is the Message Authentication Code (MAC) used to keep the message's integrity while the network authentication token (AUTN) is used to authenticate the Authenticator and RES used to authenticate the Supplicant. The NAI is the Network Access Identifier, and is used to identify the Supplicant. The NAI can for instance be composed of a username (or IMSI) and a realm divided by @. The username portion identifies the supplicant within the realm.

2.5.3 EAP-TLS

The Transport Layer Security (TLS) is the IETF latest version of the Secure Socket Layer (SSL) protocol. TLS provides a way to use certificates for both user and server authentication and for dynamic session key generation. The specification RFC 2716 [40] modifies TLS to be used over EAP. When we use TLS as the authentication protocol, the user and the server do not need to have a pre-shared secret to authenticate each other. The advantage is that TLS will establish such a pre-master key at run time. The private key needed by the user can be stored on a smart card, i.e. SIM card. The disadvantages are that the use of digital certificate requires a public key infrastructure, which can have a high maintenance cost, and the use of public key cryptography is more computational expensive.

2.5.4 EAP-POTP

The Protected One-Time Password Protocol (EAP-POTP) is an EAP method for use with OTP tokens. The method is currently a new Internet draft [41] created by the RSA Security group to overcome the poor EAP support for OTP algorithms [42]. EAP-POTP is designed to be independent of particular OTP algorithms and provides mutual authentication, generation of keying material and does not rely on tunneling. It assumes that the authentication server and the client share a secret key (also called "seed") and a PIN code. The authentication is performed through the creation of OTPs. The other options of using OTP over EAP is to use an existing EAP tunneling method, e.g. EAP-TTLS, EAP-PEAP, and then use a specialized OTP method "inside" the tunnel.

2.5.5 EAP-SC

The EAP Smart Card protocol (EAP-SC) provides a framework for interfacing with smart cards within the EAP context. This means that EAP-SC will encapsulate other EAP methods, such as EAP-SIM, EAP-AKA and EAP-TLS. The EAP-SC is specified in [43], and is a proposal to get one EAP type for all smart cards. EAP messages that are sent and received

to/from smart card handlers are encapsulated in EAP-SC packets. The EAP-SC framework is a three-layer model, shown in Figure 23.

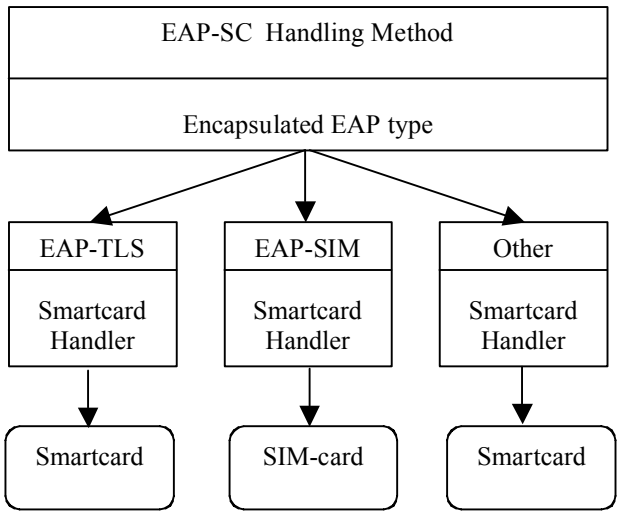


Figure 23: EAP-SC Framework Model

2.6 RADIUS

Remote Authentication Dial In User Service (RADIUS) is a standardized protocol for Authentication, Authorization, and Accounting (AAA) that is being used by the majority of remote dial-in users, routers, and VPNs to centralize network authentication of remote accesses. The protocol RADIUS is defined in the RFC 2865 [44], and is based on the client/server-model, where you have a network access point (probably a (Network Access Server (NAS)) that acts like a client and a RADIUS server. The client is responsible for passing information from the user to the RADIUS servers, and then acting on the response that is returned. RADIUS servers are responsible for acting on the request from the client and authenticating the user. The communication can be between two RADIUS servers as well, for instance when a RADIUS server acts as a proxy client for another RADIUS server.

The chosen transport protocol for RADIUS is the User Datagram Protocol (UDP). According to the specification UDP was chosen because it simplifies the server implementation and it does not need retransmission or ACK-overhead. If the request to a RADIUS-server fails the client can just try an alternate server. Nevertheless using UDP instead of for example TCP, miss one central feature: with UDP we must artificially manage retransmission timers to the same server.

The RADIUS server can support a variety of methods to authenticate a user. Features can vary, but most can look up the users in text files, LDAP (Lightweight Directory Access Protocol) servers, various databases, etc. Authentication between the RADIUS server and the client is performed using a shared secret, which should be at least 16 octets long. The shared secret is used by the client when performing the authentication of the user, e.g. in user-password authentication the shared secret is used in the hashing of the user password. RADIUS is incapable of protecting against real-time wiretapping of an authenticated session, but generation of unique request packets protects it against a wide range of authentication attacks. The client receives a request packet with an Authenticator-field (16 octets long), this number is sent through a one-way MD-5 hash function together with the shared secret, and then XORed with the password. The result value is sent back to confirm the user.

RADIUS Packet format

The officially assigned port number for RADIUS is 1812. The maximum radius packet will fit inside one UDP packet. Figure 24 illustrates the data format of one RADIUS packet. The Code field identifies the type of packet. During authentication packets sent from the client use the Access-Request code, and packets sent from the server uses Access-Accept, Access-Reject or Access-Challenge. The identifier field is used to match the request and response; the client starts with 0 and increments the identifier by one for each request. The length field specifies the total length of the packet including the header fields. The authenticator field is used to authenticate the reply from the RADIUS server. An authenticator value is chosen by the client and should be unpredictable and unique over the lifetime of a secret. In the response from the server the authenticator is the value of the MD5 hash calculated over: MD5 (Code, ID, Length, Request Authenticator, Attributes, Shared Secret).

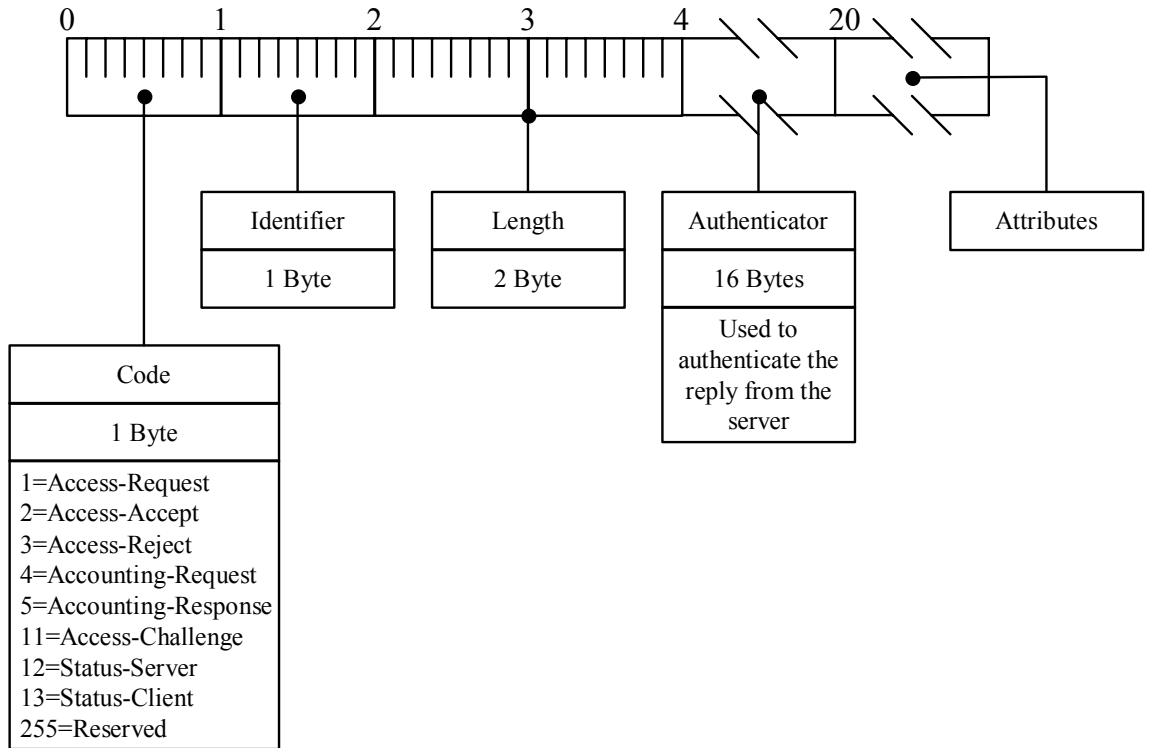


Figure 24: RADIUS Packet Format

The remaining part of the packet contains a list of attributes, where the length field indicates the end of the list. It is the attributes that carry the actual authentication information. The Figure 25 shows the format of the attributes. The type field indicates the type of attribute; the length field specifies the length of this attribute including the type and length field. The figure only shows the attributes needed during the SIM authentication, and will be described next.

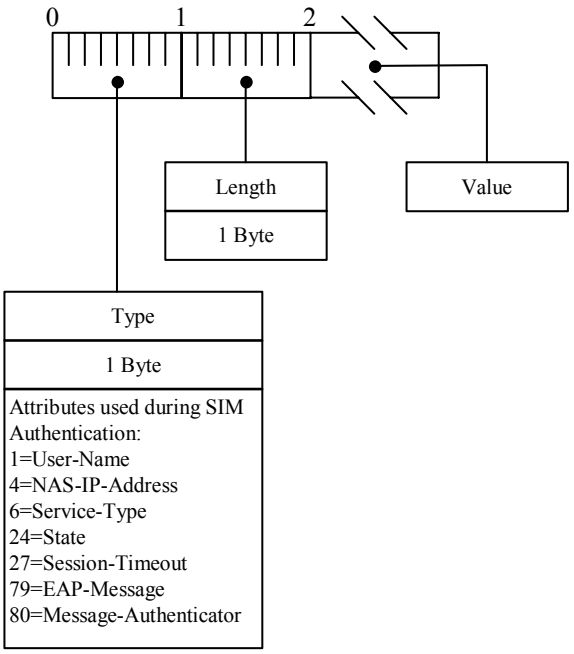


Figure 25: RADIUS Attribute Format

The User-Name attribute indicates the name of the user to be authenticated, and the value field includes the UTF-8 encoded characters of the name. In the NAS-IP-Address attribute the client includes the identifying IP address. The Service-Type attribute classifies the type of service the user has requested. If the RADIUS server includes the State attribute in an Access-Challenge the client need to include the same value in the reply that follows. The Session-Timeout attribute sets the maximum number of seconds before the session will timeout.

RFC 3579 [45] specifies additional attributes for providing EAP support within RADIUS, it introduce the two new attributes EAP-Message and Message-Authenticator. The EAP-Message attribute is assigned the type number 79. The EAP-message is placed in the value field; if the EAP-message is larger than 253 bytes it can be divided into multiple EAP-Message attributes and the RADIUS server will concatenate them at reception. If a RADIUS server receives an EAP messages that it does not understand it should return an Access-Reject.

All RADIUS packets that include the EAP-Message attribute must also include the Message-Authenticator attribute, which is assigned type number 80. This attribute is used to sign the message to prevent spoofing attacks. The Message-Authenticator is an HMAC-MD5 [46] checksum calculated over the entire Access-Request packet using the shared secret as the key: HMAC-MD5 (Type, Identifier, Length, Request Authenticator, Attributes). During the calculation of the RADIUS authenticator the Message-Authenticator should be considered to be sixteen bytes of zeroes. When the RADIUS server or client receives a packet with the Message-Authenticator included it must calculate the correct value and silently discard the packet if it does not match the value sent. Packets missing the Message-Authenticator attribute should also be silently discarded.

2.6.1 DIAMETER

RADIUS was initially deployed to provide dial-up PPP and terminal server access. With the introduction of new access technologies, everything has become more complex and increased in density, putting new demands on AAA protocols. RADIUS has improved in many ways, but DIAMETER has been developed with these new technologies in mind.

DIAMETER is the planned successor of RADIUS. DIAMETER evolves the RADIUS scheme by adding new features, such as the ability to ask for additional logon information beyond the basic authentication. For example, DIAMETER uses the initial logon and authentication to identify a user, and then can continue to query the user for additional information that might be used to strengthen the authentication or to give the user access to additional systems. DIAMETER is planned to work just as well in local AAA as in roaming situations.

The RADIUS is a client/server protocol, while DIAMETER is based on a peer-to-peer model. Therefore, it is difficult, e.g., to implement server initiated messages in RADIUS without extensions to the protocol. See Table 3 for a more detailed comparison between RADIUS and DIAMETER. The information on the comparison is found in [47].

DIAMETER runs over transport protocols (Stream Control Transmission Protocol (SCTP) or TCP) that prevent packet loss. Packet loss may translate into revenue loss. In the event that a transport failure is detected with a peer, it is necessary for all pending request messages to be forwarded to an alternate agent, if possible. This is commonly referred to as failover. DIAMETER supports application layer acknowledgments and failover algorithms to provide well-defined failover behaviour.

Table 3: Comparison of RADIUS/DIAMETER

Property	**RADIUS**	**DIAMETER**
Failover	Not defined (depends on implementation)	Supported
Transmission-level security (authentication and integrity)	Defined only for response packets. In extension, IPSec and IKE support is optional.	IPSec support is mandatory and TLS support is optional
Reliable transport	UDP. Reliability varies between implementations.	TCP/SCTP. Reliable.
Agent Support (proxies, redirects and relays)	Not defined. In extension, server-initiated messages are optional.	Supported.
Audit-ability	Not supported.	Supported / optional. Data object security is defined in extension.
Transition support	Not defined	Supported in extension
Capability negotiation	Not supported	Supported
Peer discovery and configuration	Manual configuration	Dynamic
Roaming support	Not suitable for global roaming in open environments due to lack of security.	Secure and scalable roaming support.

All data delivered by the protocol is in the form of an AVP (Attribute Value Pairs). It has been developed several DIAMETER application, but common for all of them is that they have to use the underlying services defined in RFC 3588 [48]. This gives a well-defined service where all underlying services are alike.

2.7 Mobile Terminal Communication Interfaces

2.7.1 Bluetooth

The original idea (from Ericsson) was to connect mobile phones to other devices (e.g. PDAs) without using cables. Together with Toshiba, IBM, Intel and Nokia, Ericsson formed a Special Interest Group (SIG) to develop a standard for interconnecting communication devices and accessories using short-range, low-power, inexpensive radios. The project was called Bluetooth. IEEE 802.15.1 is a standard where IEEE has reviewed and provided a standard adaptation of the Bluetooth Specification v1.1 Foundation's MAC (L2CAP, LMP and Baseband) and PHY (Radio). The two standards are not completely similar. See the IEEE 802.15.1 homepage [49] or SIG's Official Bluetooth Membership Site [50] for more information.

The main differences in the newer versions of Bluetooth are faster connections and enhanced synchronization and error detection, the current version of SIG Bluetooth is v2.0. The Bluetooth Radio Frequency operates in the unlicensed 2.4 GHz ISM (industrial, scientific, and medical) band (79 channels of 1 MHz each). Bluetooth uses Frequency Hopping Spread Spectrum (FHSS) to combat interference and fading. The 2.4 GHz band is also used by the IEEE 802.11-standards and because of Bluetooth's faster hop frequency, Bluetooth is more likely to destroy 802.11-packets than the other way round. The hopping pattern may be adapted to exclude a portion of the frequencies that are used by the interfering devices. Adaptive frequency hopping is included in newer versions and may improve co-existence with non-hopping systems. Bluetooth v2.0 supports Basic Rate mode (1 Mb/s) and Enhanced Data Rate (EDR) mode (2 or 3 Mb/s).

In Bluetooth, one device provides synchronization reference and is known as the master. All other devices are called slaves, and a group of master and slaves is called a piconet. One device may be included in two piconets and works as a bridge. An interconnected collection of piconets is called a scatternet.

In order to use Bluetooth, a device must be able to interpret certain Bluetooth profiles. The profiles define possible applications. The Serial Port Profile (SPP) and SIM Access Profile (SAP) are the most interesting profiles for this thesis.

The SPP [51] defines the protocols and procedures that shall be used for serial cable emulation. Figure 26 shows the SPP's protocol stack and describes how the application may communicate with each other. The Baseband, LMP, and L2CAP correspond to layer 1 and 2 of the OSI layer. RFCOMM provides a transport protocol for the serial port emulation, and is the Bluetooth adaptation of GSM TS 07.10 [52]. The SDP is the Bluetooth Service Discovery Protocol. The Serial Port emulation layer provides an API to the applications on the higher layer. Examples of applications on top of SPP are SIM Access Profile, Hands-Free Profile and Dial-up Networking Profile. The SAP allows devices such as car phones, with built in GSM transceivers, to connect to a SIM card in a phone with Bluetooth. A more detailed description will be given later in this section.

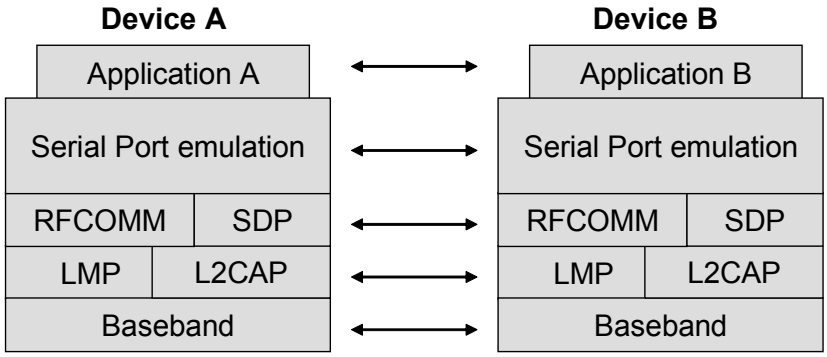


Figure 26: Serial Port Profile - Protocol Stack

The general information about Bluetooth is found in chapter 4.6 of [53] and in [54].

Security

The Bluetooth specification defines a security model based on three components: authentication, encryption and authorization. In addition, three security modes are defined, enforcing different levels of security:

Security Mode 1: No active security enforcement
Security Mode 2: Service level security
Security Mode 3: Device level security (link level)

Security mode 2 is the mode used for the majority of Bluetooth devices, hence the service decides whether security is required or not. In security mode 3, security procedures are initiated during the setup of a Bluetooth link. If security measures fail, the link setup will fail.

Pairing or also called bonding, is the procedure of a Bluetooth device authenticating another device, and is dependent on a shared authentication key. If the shared key is missing the devices need to create a new key. This involves generation of an initialisation key and an authentication key. The initialisation key is based on user input (passkey up to 128 bits), a random number (128 bits) and the Bluetooth address (48 bits) of one of the devices. When the pairing process is completed, the devices have authenticated each other and share the same key. The devices can now store the key for future use, which means that they can authenticate each other without the use of a passkey.

There have been some attacks on Bluetooth, and there will probably be more in the future. Papers on cracking passkeys, viruses over Bluetooth, and using Bluetooth enabled mobile phones to locate computer equipment in cars are examples on Bluetooth related attacks. But often, there are not the Bluetooth protocols that contain the main security flaws, but bad implementations of them. There is more information and links on Bluetooth security concerns on Wikipedia[8], using “Bluetooth” as a search string.

SAP – SIM Access Profile

The SIM Access Profile [55] allows a Bluetooth enabled device to access the SIM card contained in another Bluetooth device. SAP builds on the well-defined interface between the mobile and the subscription module. The profile is for instance used in the Nokia 616 car kit, which in the SIM Access Mode retrieves SIM credentials so the car phone itself doesn't require a separate SIM card.

[8] Wikipedia, http://wikipedia.org/

The profile defines two different roles:

- The SIM Access Server, which has direct access to the SIM card (mobile) and acts as a SIM card reader.
- The SIM Access Client, which is connected via a Bluetooth link and communicate with the SIM card using the SAP server.

In order to ensure secure communication between the client and the server the specification identifies several mandatory security demands, e.g. the passkeys must be 16 digits at least and the link between the client and server shall be encrypted using Bluetooth baseband encryption with a minimum encryption key length of 64 bits.

The profile consists of different features, where each feature includes one or more procedures. Each procedure consists of one or more messages that are exchanged between the client and the server. The SAP messages are transported on an RFCOMM link, and are formatted as shown in Figure 27. The figure also shows the most important MsgIDs.

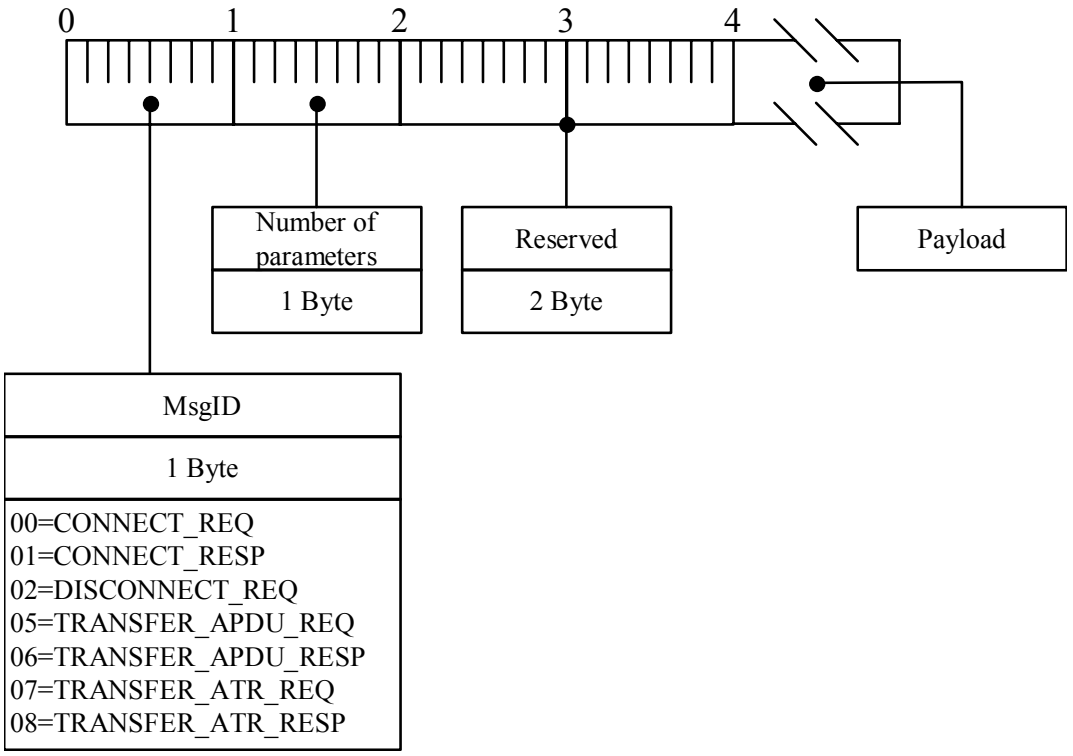


Figure 27: SAP Message Format

The payload of the SAP message consists of parameters, where each parameter is formatted as shown in . Important to notice is that the parameter shall be a multiple of four bytes where the extra bytes are padded with zeroes. The parameter length field gives the length of the parameter value without the padding. The most important ParameterIDs are also shown in Figure 28.

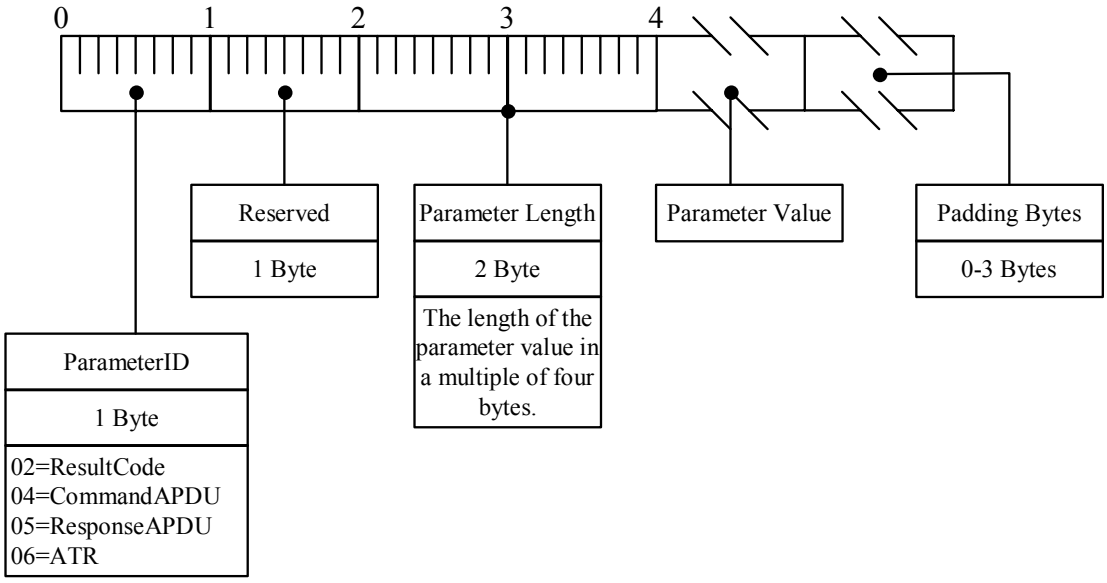


Figure 28: SAP Parameter Format

2.7.2 Java Platform, Micro Edition (Java ME)

Java ME, formerly known as J2ME, is a collection of Java APIs for the development of software for embedded devices market, i.e. mobile phones, Personal Digital Assistants (PDA) and other small devices. Java ME provides an environment with the Java virtual machine for the applications running on these devices.

The Java ME defines many different profiles, which each defines a different set of APIs. Many of the profiles are optional and embedded devices come with different features and functionalities.

SATSA (JSR-177)

The Security and Trust Services APIs (SATSA) provide smart card access and cryptographic capabilities to applications running on small devices. The profile defines four different APIs:

- SATSA-APDU: Communication with smart card using APDUs.
- SATSA-JCRMI: Communication with smart card using a remote object protocol.
- SATSA-PKI: Support for digitally signing of data and certificates.
- SATSA-CRYPTO: General-purpose cryptographic API that supports message digests, digital signatures, and ciphers.

For the communication with the SIM card the SATSA-APDU API is the most interesting.

Bluetooth (JSR-82)

JSR-82 is the official Java Bluetooth API and is independent of both the stack and the Bluetooth hardware. JSR-82 consists of two packages:

- javax.bluetooth: The core Bluetooth API
- javax.obex: The object exchange API

The javax.blueooth package provides an API for device discovery and service discovery. It also makes it possible to setting up L2CAP and RFCOMM connections.

2.7.3 Hayes Command Set (AT-Commands)

Most modem manufactures and newer mobile phones implements the Hayes standard, which has been a de facto standard for modem communication. The Hayes Command Set includes many different commands, where each command must begin with the two letters *AT*. This is the reason the standard often is called AT-Commands. GSM 07.07 [56] specifies an extended AT commands set for use with GSM Mobile Equipment. The command prefix +C is reserved for Digital Cellular by ITU-T (ITU-Telecommunication). The basic structure of an AT command line is shown in Figure 29.

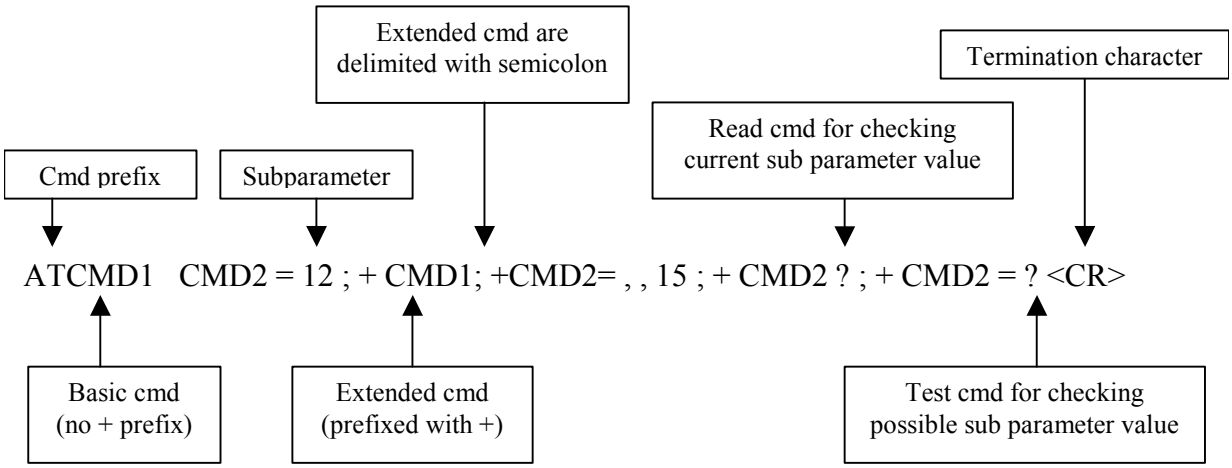


Figure 29: Basic Structure of an AT Command Line

GSM 07.07 describes four commands that can be useful in the communication with the SIM:

- +CIMI is used to request the international mobile subscriber identity (IMSI)
- +CPIN is used to enter ME passwords, which are needed before any other functionality of the ME can be used (e.g. SIM PIN, PUK).
- +CSIM, Generic SIM Access can be used to access all data on the SIM.
- +CRSM, Restricted SIM Access can be used to give a computer easier, but more restricted access to the SIM compared with the +CSIM command.

The most useful command with regards to this thesis is the +CSIM command:

CSIM

The command syntax is: AT+CSIM=<length>,<command>

The ME will send the <command> as it is to the SIM, and the response will be returned in the same manner. This command allows an application to send APDUs to the SIM in the format described in the section 2.3.4.

2.8 Summary

This chapter describes a lot of different technologies, protocols, etc. Everything that is elaborated here can be related to the SIM, cryptography, or authentication, whether it is transport, security, or authentication protocols. The idea is to form a synthesis of the background information into a design and an implementation of a solution, and at the same time yield a basis for a technical and security related discussion. It is also important to know what kind of related technologies exist.

The background should also be used as a reference when reading the rest of the thesis together with the reference list, if a deeper understanding is wanted.

3 Analysis

In this chapter the Generic SIM Authentication System will be presented. The first section will define the requirements to the system, both functional and non-functional. Then different use cases are presented to further illustrate the different requirements of the system. The last section analyzes different interaction diagrams, collaboration diagrams and sequence diagrams, which all together should help identifying the components and necessary interfaces in the system. The different diagrams shown in this chapter were created using a modeling tool called Visual Paradigm for UML [57].

3.1 Requirements

In this section the requirements to the authentication system will be presented. They are separated into two categories, functional requirements and non-functional requirements. In this context the term, "support", means that the requirement have to be implemented by the system but is not necessarily in use by every execution of the system.

3.1.1 Functional Requirements

Functional requirements capture the intended behavior of the system. This behavior may be expressed as services, tasks or functions the system is required to perform. (Bredemeyer [58])

The functional requirements to the system are described in Table 4 below.

Table 4: Functional Requirements

#	Functional requirement
FR1	Support the following strong end-to-end authentication mechanisms: • One-Time Password • GSM/UMTS Authentication
FR2	The system must as a minimum use two-factor authentication
FR3	The User must authenticate to the SIM (by using PIN)
FR4	Support mutual authentication between: • Supplicant and Authenticator • Authenticator and Authentication Server
FR5	Have well defined mechanisms to prevent access to the network using a stolen unit
FR6	Support different SIM readers: • Mobile Phone / PDA • Smart Card and SIM readers
FR7	Support pseudonyms and temporary identities

3.1.2 Non-Functional Requirements

Non-functional requirements include constraints and qualities. Qualities are properties or characteristics of the system that its stakeholders care about and hence will affect their degree of satisfaction with the system. Constraints are not subject to negotiation and, unlike qualities, are (theoretically at any rate) off-limits during design trade-offs. (Bredemeyer [59])

The previous section introduced the functional requirements to the system. Equally important are non-functional requirements, which are described in the table below. Non-functional requirements address quality issues associated with both the executing behavior of the system and the development work. With some of the non-functional requirements to the system trade-offs have to be made between run-time and development-time qualities. For example, performance and modifiability may be in tension in the system design, so that the users' desire for performance may have to be traded off against the goal of having a more maintainable architecture.

Table 5: Non-Functional Requirements

#	Non-Functional requirement
NFR1	Usability: • Only one SIM and PIN is needed • Support existing SIM cards • All authentication mechanisms supported by one SIM • Support mobile phone as a SIM reader • The Supplicant can be used on any arbitrary PC • User friendly interface • Support different Operating Systems and Internet browsers
NFR2	Use open standards and well-known APIs to ensure better portability
NFR3	The user should trust the authentication system, including: • Well-known authentication mechanisms • Authenticator must be a trusted source
NFR4	Anonymity: • Only the minimum of information needed should be transferred between entities. (e.g. a weather service should only receive a postal code or similar) • Should not be able to track users movement
NFR5	Support following types of system: • Web-based (WWW) services (i.e. Internet Accounts) • Distributed Client-Server services in general (i.e. SIP) • Physical Services (i.e. Access Control, Payment terminal)
NFR6	Scalability: • The authentication system must not be a choke point • Scale gracefully in a distributed environment (load balancing between servers)
NFR7	Reliability, mechanisms to ensure: • Detection of system failures • Notification when a failure occurs • High-quality exception handling
NFR8	Have redundancy on databases to ensure scalability and prevent loss of data
NFR9	The system must have high uptime, to ensure availability

NFR10	The system should not cost anymore than existing systems: • Cost-effective for the manufactures/providers • Low price for the users
NFR11	Performance: • Short response time • Low bandwidth usage, i.e. minimize transactions
NFR12	Extensibility: • Easy to add new functionality • Support changes to platforms and environments

3.2 Use Cases

A use case defines a goal-oriented set of interactions between external actors and the system under consideration. Actors are parties outside the system that interact with the system. (Bredemeyer [58])

To further illustrate the different requirements of the system, this section will present use cases. The use case structure will be graphically summarized in a use case diagram according to the Unified Modeling Language (UML) 2.0 [60]. The different use cases will also be described textual according to a use case template proposed by Derek Coleman [61], with some minor modifications. The template is shown and described in the appendix.

3.2.1 High Level Use Case

The first use case (see Figure 30) shows the high level view of the authentication system with 5 use cases and two actors. The actors are named Supplicant and Service Provider (SP), where the Supplicant needs to be authenticated and the SP is the one offering a service that requires authentication. The Supplicant is actually split into a Service Supplicant, communicating with the Authenticator and hence the Authentication Server, and a SIM Supplicant communicating with the SIM.

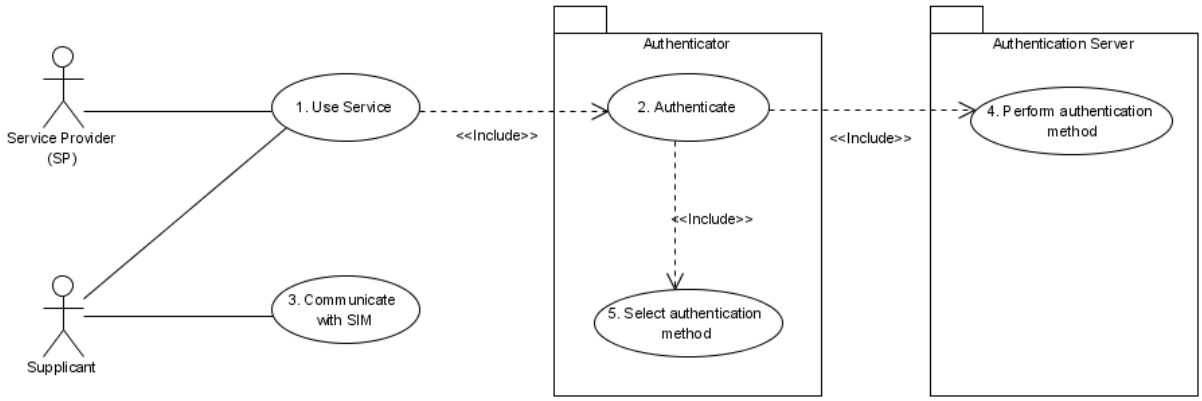


Figure 30: Use Case – High-level, Showing Actors and Systems

Use case (UC) 1 and use case 2 are described in Table 6 and Table 7, while the other use cases are elaborated in chapter 3.2.2, where lower-level use cases are described.

Table 6: Use Case – Use Service

Use Case	1. Use Service **History** created 16.01.06, modified 10.02.06
Description	A Supplicant needs access to a service that requires strong end-to-end authentication.
Actors	Supplicant Service Provider (SP)
Assumptions	The SP needs to be connected to an Authenticator. The Supplicant needs to provide the correct credentials
Steps	1. The Supplicant tries to access the service 2. The SP redirects the request to the Authenticator 3. The Authenticator starts the authentication process (Start UC2: "Authenticate") 4. The Authenticator vouches for the Supplicant 5. The SP gives access to the service
Variations	#4. The authentication process fails #5. The SP denies access to the service
Issues	Should the SP double check the approval with the Authenticator?

Table 7: Use Case – Authenticate

Use Case	2. Authenticate **History** created 16.01.06, modified 25.01.06
Description	A Supplicant is being authenticated. The Supplicant will select an authentication method and he/she will be authenticated according to this procedure.
Actors	
Assumptions	The Supplicant needs to be connected to an Authenticator. The Supplicant needs to provide the correct credentials.
Steps	1. The Supplicant is redirected from the SP to the Authenticator with a minimum-security level. 2. Start UC4: "Perform authentication method" 3. Redirect Supplicant to the SP with an assertion vouching for the Supplicant.
Variations	#1 No security level is provided, assume the minimal security level #3 Authentication fails, RETURN failure
Issues	Is the Supplicant redirected to the SP with an approval, or does the Authenticator notify the SP directly? (#3)

3.2.2 Lower Level Use Cases

In this section use case 3-5 will be elaborated in more detail, introducing new use cases. The lower level use cases are used to get a deeper understanding of the communication and components needed to be able to authenticate a Supplicant. The different use case shown in the figures will be described to clarify its tasks. A generic SIM authentication system should be able to implement numerous authentication methods, and to illustrate this GSM, PKI, and OTP authentication is shown.

Use Case 3 – Communicate with SIM

UC3 (shown in Figure 31) is linked to the SIM Supplicant and has as a primary goal to retrieve information needed from the SIM. The Supplicant should be able to get access to the card (by verifying the PIN), retrieve IMSI and the needed credentials to be authenticated.

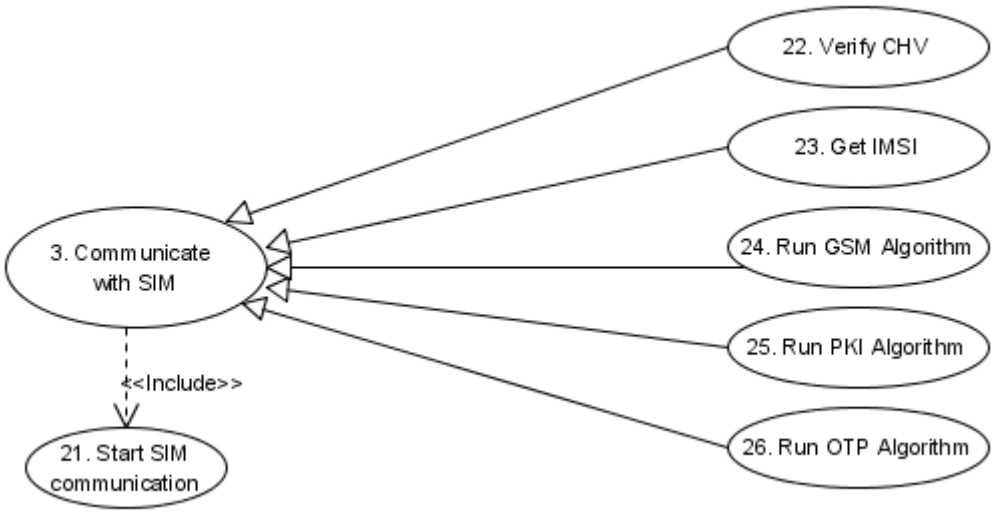


Figure 31: Use Case – Communicate with SIM

As Figure 31 shows, UC3 (see Table 8) has 5 generalizations showing what functionality is needed (described in Table 10 to Table 14). In addition, a way to initiate the communication with the SIM is needed (UC21 is described in Table 9).

Table 8 : Use Case – Communicate with SIM

Use Case	3. Communicate with SIM **History** created 16.01.06, modified 16.01.06
Description	The SIM supplicant communicates and performs the necessary commands with the SIM card.
Actors	Supplicant
Assumptions	The Supplicant needs access to the SIM card with a SIM reader.
Steps	1. The authentication is started, need to read the IMSI 2. Start UC28': "Start SIM communication" 3. IF Supplicant not verified to SIM THEN 3.1 Start UC22: "Verify CHV" ELSEIF Need the IMSI THEN 3.2 Start UC23: "Get IMSI" ELSEIF GSM authentication THEN 3.3 Start UC24: "Run GSM algorithm" ELSEIF PKI authentication THEN 3.4 Start UC25: "Run PKI algorithm" ELSEIF OTP authentication THEN 3.5 Start UC26: "Run OTP algorithm" 4. RETURN success
Variations	#2 Communication with the SIM card fails, RETURN failure #3.1 CHV authentication fails, RETURN failure #4 The request fails, RETURN failure
Issues	Consider temporary identities instead of IMSI

Table 9: Use Case – Start SIM Communication

Use Case	21. Start SIM communication **History** created 16.01.06, modified 16.01.06
Description	Start the communication with the SIM card, using a supported communication method (i.e. smart card reader or mobile phone reader)
Actors	Supplicant
Assumptions	The Supplicant has specified the communication method. The SIM card is connected with the SIM reader.
Steps	1. The communication with SIM card is started 2. Read the ATR to check if the card is supported 3. RETURN success
Variations	#2 The card in the reader is not supported, return error message
Issues	

Table 10: Use Case – Verify CHV

Use Case	22. Verify CHV **History** created 16.01.06, modified 16.01.06
Description	Verification of the PIN code (CHV) with the SIM
Actors	Supplicant
Assumptions	The UC21: "Start SIM communication" was successful
Steps	1. Ask the Supplicant for the PIN code 2. Send the "Verify_CHV" APDU, with the PIN code 3. The Supplicant is authenticated to the card, RETURN success
Variations	#1 The PIN protection is disabled on the SIM card, enable the PIN on the card and continue. #3 Wrong PIN code, after 3 attempts need PUK 3.1 Wrong PUK code, after 10 attempts card "closed"
Issues	Should the software ask for PUK code also?

Table 11: Use Case – Get IMSI

Use Case	23. Get IMSI **History** created 16.01.06, modified 20.01.06
Description	Read the IMSI from the SIM card, the IMSI is stored in one of the files on the card
Actors	Supplicant
Assumptions	The UC21: "Start SIM communication" was successful The UC22: "Verify CHV" was successful
Steps	1. Select the file DF_GSM 2. Select the file EF_IMSI 3. Read the binary data of the EF_IMSI file 4. Convert the binary IMSI to decimal IMSI 5. Return IMSI
Variations	#RETURN failure
Issues	On what format is the IMSI handled in the system? (Binary or decimal?)

Table 12: Use Case – Run GSM Algorithm

Use Case	24. Run GSM Algorithm **History** created 16.01.06, modified 20.01.06
Description	Run the GSM algorithm to produce the SRES/Kc needed in the GSM authentication
Actors	Supplicant
Assumptions	The UC21: "Start SIM communication" was successful The UC22: "Verify CHV" was successful Received challenge (RAND) from the Authenticator
Steps	1. Select the file DF_GSM 2. Send the "Run_GSM_algorithm" APDU, with a RAND value 3. Get the response from the command; SRES and Kc 4. RETURN SRES/Kc
Variations	#RETURN failure
Issues	

Table 13: Use Case – Run PKI Algorithm

Use Case	25. Run PKI Algorithm **History** created 16.01.06, modified 16.01.06
Description	Run the PKI algorithm used in the PKI authentication
Actors	Supplicant
Assumptions	The UC21: "Start SIM communication" was successful The UC22: "Verify CHV" was successful Received challenge from the Authenticator
Steps	1. Send the "Run_PKI_algorithm" APDU 2. Get the response from the command 3. Return the response
Variations	# RETURN failure
Issues	How does the SIM card perform the PKI algorithm? How to communicate with a Java Card Applet?

Table 14: Use Case – Run OTP Algorithm

Use Case	26. Run OTP Algorithm **History** created 16.01.06, modified 16.01.06
Description	Run the OTP algorithm to get a one-time password
Actors	Supplicant
Assumptions	The UC21: "Start SIM communication" was successful The UC22: "Verify CHV" was successful
Steps	1. Send the "Run_OTP_algorithm" APDU 2. Get the response from the command, OTP 3. Return the OTP
Variations	# RETURN failure
Issues	How does the SIM card perform the OTP algorithm? How to communicate with a Java Card Applet?

Use Case 4 – Perform Authentication Method

UC4 (see Figure 32) illustrates more details of the authentication procedure performed by the different authentication methods. These are actions performed at the authentication server and are initiated by a chosen authentication method. Hence, the different methods are generalizations of UC4: "Perform authentication method". The Authentication Server performs the authentication procedure decided by the Authenticator.

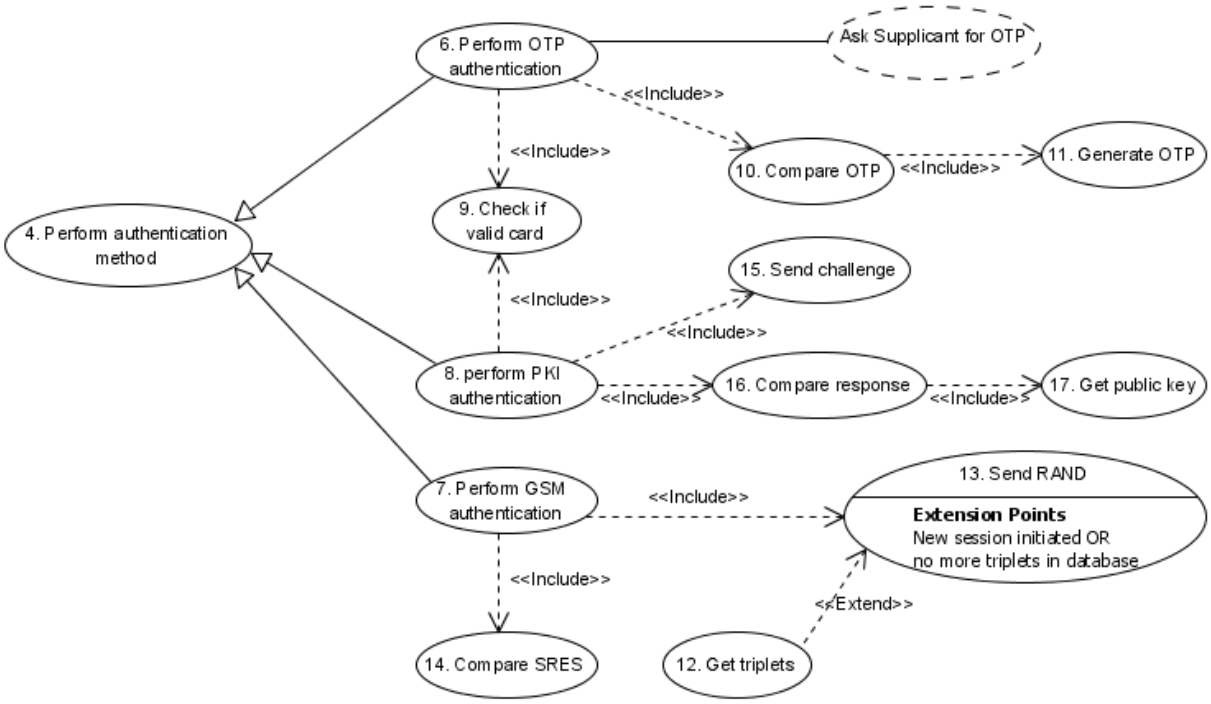


Figure 32: Use Case – Perform Authentication Method

UC4 is described in details in Table 15, and its generalizations and their use cases (UC6-UC17) are described in the subsequent tables (Table 16-Table 27). The PKI and OTP authentication methods need to check if the card is valid, while this is done automatically when asking for triplets in the GSM authentication method.

Table 15: Use Case – Perform Authentication Method

Use Case	4. Perform authentication method **History** created 16.01.06, modified 16.01.06
Description	A Supplicant is being authenticated. The Supplicant will select an authentication method and he/she will be authenticated according to this procedure.
Actors	
Assumptions	The Supplicant needs to be connected to an Authenticator. A minimum authentication level is presented.
Steps	1. Invoked when the authentication process starts at the Authenticator. 2. Start UC5: "Select authentication method" 3. IF OTP authentication THEN 3.1 Start UC6: "Perform OTP authentication" ELSEIF GSM authentication THEN 3.2 Start UC7: "Perform GSM authentication" ELSEIF PKI authentication THEN 3.3 Start UC8: "Perform PKI authentication" 4. RETURN success.
Variations	#4 The Authentication fails, RETURN failure.
Issues	

Table 16: Use Case – Perform OTP Authentication

Use Case	6. Perform OTP authentication **History** created 16.01.06, modified 16.01.06
Description	The Authenticator performs an OTP authentication to approve a Supplicant.
Actors	
Assumptions	The Authenticator needs a shared secret with the Supplicant. The Authenticator needs access to the GSM network.
Steps	1. The chosen authentication method is OTP 2. IF new session THEN 2.1 Start UC9: "Check if valid card" 3. Ask Supplicant for OTP 4. Wait for response from the Supplicant 5. Check if valid OTP (Start UC10: "Compare OTP") 6. RETURN success
Variations	#2 Stolen or invalid card, authentication fails #4 No response (timeout), authentication fails #6 The authentication fails, RETURN failure.
Issues	What happens when authentication fails?

Table 17: Use Case – Perform GSM Authentication

Use Case	7. Perform GSM authentication **History** created 16.01.06, modified 22.01.06
Description	The Authenticator performs a GSM authentication to approve a Supplicant.
Actors	
Assumptions	The Authenticator needs access to the GSM network.
Steps	1. The chosen authentication method is GSM 2. Send Challenge (Start UC13: "Send RAND") 3. Wait for the response from the Supplicant 4. Compare response (Start UC14: "Compare SRES") 5. RETURN success
Variations	#3 No response (timeout), RETURN failure. #5 The authentication fails, RETURN failure.
Issues	Should EAP and RADIUS be considered in this context? What happens when authentication fails?

Table 18: Use Case – Perform PKI Authentication

Use Case	8. Perform PKI authentication **History** created 16.01.06, modified 16.01.06
Description	The Authenticator performs a PKI authentication to approve a Supplicant.
Actors	
Assumptions	The Authenticator needs the Supplicant's public key. The Authenticator needs access to the GSM network.
Steps	1. The chosen authentication method is PKI 2. IF new session THEN 2.1 Start UC9: "Check if valid card" 3. Send challenge (Start UC15: "Send challenge") 4. Wait for response from the Supplicant 5. Validate the response (Start UC16: "Compare response") 6. RETURN success
Variations	#2.1 Stolen or invalid card, authentication fails #4 No response (timeout), authentication fails #6 The authentication fails, RETURN failure
Issues	Should EAP and RADIUS be considered in this context?

Table 19: Use Case – Check If Valid Card

Use Case	9. Check if valid card **History** created 17.01.06, modified 17.01.06
Description	Checks with the GSM network if the SIM is valid based on the Supplicant's IMSI.
Actors	
Assumptions	The Authenticator needs the Supplicant's IMSI. A new session is initiated. Connection with the GSM Network.
Steps	1. Ask the GSM network: "Is this IMSI valid?" 2. Wait for response from the GSM Network 3. RETURN valid SIM
Variations	#3 RETURN not valid SIM
Issues	What kind of message is sent to the GSM network? Is it standardized?

Table 20: Use Case – Compare OTP

Use Case	10. Compare OTP **History** created 16.01.06, modified 16.01.06
Description	The Authenticator performs an OTP authentication to approve a Supplicant, and needs to compare the OTP with an OTP generated by the Authenticator.
Actors	
Assumptions	The Authenticator needs a shared secret with the Supplicant.
Steps	1. The Authenticator receives an OTP from the Supplicant 2. Start UC11: "Generate OTP" 3. Compare values 4. RETURN success
Variations	#4 The values are not equal, RETURN failure
Issues	

Table 21: Use Case – Generate OTP

Use Case	11. Generate OTP **History** created 17.01.06, modified 17.01.06
Description	An OTP is generated to be used in a comparison with a Supplicant's OTP sent to the Authenticator during an OTP authentication.
Actors	
Assumptions	The Authenticator has a shared secret with the Supplicant.
Steps	1. Generate an OTP based on the shared secret. 2. Return value.
Variations	
Issues	What if the Authenticator doesn't have a shared secret with the Supplicant? Should this be checked here or in UC9/UC6?

Table 22: Use Case – Send RAND

Use Case	13. Send RAND **History** created 17.01.06, modified 17.01.06
Description	The Authenticator performs a GSM authentication to approve a Supplicant. A RAND is sent as a challenge to the Supplicant.
Actors	
Assumptions	Valid triplets are available in database. Has Supplicant's IMSI.
Steps	1. Initiated by send challenge. 2. Read triplet from database (use IMSI) 3. Send RAND to Supplicant
Variations	
Issues	How should the SRES/RAND be stored when read from the database?

Table 23: Use Case – Get Triplets

Use Case Extension	12. Get triplets **extends** 13. Send RAND **History** created 16.01.06, modified 16.01.06
Description	Get triplets from the GSM network based on the Supplicant's IMSI
Condition	A new session is initiated OR no more triplets in database
Steps	#2 IF no valid triplets THEN 2.1 Get triplets from GSM network 2.2 Store returned triplets in database 2.3 Read triplet from database (use IMSI)
Variations	#2.2 No triplets returned from GSM network, RETURN failure
Issues	

Table 24: Use Case – Compare SRES

Use Case	14. Compare SRES **History** created 17.01.06, modified 17.01.06
Description	The Authenticator performs a GSM authentication to approve a Supplicant. The Supplicant has been sent a challenge and the Authenticator has received a response (SRES) that has to be compared.
Actors	
Assumptions	The Authenticator has the RAND/SRES-pairs necessary.
Steps	1. Compare response 2. Get the SRES belonging to the sent challenge (RAND) 3. Compare the two SRESs 4. RETURN success
Variations	#4 The SRES are not equal, RETURN failure
Issues	

Table 25: Use Case – Send Challenge (PKI)

Use Case	15. Send challenge **History** created 17.01.06, modified 17.01.06
Description	The Authenticator performs a PKI authentication to approve a Supplicant. A challenge is being sent.
Actors	
Assumptions	Supplicant needs to have the Authenticator's public key
Steps	1. Generate a one-time challenge and sign it with the Authenticator's private key. 2. Send the challenge to the Supplicant
Variations	
Issues	Must choose an algorithm?

Table 26: Use Case – Compare Response

Use Case	16. Compare Response **History** created 17.01.06, modified 17.01.06
Description	The Authenticator performs a PKI authentication to approve a Supplicant. A challenge is being sent and the response is received for approval.
Actors	
Assumptions	Received response from the Supplicant
Steps	1. Get the public key (Start UC17: "Get public key") 2. Decrypt the response using the Supplicant's public key 3. Compare the response with sent challenge 4. RETURN success
Variations	#1 Failure received, RETURN failure #4 Decrypted response is different from sent challenge, RETURN failure
Issues	

Table 27: Use Case – Get Public Key

Use Case	17. Get public key **History** created 17.01.06, modified 17.01.06
Description	The Authenticator performs a PKI authentication to approve a Supplicant. A challenge is being sent and the response is received waiting to be approved. The Supplicant's public key is needed.
Actors	
Assumptions	The Authenticator has the Supplicant's certificate.
Steps	1. Get the Supplicant's certificate 2. Extract the Supplicant's public key 3. Return public key
Variations	#1 No certificate stored, RETURN failure
Issues	What if no certificate is stored, contact CA?

Use Case 5 – Select Authentication Method

The Authenticator needs to have a list of available authentication methods. By receiving a minimum-security level required by the service, the Authenticator should be able to return a list of allowed authentication methods. Figure 33 shows this process. Finally, the Supplicant receives a list from the Authenticator.

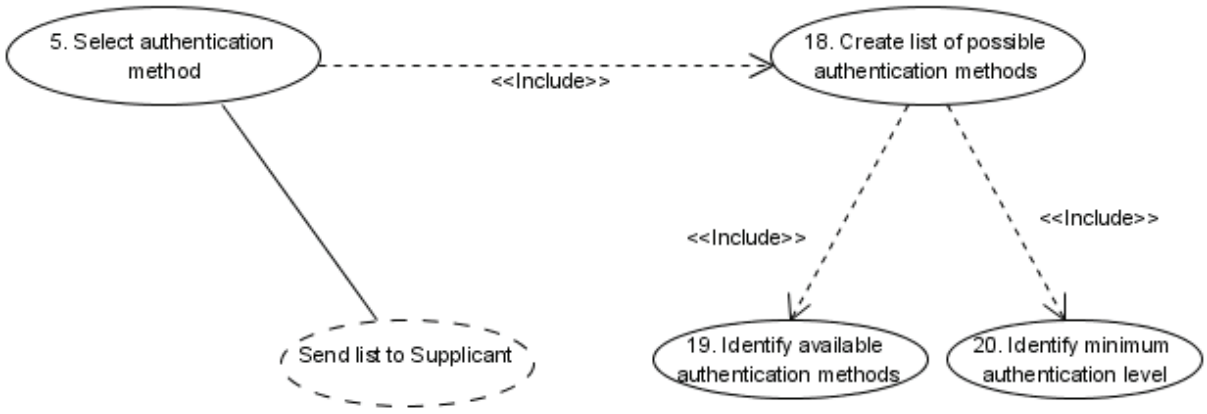


Figure 33: Use Case – Select Authentication Method

The different use cases shown in Figure 33 are described below in Table 28 to Table 31.

Table 28: Use Case – Select Authentication Method

Use Case	5.Select authentication method **History** created 16.01.06, modified 25.01.06
Description	The Authenticator creates a list of possible authentication methods and sends this to the Supplicant.
Actors	
Assumptions	
Steps	1. Start UC18: "Create list of possible authentication methods" for this session. 2. Send list to Supplicant
Variations	#2 If no available authentication method; send error message to Supplicant
Issues	

Table 29: Use Case – Create List of Possible Authentication Methods

Use Case	18. Create list of possible authentication methods **History** created 16.01.06, modified 25.01.06
Description	The preparation of the list of possible authentication methods
Actors	
Assumptions	The Authenticator needs access to an authentication server. It also needs information about the minimum-security level required by the service.
Steps	1. Start UC19: "Identify the available authentication methods" 2. Start UC20: "Identify minimum authentication level" 3. Create a list of possible authentication methods for this service 4. RETURN list
Variations	#2 None of the methods provides high enough level of security, RETURN failure
Issues	

Table 30: Use Case – Identify Available Authentication Methods

Use Case	19. Identify available authentication methods **History** created 16.01.06, modified 25.01.06
Description	Contact the Authentication Server to get a list of supported authentication methods, or use a cached list.
Actors	
Assumptions	The Authenticator needs access to an authentication server.
Steps	1. IF cached list need renewal THEN 1.1 Send request to the Authentication Server 1.2 Store the responded list in the cache register 2. RETURN list
Variations	
Issues	How often should we update the list? (Manually or automatically)

Table 31: Use Case – Identify Minimum Authentication Level

Use Case	20. Identify minimum authentication level **History** created 16.01.06, modified 25.01.06
Description	Identifies the security level needed by the service
Actors	
Assumptions	The Authenticator needs information about the security level needed by the service.
Steps	1. Map the minimum security level of the service to a minimum authentication level (method) 2. RETURN level
Variations	
Issues	How to map between security levels and authentication levels?

3.3 Interaction Diagrams

Interaction diagrams, collaboration diagrams and sequence diagrams, are included to help identify components and necessary interfaces. Section 3.3.1 shows a collaboration diagram (also called communication diagram) describing the high-level communication performing a generic authentication. The messages sent in the collaboration diagram will be elaborated in chapter 3.3.2, where they are described with the help of sequence diagrams.

3.3.1 Collaboration Diagrams

"A collaboration describes a structure of collaborating elements (roles), each performing a specialized function, which collectively accomplish some desired functionality. Its primary purpose is to explain how a system works and, therefore, it typically only incorporates those aspects of reality that are deemed relevant to the explanation. Thus, details, such as the identity or precise class of the actual participating instances are suppressed." (UML 2.0 [62])

Figure 34 shows a high-level collaboration diagram describing a generic authentication procedure. It contains all the major components needed to authenticate and authorize a user to a specific service. The User tries to access a service, which initiates an authentication process where the Supplicant represents the user client. When the Supplicant is authenticated it notifies the Service, which optionally may double-check with the Authenticator if the authentication was a success. This is not shown in the figure.

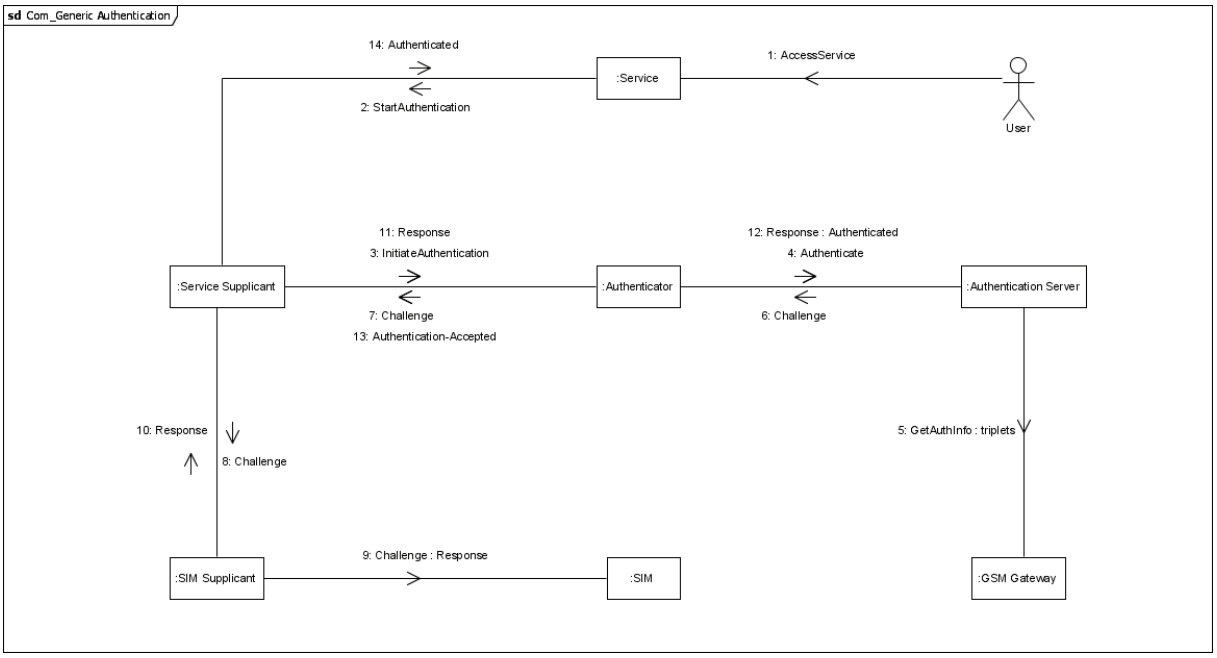


Figure 34: Collaboration Diagram – Generic SIM Authentication System

This generic authentication procedure will be elaborated in several sequence diagrams in the next chapter. Since this requires specific messages, the GSM authentication will be used as an example.

3.3.2 Sequence Diagrams

"The most common kind of Interaction Diagram is the Sequence Diagram, which focuses on the Message interchange between a number of Lifelines (represent an individual participant in the Interaction). A sequence diagram describes an Interaction by focusing on the sequence of Messages that are exchanged." (UML 2.0 [62])

In this section a successful authentication will be described using the GSM authentication as an example. Sequence diagrams showing failures, together with OTP authentication sequence diagrams can be found in Appendix A.1. The specific messages shown in this section are based on EAP between the Service Supplicant and the Authenticator, and EAP over RADIUS between the Authenticator and the Authentication Server. RADIUS is chosen because it is most widespread, but it could just as easily be performed with for instance DIAMETER. The EAP protocol is extended with EAP-SIM in the GSM authentication specific messages, according to the EAP-SIM specification.

Figure 35 shows a full authentication from the User accesses the service until he/she has been authenticated and given access to the service. The authentication is initiated when a user tries to access a service. If the service requires authentication it will start the Service Supplicant by sending the "2. StartAuthentication"-message with the minimum-security level required by the service. The Service Supplicant then contacts the Authenticator to request a list of possible authentication methods that satisfies the given security level. When the Authentication Server has authenticated the Service Supplicant, the Service Supplicant notifies the Service that it is authenticated. The Service may then check with the Authenticator if the Supplicant is Authenticated (message 8).

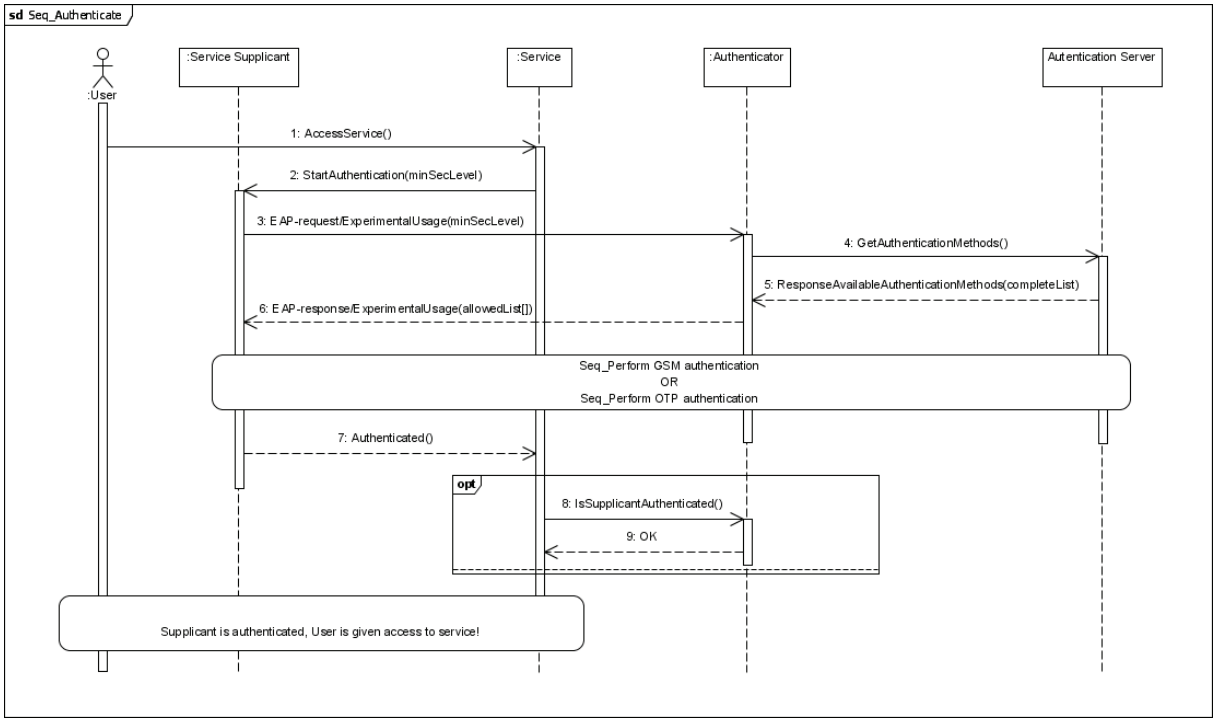


Figure 35: Sequence Diagram – A Full Authentication

In the middle of Figure 35 there is a reference to two other sequence diagrams, namely "Seq_Perform GSM Authentication" (Figure 36) and "Seq_Perform OTP Authentication" (see Appendix A.1). In this spot there is possible to have other authentication methods as well, i.e. PKI.

Figure 36 shows the authentication according to the EAP-SIM authentication process. It is initiated by a special message that tells the Authenticator that the EAP-SIM (GSM) method has been chosen. This message uses a subtype of EAP called "Experimental usage" and its content is made specifically for this system. It was needed to be able to initiate the correct authentication method. The Authenticator and the Service Supplicant belonging to this system are the only ones able to interpret the "Experimental Usage" -messages.

The Authenticator requests the identity of the Supplicant, and when the Supplicant sends the identity (IMSI), the Authenticator forwards the message to the Authentication Server wrapped in a RADIUS request message. From this point on it is the Authentication Server that controls the authentication process and the Authenticator only wraps and unwrap messages. The EAP messages sent are according to the EAP-SIM full authentication described in Figure 20 in section 2.5.1.

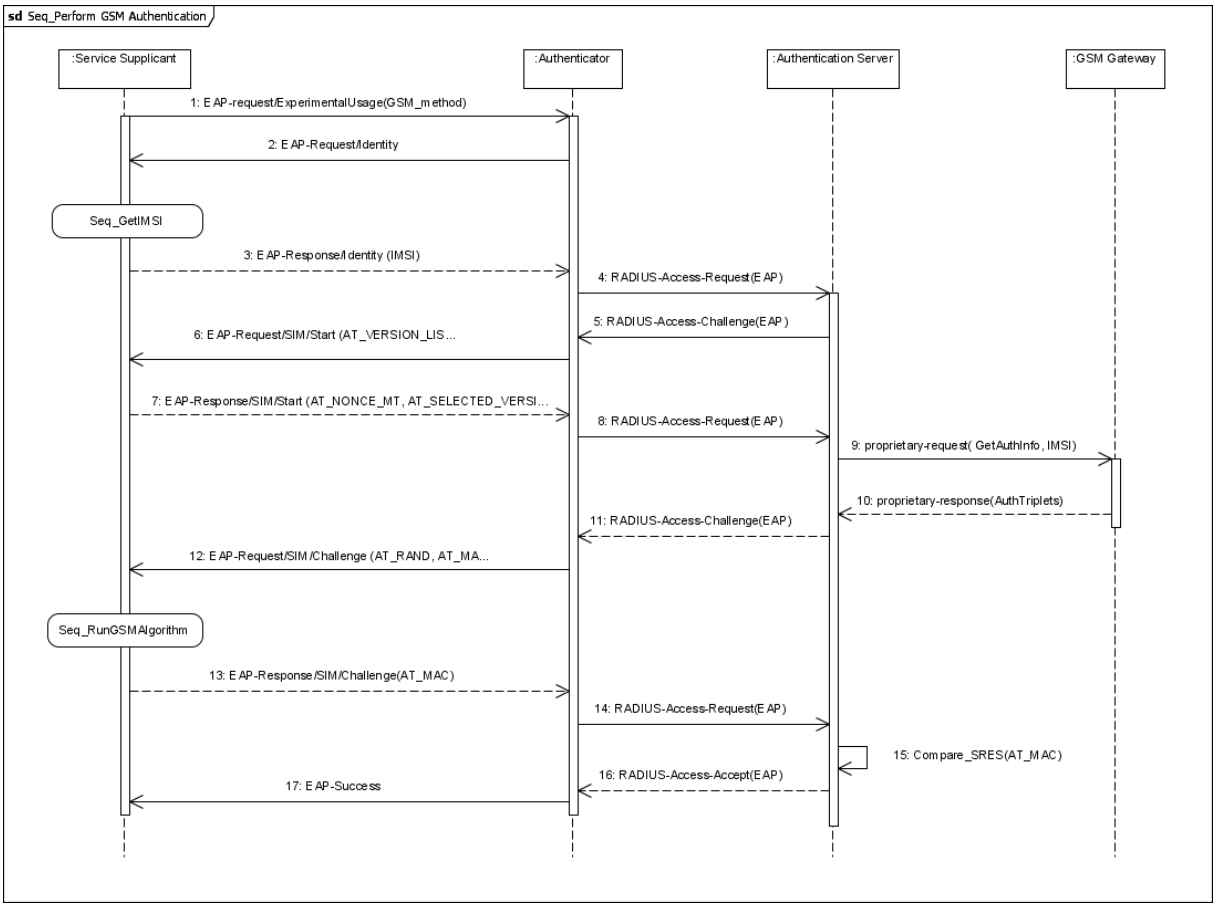


Figure 36: Sequence Diagram – Perform GSM Authentication

There are two references to other sequence diagrams in Figure 36. "Seq_GetIMSI" (Figure 37) shows how to retrieve the IMSI from the SIM and "Seq_RunGSMAlgorithm" (Figure 38) gets the Signed Response (SRES) on the RAND sent from the Authentication Server. "Seq_RunGSMAlgorithm" is actually performed several times depending on how many

RANDs the attribute AT_RAND contains. The Authentication Server chooses the number of RANDs used in every transaction.

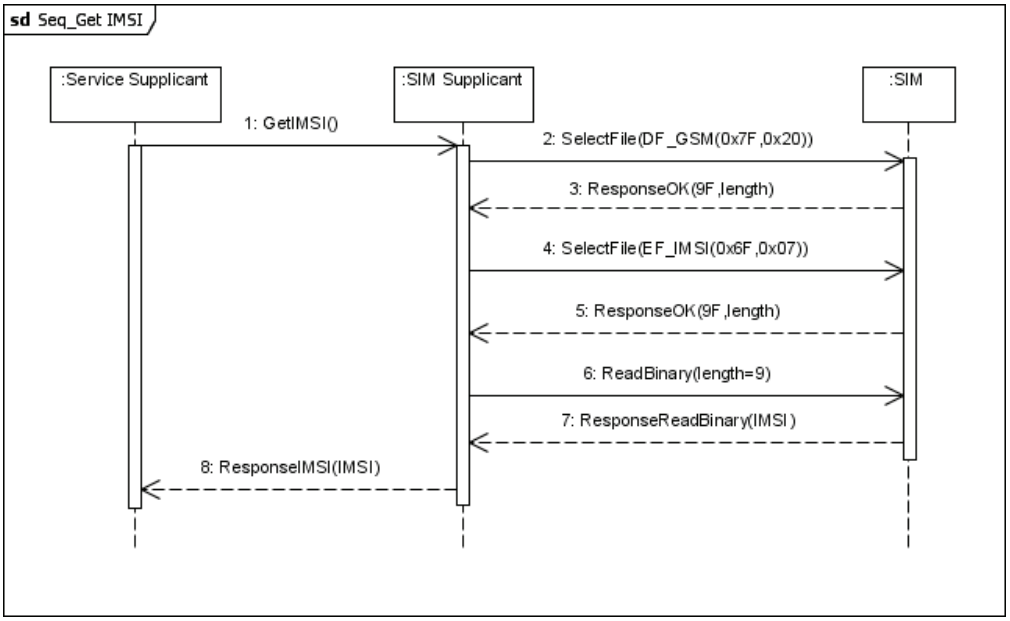


Figure 37: Sequence Diagram – Get IMSI

The Service Supplicant communicates with the external entities and uses information gathered from the SIM. Since the retrieval of SIM credentials often require an intricate sequence of APDUs, this responsibility has been given to the SIM Suppliant. The SIM Supplicant provides a simple interface to the Service Supplicant. The division of the SIM and Service Supplicant also makes it possible to implement the Service Supplicant and the SIM Supplicant on different parts of the system (i.e. SIM Supplicant located on the mobile phone and the Service Supplicant on a PC).

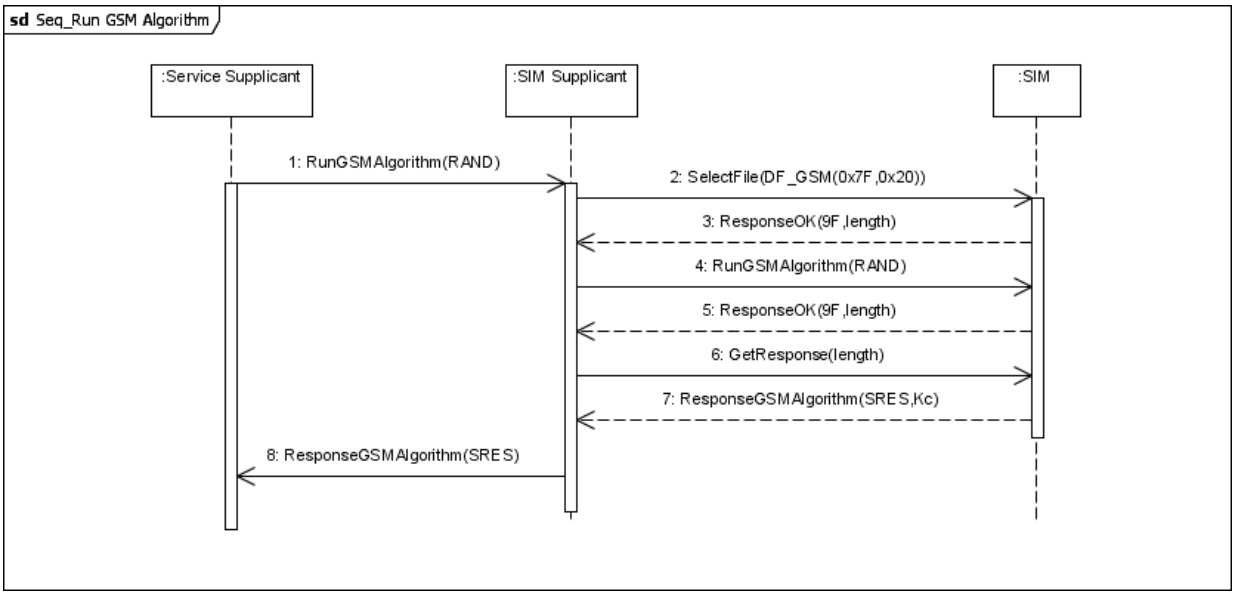


Figure 38: Sequence Diagram – Run GSM Algorithm

The messages sent from the SIM Supplicant to the SIM are APDUs according to GSM 11.11 [18]. There are only a few allowed messages, and as can be seen there is for instance no easy way to retrieve the SRESs and Kcs. The interfaces used to communicate with the SIM over APDUs will be explored in the next chapter.

3.4 Summary

The User (Supplicant) communicates with the SIM through a SIM Reader or a mobile phone. An interface for each authentication mechanism is needed towards the SIM. To be able to handle the different messages, a SIM Supplicant is needed (if an application is needed on the mobile phone) and a Service Supplicant is needed to perform the authentication and translate messages into the correct format towards the Authenticator.

The Generic SIM Authentication System has been thoroughly analyzed through requirements, use cases and interaction diagrams. This chapter has given a deeper understanding and at the same time provided a foundation for the design phase (chapter 4). On the basis of the interaction diagrams it is possible to identify the components of the system, and most of the interfaces are already clarified throughout the messages being sent.

4 Design

In the design phase, the different software components will be elaborated and mapped into different packages and classes.

4.1 Components

"*A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. A component defines its behavior in terms of provided and required interfaces. As such, a component serves as a type whose conformance is defined by these provided and required interfaces (encompassing both their static as well as dynamic semantics). One component may therefore be substituted by another only if the two are type conformant.*" (UML 2.0 [62])

The purpose of this section is to present the different components that are needed to build an architecture performing a general authentication using SIM. This section will clarify which of the components that already exist and which that do not currently exist. Figure 39 presents the components of the Generic SIM Authentication System. This is a high-level presentation and the different components will be elaborated in the section 4.1.1.

Small circles illustrate the interfaces provided by a component, a socket illustrates the required interface, and dependency of an interface or a component is depicted with a stapled arrow.

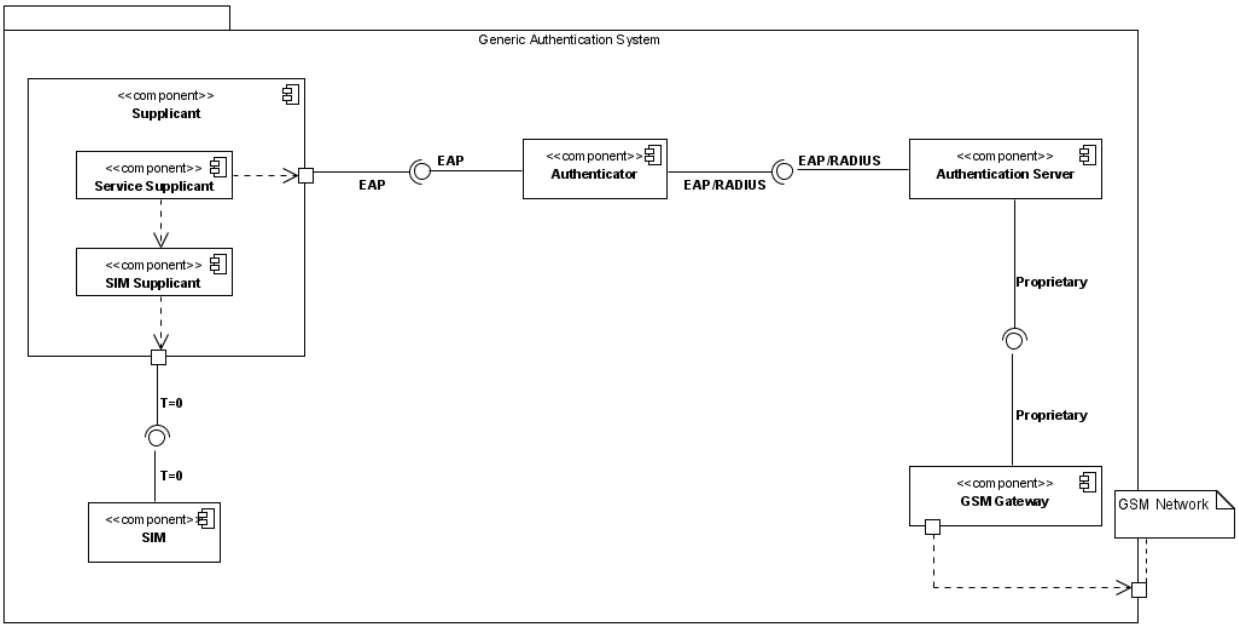


Figure 39: Component Diagram – High-level

The Supplicant consists of both a Service Supplicant and a SIM supplicant, where the Service Supplicant communicates with the Authenticator over EAP and the SIM Supplicant communicates with the SIM with APDUs over T=0. The Authenticator communicates with the Authentication Server over RADIUS and uses RADIUS' interfaces. The communication between the Authentication Server and the GSM gateway depends on the implementation of the GSM Gateway used by the Authentication Server. The GSM Gateway communicates with the GSM Network over SS7. The GSM Gateway will not be explored any more since there are no interfaces in the GSM Gateway that are required in the implementation.

4.1.1 Components – Low-Level

This section describes the different components more thoroughly. A short description is given in relation with the component's figure.

Component – Supplicant
Status: None of the components exist

The Supplicant (see Figure 40) needs to communicate with a service, the Authenticator, and the SIM to exchange credentials and access services. The Supplicant consists of a Service Supplicant and a SIM Supplicant.

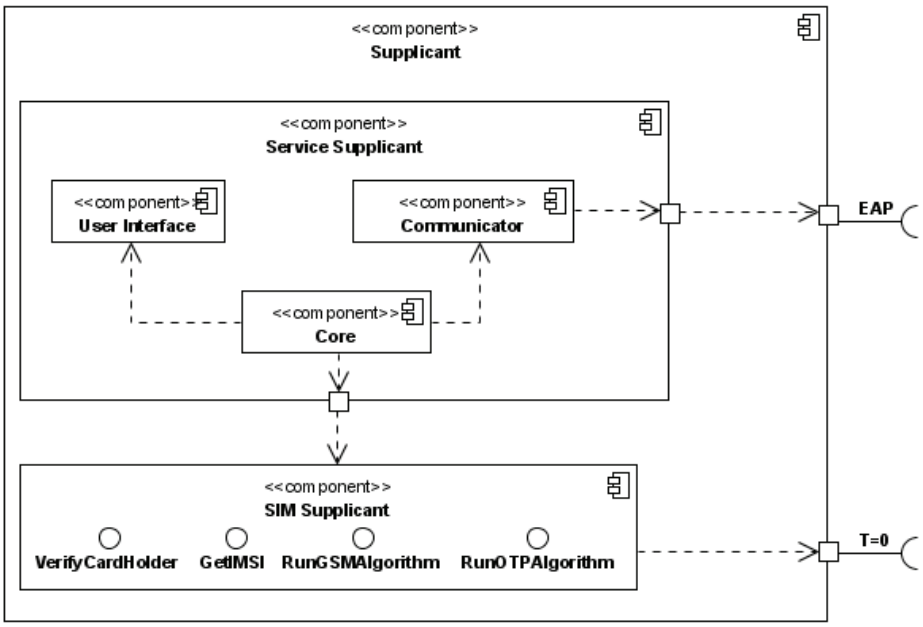


Figure 40: Component Diagram – Supplicant

The Service Supplicant
The Service Supplicant consists of the components User Interface, Communicator, and Core. The Service Supplicant will be initiated by the service requiring a SIM authentication, and will respond when authenticated.

The User Interface provides the User with messages and receives user inputs, i.e. the PIN.

The Communicator sets up a connection with the Authenticator, which should have the same Communicator installed.

The Core, besides being the "brain", will interpret messages and handle the different algorithms and protocols needed to authenticate the User (e.g. EAP-protocols).

The SIM Supplicant
The SIM Supplicant provides an interface to the Service Supplicant for easy retrieval of SIM information. The SIM Supplicant may be located with the Service Supplicant on a PC, i.e. when using a smart card reader, or separately, i.e. when SIM is located on a mobile phone.

The communication between the SIM and Service Supplicant may be direct method-calls or over for instance a communication protocol like Bluetooth.

The SIM Supplicant communicates with the SIM using APDUs according to the SIM's interface. This is done either directly (when located with the SIM) or over well-known APIs as OCF and PC/SC.

Component – SIM

Status: The SIM exists and is available through GSM 11.11 [18].

The SIM provides a series of interfaces; the ones below are needed during the authentication process. When the different interfaces are called, they will always return a message with the appropriate success-/failure-values, and any data.

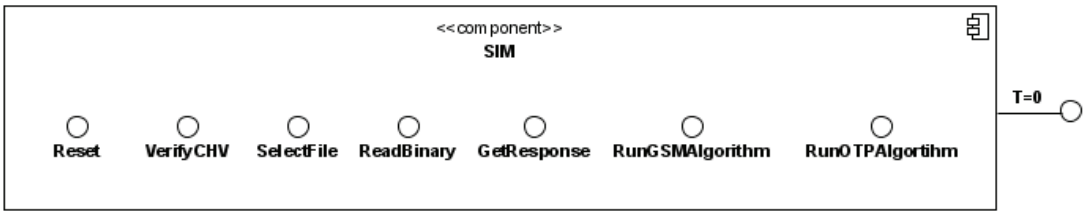


Figure 41: Component Diagram – SIM

Component – Authenticator

Status: None of the components exist

The Authenticator will need to have different plug-ins/implementations to accommodate the different authentication methods. The Authenticator needs to have an Authentication Server client (here shown as a RADIUS client) to be able to communicate with the Authentication Server. The Authenticator should store some information to be able to confirm the successful authentications to Service Providers.

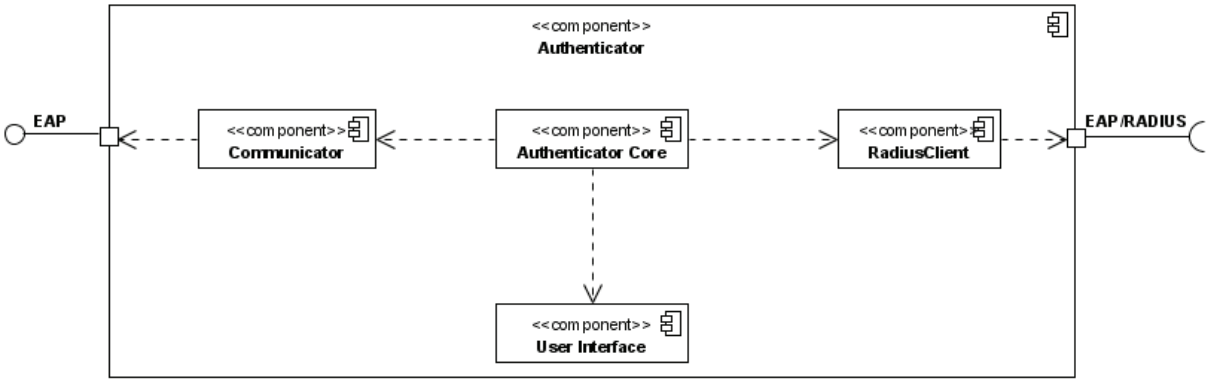


Figure 42: Component Diagram – Authenticator

The Authenticator Core, besides storing some state information, will interpret messages and handle the different algorithms and protocols needed to authenticate the User (e.g. EAP-protocols). It needs to be able to handle several sessions at the same time.

The Communicator has an open port, ready to receive connections from Supplicants, which should have the same Communicator installed.

The User Interface is used to read debug information, check active session, and should be able to kill sessions.

The Radius Client sends and receives messages to/from the Authentication Server. By designing this as a self-contained component with a standardized API it is easy to replace it to suit any Authentication Server.

Component – Authentication Server
Status: Exists, but plug-ins are required (some servers lack plug-ins)

In the back-end, the Generic SIM Authentication System has an Authentication Server that authenticates the Supplicants. A RADIUS server will be used as the Authentication Server because it is currently the de-facto standard for remote authentication. The reader is referred to section 2.6 for more information on RADIUS. To perform the GSM authentication the RADIUS server will use the EAP-SIM Plug-in and GSM Gateway Client. For other types of authentication additional plug-ins (i.e. EAP-OTP Plug-in) are used.

If the Authentication Server does not support the correct EAP-protocols, making the plug-ins are necessary.

The interfaces shown in the "Authentication Server"-component shows the packet types being used by RADIUS in all kinds of transactions. The EAP-messages are wrapped within the RADIUS packets.

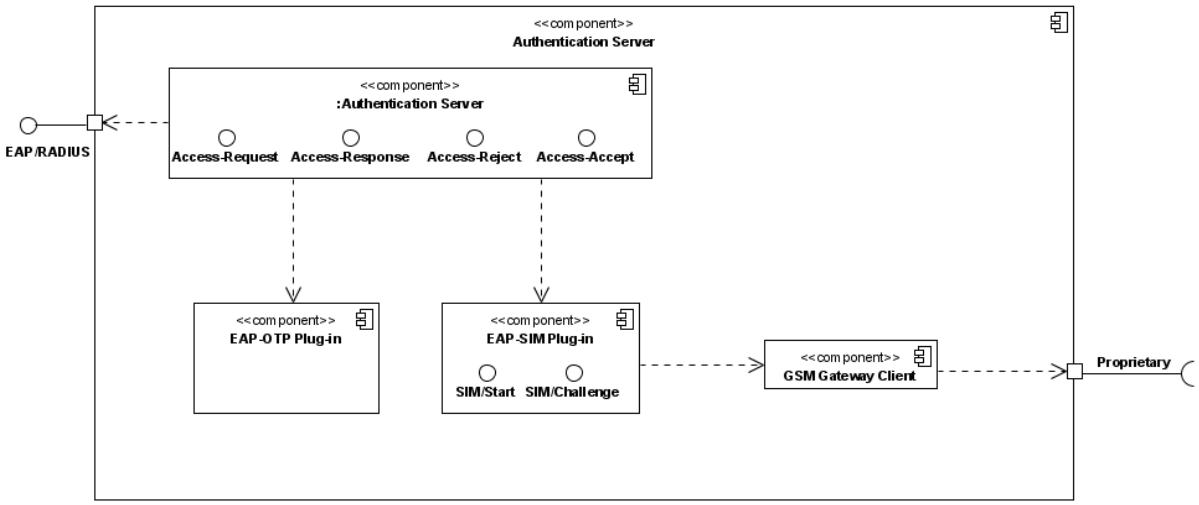


Figure 43: Component Diagram – Authentication Server

4.2 Class Diagrams

"*A class describes a set of objects that share the same specifications of features, constraints, and semantics. … The purpose of a class is to specify a classification of objects and to specify the features that characterize the structure and behavior of those objects.*" (UML 2.0 [62])

The package diagram (see Figure 44) is a reflection of the components shown in the component diagram in the section above. The package diagram is the basis for the class diagrams. The different class diagrams form the basis for the implementation. The implementation phase will evolve the structure of the classes, especially by adding many new

methods. The diagrams shown here is the original design, and for an exact reflection of the prototype defined in the next chapter use the Javadoc on the CD referred to in Appendix G.

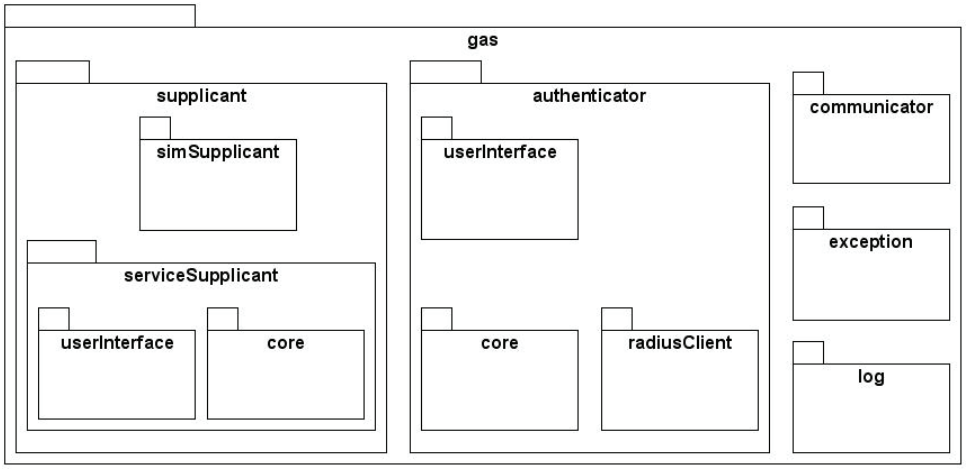


Figure 44: Package Diagram – Generic SIM Authentication System (GAS)

This section will elaborate four of the most important packages shown in Figure 44, namely the core of the serviceSupplicant and the authenticator, the simSupplicant, and the communicator.

Figure 45 shows the classes of the SIM Supplicant. The SimCore uses the SimServer to receive messages from the Service Supplicant and to send the responses. SimServerMobile and SimServerPC are the communication specific implementation towards the Service Supplicant. The active SimServer implementation depends on whether the SIM Supplicant is located on a mobile phone or with the Service Supplicant. The SIMcommands-class contains the APDUs needed to communicate with the SIM. Specific implementations according to communication protocols may be needed. The APDUs should be reused.

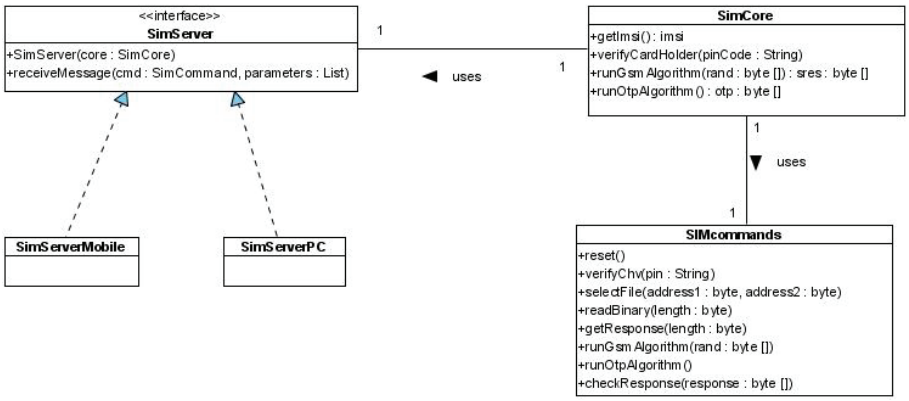


Figure 45: Class Diagram – gas.supplicant.simSupplicant

The serviceSupplicant consists of two packages, the core and the userInterface. The userInterface is just an interface (either a graphic user interface or using the command interface) towards the user. The core (shown in Figure 46) on the other hand contains all of the authentication method protocols. The ServiceCore contains the main-method and initiates an authentication method (here shown by EapSim and EapOtp) that inherits methods from the abstract class AuthenticationMethod. The ServiceCore uses a SimClient to communicate with the SIM Supplicant. The communication towards the Authenticator is described using the package communicator (see Figure 48).

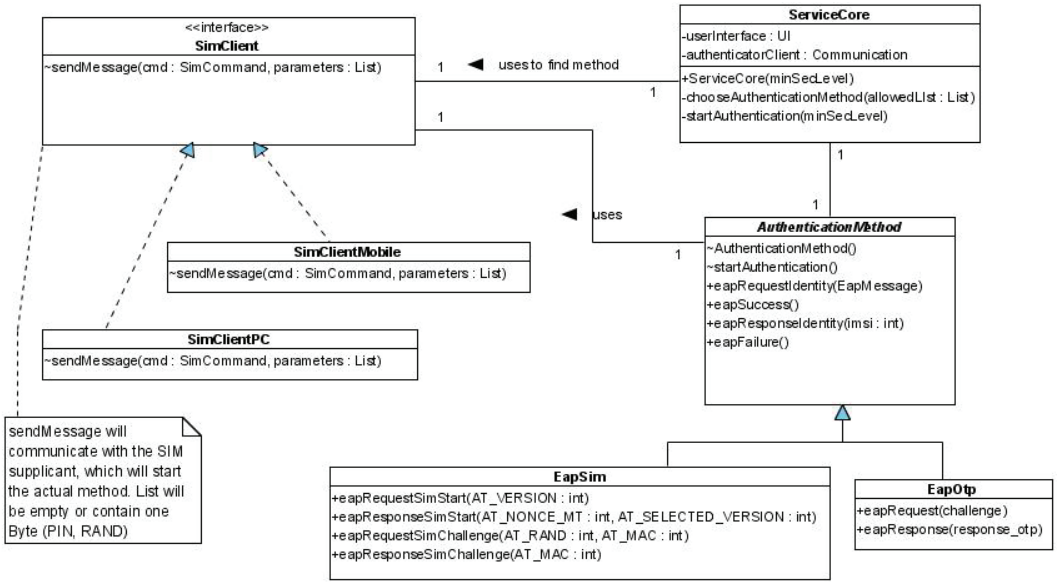


Figure 46: Class Diagram – gas.supplicant.serviceSupplicant.core

Figure 47 shows the core-package of the authenticator. In addition to the core, the authenticator-package consists of a radiusClient package (responsible for un-/wrapping EAP-messages into a RADIUS-packet and sending/receiving from the RADIUS server) and an interface towards the user (userInterface-package). Nevertheless, the most interesting part of the authenticator is defined in the core-package.

The AuthenticatorCore uses the AuthenticationSelector to investigate supported mechanisms by the Authentication Server (RADIUS server). The AuthenticationSelector stores the different authentication methods in a list of EapMethod-objects. The AuthenticationSelector is also able to return a list of authentication methods with a higher security level than minSecLevel. The AuthenticatorListener listens for incoming connections and notifies the AuthenticatorCore for every new connection. These new connections are put in a new object represented by the AuthenticationSession. Each AuthenticationSession is an own thread and performs the authentication process. A UserSession is a representation of an authenticated Supplicant. These are stored in a list and are used to verify alleged authenticated Supplicants.

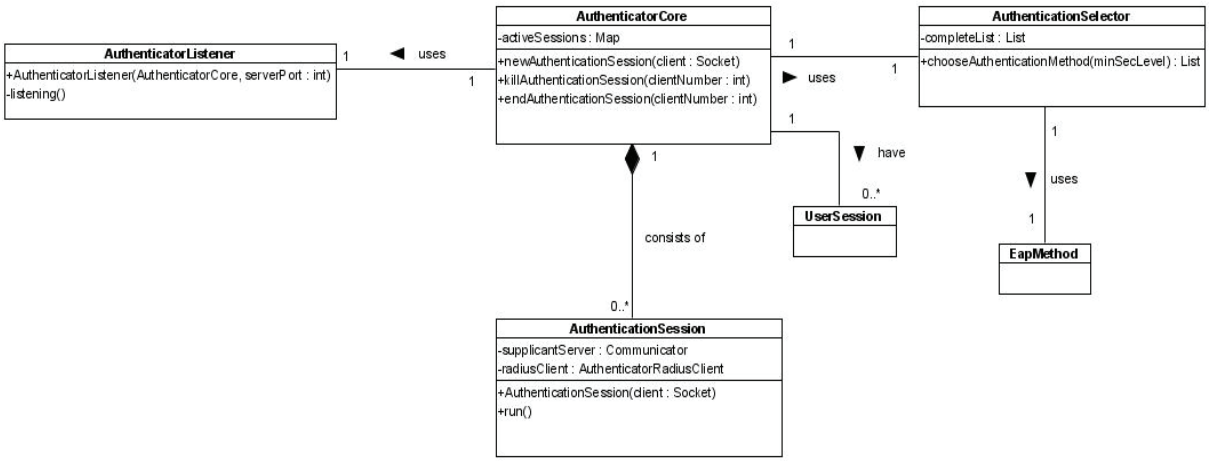


Figure 47: Class Diagram – gas.authenticator.core

Both the serviceSupplicant and the authenticator share some classes. These are put in the communicator-package (see Figure 48). The Communication-class is an abstract class used when setting up either a secure or a non-secure connection between the Supplicant or a Service Provider and the Authenticator.

The EapValue, EapMessage, and Attribute are three classes representing EAP-messages and the understanding of them.

The ByteArray-class is used for easier handling of adding and extracting part of a byte array.

The Crypto-class is used to handle all the calculation of cryptographic values used by the EAP authentication protocols and the RADIUS protocol. The implementation of the Crypto-class is shown in Appendix C.

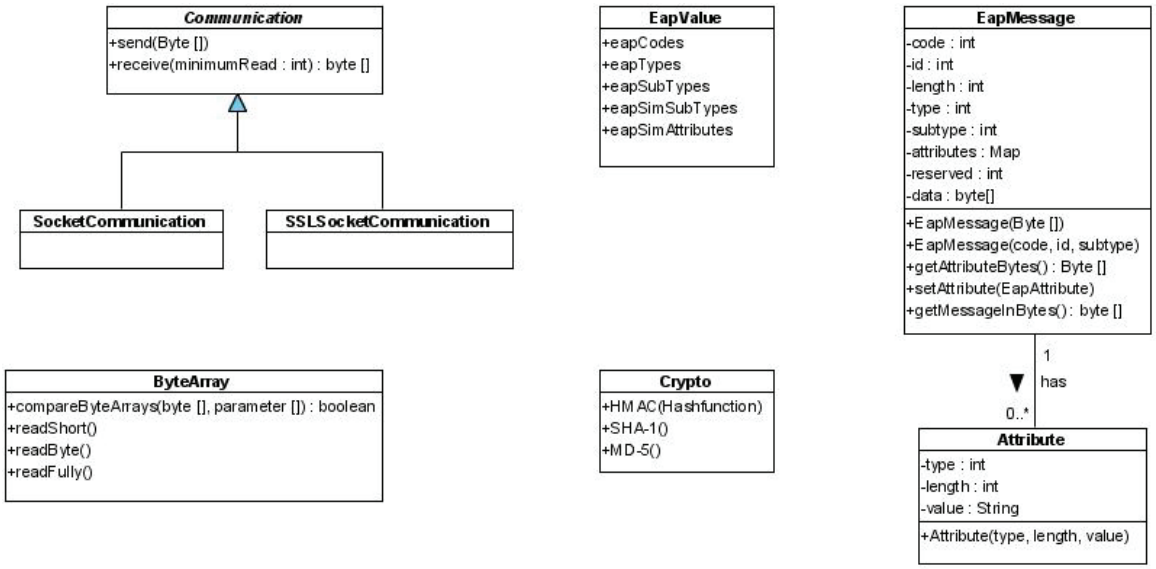


Figure 48: Class Diagram – gas.communicator

The rest of the class diagrams are found in Appendix A.2.

4.3 Summary

The design phase has shown that the Generic SIM Authentication System can be based on existing technologies and protocols, but needs to implement a Supplicant and an Authenticator. It might also be necessary to implement plug-ins to the Authentication Server to be able to implement more functionality on the SIM, and some authentication servers do not have the EAP-SIM already implemented.

The component diagrams have given a high-level understanding of the system and are the basis for package diagrams for the classes. The different communicating components need to have well-defined interfaces towards each other as shown in the component diagrams.

With this design phase in mind it is possible to realize the components in a deployment diagram and develop a Prototype of the Generic SIM Authentication System.

5 Realization of a Prototype

This chapter will describe the implementation of a Prototype showing the main concept of the Generic SIM Authentication System (GAS). Before the implementation is described, a possible deployment of the components will be shown. After the "how to" implement section in section 5.2, configuration of the system will be illustrated in section 5.3, example of services that use GAS are described in section 5.4, and finally a review of the implementation will be performed in section 5.5 and 5.6.

5.1 Deployment Diagrams

"*The Deployments package specifies a set of constructs that can be used to define the execution architecture of systems that represent the assignment of software artefacts to nodes. Nodes are connected through communication paths to create network systems of arbitrary complexity.*" (UML 2.0 [62])

Before implementing the Prototype a deployment diagram was designed to give a better overview of the system and how it interworks. The deployment diagram (see Figure 49) shows where the different components belong and what kind of Operating Systems (OS) and hardware is needed.

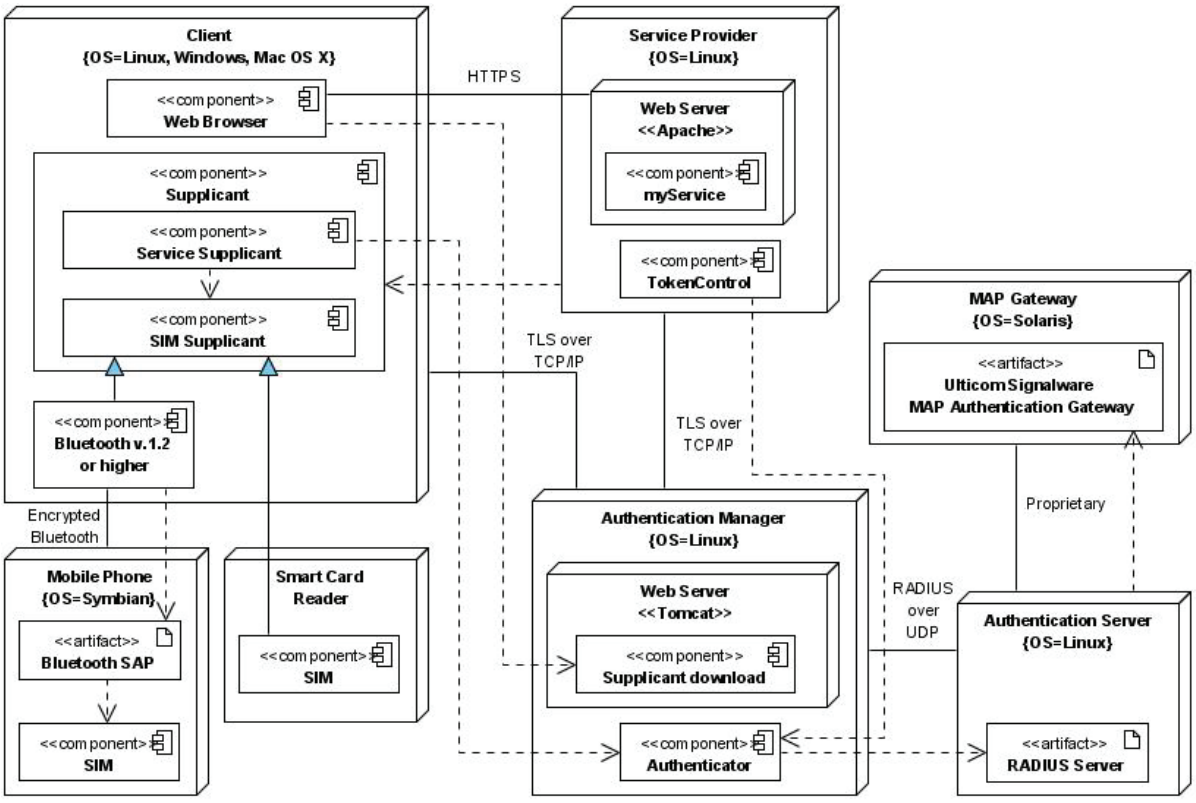


Figure 49: Deployment Diagram – High-level

As the figure shows, the different entities have been placed on different Operating Systems. This is only arbitrarily due to the available machines that have been used during the implementation and testing. The same pass for the web servers used in the deployment of the Prototype.

5.2 Implementation

5.2.1 SIM Communication

There are two main classes of SIM readers, the integrated reader in the mobile phone and the smart card reader. They incorporate different security mechanisms, where the former is stricter and less implemented. The implementation of the communication in Java will be described next.

Both communication choices communicate with the SIM over GSM 11.11, which means to transfer APDUs from the computer to the SIM.

Mobile Phone SIM Reader

To communicate with the SIM card through the mobile phone it is required that the phone provides a SIM access interface. The equipment providers are often holding back the interface to the SIM card because of security reasons, and therefore many of the access mechanisms discussed in section 2.7 are not implemented by current mobile phones. The Bluetooth SIM Access Profile seems like the most promising technology for the future, but unfortunately the profile is only available on recent phones (in particular Nokia and Siemens phones).

SIM Access Profile (SAP)

Before the SAP connection can be started the user needs to turn on Bluetooth and allow remote SIM connections on the mobile phone. A list of Nokia and Siemens phones that support the SAP profile can be found in Appendix D. The SAP connection also requires a 16-digit passkey and the computer needs a newer Bluetooth device (see section 5.6.2 for tested devices).

To use the SIM Access Profile (SAP) the client needs to perform a Bluetooth service discovery to find the channel number that the SIM Access Server is using. The SAP service name is SIM Access and the service class id is 0x112D.

The message formats and the protocol are described in section 2.7.1. It is important to notice that the client must start the communication with a connection request message and finish the communication with a disconnect message. When a SIM access connection is established the client can transfer a Command-APDU to the SIM by using the message "TRANSFER_APDU_REQ". The server will then respond with a "TRANSFER_APDU_RESP" to send the Response-APDU from the SIM.

The implementation of the Bluetooth client uses the Avetana library in Java, which supports the built in Bluetooth stacks on most operating systems. A description of the different Java Bluetooth implementations is provided in more detail in the next section.

Smart Card Reader

To communicate with the smart card reader the client needs to use the readers API. Today there exist different types of standardized smart card interface APIs, the two most known PC/SC and OCF were discussed in section 2.3.4. The PC/SC API has become the de-facto standard implemented by most smart card readers on the market today. Because of this the PC/SC interface was a natural choice for our implementation.

The PC/SC interface works on many platforms, either through the Win32 API or the PCSC Lite tool. The communication with the smart card reader from Java needs a library that implements an interface between Java and the lower level PC/SC on the operating system. We used an open source library called JPCSC [63], which implements a Java Native Interface (JNI) wrapper to allow the access to PCSC functions from Java. JPCSC offers an API to establish a connection with the SIM card and communicate through APDUs.

JPCSC supports both Windows and Linux, and with some small changes it also works on Mac OS X. JPCSC needs to have a native library installed in the Java library path in order to work. To work around the issue that the user needs to install this native library, the source code was changed to rather look for the native library in the JPCSC jar file. This is especially useful when the client is implemented as a Java Applet.

5.2.2 Bluetooth and Java

The Bluetooth stack is the software or firmware component that has direct access to the Bluetooth device. In newer operating systems the integration of a Bluetooth stack has become standard. With Windows XP service pack (sp) 2 the Microsoft Bluetooth stack was built in. Before sp2, the WIDCOMM stack was the most widespread implementation of a Bluetooth stack on Windows. The official Linux Bluetooth stack is called BlueZ [64], and was included with Linux 2.4 kernel series and higher. On Mac OS X the Apple's Bluetooth stack was integrated starting with version 10.2.

Java APIs for Bluetooth are specified in JSR-82, and were discussed in section 2.7.2. To make the implementation as independent as possible from the stack the standardized API was used and a Java Bluetooth implementation that supported the JSR-82 was chosen. Currently the JSR-82 can only be implemented on the Java ME platform, which supports the Generic Connection Framework (GCF). That means any J2SE implementation needs to include an implementation of the GCF to be able to communicate over Bluetooth.

When accessing a Bluetooth stack through Java, there are two main choices:

- Java interface to a native stack
- Java implementation of the whole stack

There are many Java Bluetooth SDKs available, some of them are:

- Avetana (Win-32, Mac OS X and Linux)
- BlueCove (WinXP SP2)
- JavaBluetooth (Devices that are connected with a UART COM port)
- MinShara (Based on JavaBluetooth, add RFCOMM and OBEX support)
- Atinav, aveLink (Win-32 and Linux)

The Avetana Bluetooth [65] JSR-82 implementation was chosen, because of the support for the most widely spread Bluetooth-Protocol Stacks and operating systems. The software is commercial and needs a license in order to work on Windows and Mac OS X, but the Linux implementation is free.

5.2.3 Cryptography

This section briefly describes the implementation and references for the different cryptographic algorithms implemented in the Prototype of the Generic SIM Authentication System.

G-function
The G-function is mainly a part of the SHA-1 implementation, but since it is used by another algorithm it is separated from the SHA-1 implementation. See SHA-1 (below) and PRNG (in section 5.2.4) for more information.

The implementation of the G-function follows the one specified in FIPS 186-2 Change Notice #1 [66]. When used by SHA-1, the input message is padded with a 1, then zeroes and the last 64 bits indicates the length of the input message. The padded message is then a multiple of 512 bits (n*512). The G-function has an input of 512 bits and will be used n times by the SHA-1 algorithm. When the G-function is used in conjunction with EAP-SIM it is IMPORTANT to pad the input with ONLY zeroes. The 512 bits (64 Bytes) input to the G-function returns 160 bits (20 Bytes).

SHA-1
SHA-1 (Secure Hash Algorithm) 1 or SHA-160 is a hash function defined in the Secure Hash Signature Standard (SHS) [67] (FIPS PUB 180-2 Change Notice #1). The SHA-1 pads the message as specified above. For each 512 bits, SHA-1 uses the G-function. SHA-1 returns a hash message digest of 160 bits.

MD-5
Message Digest 5 (MD-5) is a hash function specified in RFC1321. The first part of the method pads the incoming message into a multiple of 512 bits. Every block of 512 bits undergoes shift operations as specified before the message digest of 160 bits is returned.

HMAC
The Keyed-Hash Message Authentication Code (HMAC) is specified in National Institute of Standards and Technology (NIST) Federal Information Processing Standards (FIPS) Publication 198-a [68]. It uses a secret key to generate a MAC using a specific hashing algorithm in the process. The HMAC normally returns a 160 bits MAC, but our implementation is an HMAC-128, i.e. it truncates the 160 bits return value to 128 bits (16 bytes).

Our implementation can use either SHA-1 or MD-5 as the specific hashing algorithms.

5.2.4 EAP over TCP

Transmission Control Protocol (TCP) is one of the main protocols in TCP/IP networks. Whereas the Internet Protocol (IP) provides the basic delivery mechanism for packets of data sent between systems, TCP enables two hosts to establish a connection and exchange streams of data. TCP guarantees delivery of data and also guarantees that packets will be delivered in the same order in which they were sent. Since packets can be lost in intermediate systems, TCP adds support to detect errors or lost data and trigger retransmission until the data is correctly and completely received.

The implementation in Java of the TCP-connection, the Transport Layer Security, and EAP will be elaborated in this section. It will have a special focus on challenges met during implementation and clarify indistinctness in the EAP-SIM protocol [38].

TCP – Transmission Control Protocol
Java uses two classes, *java.net.ServerSocket* and *java.net.Socket*, to establish a connection between a client and a server. Socket is the name given to the package of subroutines that provide access to TCP/IP on most systems.

In the Generic SIM Authentication System, the Authenticator sets up a listener that establishes a Server Socket on a specific port. When a Supplicant needs to be authenticated, it tries to connect to the Authenticator on the specific port. The Supplicant creates a socket using the Authenticator's IP-address and port. The Authenticator will accept the connection and establish an input- and output stream, and start the communication.

TLS – Transport Layer Security
When using secure sockets in Java, you may use protocols such as the Secure Socket Layer (SSL) or IETF's TLS. The Java classes are *java.net.SSLSocket* and *java.net.SSLServerSocket*, which extend Socket and ServerSocket, respectively.

These classes add a layer of protection, such as:

- Integrity Protection.
- Authentication (in most modes). Servers are usually authenticated, and clients may be authenticated as requested by servers.
- Confidentiality (in most modes). SSL encrypts data being sent between client and server.

To have both client and server certificates and full exchange of public keys is cumbersome. To make it feasible to download applets and alike without this cumbersome process, there is no authentication of the Supplicant on the Transport layer. The Authenticator has a certificate and combined with random numbers from both the Supplicant and the Authenticator, the data sent will be encrypted and the integrity checked.

The making of certificates (keystores) is elaborated in section 5.3.

EAP – Extensible Authentication Protocol
There is no authentication on the Transport layer, but the Generic SIM Authentication System uses EAP to authenticate Supplicants. EAP is on the Application layer and is managed by the Supplicant, Authenticator, and the Authentication Server. EAP is implemented according to RFC 3748 [37], described in chapter 2.5. The EAP-message is a shell containing an authentication procedure. The focus in our implementation is EAP-SIM [38], described in section 2.5.1. The implementation of EAP is used by both the Supplicant and the Authenticator and will be explained below, together with EAP-SIM relevant cryptographic challenges.

When receiving a message the read-method will read as a minimum, 4 bytes (the EAP header length) from the socket. Within the header there is a packet length field telling the total length of the packet. This value is used to determine how many bytes needed to be read before interpreting the packet. The packet will then be interpreted into the correct type of packet, and the EAP-message will be created together with all its attributes. The message will then be dealt with according to protocol.

The EAP-SIM protocol uses a message authentication code (MAC) and some attributes are hashed in different ways (see section 5.2.3). IMPORTANT:

- Hashing EAP-SIM message:
  - The attributes must be sorted (small to high) by its type value when creating MACs.
  - The MAC attribute must be a part of the EAP message, but the data value must contain only zeroes.
- Creating session keys: Use the Pseudo-Random Number Function defined below

Pseudo-Random Number Function (PRNG)
The EAP-SIM specific PRNG uses a similar algorithm to SHA-1. Both use the so-called G-function (see section 5.2.3), but the difference is that when padding the incoming message (in our case the MK (Master Key)) to 512 bits, pad only with zeroes. No single bit set to 1 and no length at the end of the padding as in the case with SHA-1.

The PRNG used in the key derivation is based on the SHA-1 and the EAP-SIM protocol [38]. The function returns a 1280 bits (160 Bytes) array that can be partitioned into suitable-sized keys.

5.2.5 RADIUS Client

The RADIUS client is part of the Authenticator. The main functionality of the Authenticator is to locate a suitable Authentication Server, act as a translator from EAP to RADIUS and back, store information about authorized Supplicants, and keep track of identities when for instance temporary identities are used. When the Authenticator receives EAP messages from the Supplicant, it will forward these inside a RADIUS packet to the RADIUS server. All RADIUS packets received from the RADIUS server is checked for validity and forwarded to the right Supplicant.

The implementation of the RADIUS client uses code from the open source project JRadiusClient [69], which is an implementation of a RADIUS client intended to be used as a library. The JRadiusClient library does not have EAP support, to support this extension an entire new client was developed from scratch using the source code from the JRadiusClient. This was necessary because of the way the library was implemented; it was difficult to add this extension without changing the core of the library itself.

When testing the Authenticator with more than 255 transactions, a serious problem occurred. The problem was that the RADIUS packet identifier was incremented with every packet sent. Since this was a static byte field in Java the byte went out of bounds when the packet identifier reached 255. The solution was to add modulo into the packet identifier increment method.

The RADIUS client is compliant with the RFC 2865 and RFC 3579, however it only supports the attributes discussed in the background section. The UDP retransmission timer is implemented by using maximum 3 attempts each with a timeout of 6 seconds.

When a RADIUS message includes the EAP attribute, the client needs to check or add the Message-Authenticator (see section 2.6). The implementation of the HMAC-MD5 is explained in section 5.2.3.

5.3 Configuration

5.3.1 Tools

This section describes the different tools used in the development of the Prototype. A short description of each tool together with the basic configuration and commands will be given.

Apache Ant

Apache Ant [70] is an open source Java-based tool for automating the software build process. This tool was useful to automate the building of the different parts of the Prototype, together with the creation and signing of deployment files (i.e. jar files). After installing Ant it is necessary to perform some additional setup on the system. This includes adding the *bin* directory to the path and setting the ANT_HOME environment variable to the directory where Ant was installed. The online manual [71] describes the installation steps in more detail.

Ant's buildfiles are written in XML, and by default the buildfile is called *build.xml.* Each buildfile contains one project and one or more targets. A target is a set of tasks that Ant will execute when this target is chosen. Example of tasks is to create a directory, invoke the Java compiler or invoke some other tool on the system. When starting Ant, you can select which targets you want to be executed or the default target is executed.

The configuration file for the Prototype includes many different targets, where the default target will take care of building the whole system. To start building the Prototype, type the command in the project directory, the same directory as the *build.xml* file:
#ant
To print information about the different targets, use the command:
#ant -p
For instance to only build and run the standalone client, type the command:
#ant Client

Java Keytool – Key and Certificate Management Tool

Java Keytool is a key and certificate management tool implemented by Sun Microsystems, and distributed together with the standard J2SE JDK. This tool was used in the generation of the key and certificate used with the Java secure socket layer and the signing of jar files. This section is based on the J2SE documentation for keytool [72] and [73].

The keytool application manages a keystore (database) of private keys and their associated X.509 certificate chains authenticating the corresponding public keys. The keytool can also manage certificates from trusted entities. Keytool stores the keys and certificates in a so-called keystore, which is a file. The keys are protected with a password.

There are two different types of entries in a keystore:

- Key entries: private key and the corresponding public key
- Trusted certificate entries: Single public key certificate

All keystore entries are access with a unique alias, which is specified when you generate the key pair or when you import a trusted certificate.

The following command was used to generate the key pair:

#keytool -genkey -alias <privatename> -keyalg rsa -keystore <privatestorename>

First the keytool will require a password for the keystore and, optionally, a password for the key pair that is created. Next the application will start to ask a series of questions, e.g. your name, organization, and the like. The information will be used to create a self-signed certificate that associates the information with a public key. To use other key algorithms and key sizes that the default, the command line options -keyalg and -keysize can be used respectively.

To list the keystore and the stored entries, the following command can be used:

#keytool –list -keystore <storename>

The generated key pair can now be used together with Java Secure Socket Communication, Java Jarsigner and Apache Tomcat.

If the client wants to verify the certificate, it can check it against the local keystore (trust store) with trusted certificates. To add the newly generated certificate to the trust store it first needs to be extracted from the keystore and stored into a temporary file.

#keytool -export -alias <privatename> -file <certname.cert> -keystore <privatestorename>

Then, the certificate can be stored into the trust store with the command:

#keytool -import –alias <publicname> -file <certname.cert> -keystore <publicstorename>

Java Jarsigner - JAR Signing and Verification Tool
Java Jarsigner [74] is a tool implemented by Sun Microsystems, and is used to generate signatures for Java ARchive (JAR) files, and to verify the signatures of signed JAR files. Jarsigner uses key and certificate information from a keystore to generate digital signatures for JAR files. When the JAR file is signed, a copy of the certificate from the keystore is also added to the file. When the jarsigner verifies the digital signature of the signed JAR file, it uses the certificate that is stored inside the file.

The command to sign a jar file is:

#jarsigner -keystore <privatestorename> -storepass <storepassword> -keypass <storepassword> <JARFILE.jar> <aliasname>

Ethereal – Network Protocol Analyzer
Ethereal [75] is an open source network protocol analyzer, or “packet sniffer” application. It is a useful application when performing troubleshooting and analysis of protocols and software. Data can be captured from a live network connection and stored for later analysis. Ethereal was extensively used when developing the EAP-SIM over RADIUS in the Authenticator.

5.3.2 OS Specific Client Configurations

The Prototype client comes in two versions, the Java Applet and the Standalone client. The standalone client requires a Java Runtime Environment (JRE) installed on the client computer. The Java Applet also requires a browser with the Java Plug-in enabled, which should be automatically enabled when the JRE is installed. The client computer may need some first

time configurations for the communication with the SIM card to work. This section will describe the configuration steps needed to get the smart card reader and Bluetooth SIM Access to work for Windows, Mac OS X and Linux.

Smart Card Reader

Windows

Windows XP comes with smart card support installed, so the user only needs to make sure windows recognize the smart card reader. This may include installing the driver that was included with the smart card reader.

Mac OS X

Mac OS X also comes with smart card support installed, however there may be some problems with the software that is installed and the Java PC/SC library provided by the Prototype. To get around this problem the software must be built from the source code, the Prototype provides a Java PC/SC library tested with the PCSC-Lite version 1.2.0.

To build the smart card software from the source code, download PCSC-Lite from Musclecard [32]. Then build the software, follow the README notes. If the problem with the Java PC/SC library still exists, it is best to also build this from the source. The source code can also be downloaded from the Musclecard page, however the building procedure is slightly more complicated because the included build file only supports Linux. The changes that are needed in order to build successfully on Mac OS X are described in Appendix E.

Linux

Many Linux distributions have the PCSC-Lite (pcscd) software installed as default. If the pcscd is missing, it is easiest to install it directly from the Linux distribution, remember to also install the development/source code files. Another option is to download the software from Musclecard [32] and build it from the source code

On both Mac OS X and Linux the "pcscd" must be running for the smart card service to work. If the smart card reader is not recognized by the pcscd, the driver must be installed into the PCSC-Lite driver directory (default *<pcsc-directory>/drivers/*).

Bluetooth SIM Access

Bluetooth SIM Access requires that the mobile phone and the computer are being paired with a passkey of at least 16 digits. The bonding has to be performed before the SIM access connection is started. If the bonding is necessary the OS specific Bluetooth implementation will ask the user to enter a passkey that must be repeated at the mobile phone. This automatic bonding do not always work, so the best option is to start the bonding from the mobile phone before the SIM access connection is started.

See section 5.2.2 for a short description on the integration of Bluetooth with the different operation systems.

5.3.3 Authenticator

The Authenticator can be started in two modes, either in the normal mode running in the foreground or as a daemon running in the background. In both modes the logging functionality is configured through the *log4j.xml* configuration file. By default the Authenticator writes all messages to the log file *gas.log*.

Starting Authenticator with Ant
To start the Authenticator in normal mode, use the following command in the directory where the Authenticator is installed:
#ant Authenticator
This will start the Authenticator in the foreground, using the parameters provided in the *build.xml* configuration file. To change the default parameters edit the configuration file, e.g. to change the RADIUS server hostname change the following parameters value to the new hostname:
<property name=”authenticatorHostname” value=”localhost”>

Starting Authenticator in daemon mode
A daemon is a computer process that runs in the background, rather than under the direct control of a user. To start the Authenticator in daemon mode use the shell script *startDaemon,* and to stop a running Authenticator daemon use the shell script *stopDaemon*. The daemon mode only works on UNIX systems.

5.3.4 RADIUS Server

Our Prototype is tested with two different RADIUS servers, NavisRadius and FreeRadius. They both support EAP-SIM, and the configuration of the servers to get this to work will be described next.

NavisRadius
NavisRadius [76], developed by Lucent Technologies is a commercial RADIUS server. That means in order to run the server you need a valid license, we were provided a testing license through Telenor R&D. In the upcoming version 5.0 of the NavisRadius product, it will be renamed from NavisRadius to VitalAAA with the addition of support for Diameter.

NavisRadius is developed in Java and should work on all systems that have installed Java 1.4.2 or later. NavisRadius was chosen because of the extensive plug-in support and in our case an EAP-SIM plug-in. The plug-in is called “AuthEapSim” and implements the EAP-SIM IETF drafts 5 and 15 through a number of method properties. The gathering of GSM triplets supports three different methods:

1. Retrieve triplets from a HLR using a GSM Gateway (ReadMapGateway plug-in)
2. Read triplets from a triplet file (ReadGsmTripletFile plug-in)
3. Generate triplets at the RADIUS server (GenerateGsmTriplet plug-in)

Method 2 and 3 is considered to be most useful for testing purposes, where method 1 is the most realistic in a deployed system. The ReadMapGateway plug-in needs contact with a GSM network to communicate with the HLR. Since NavisRadius operates on an IP network and an HLR operates on a SS7 network, requests for the HLR need to be made through a special gateway. Currently NavisRadius only supports using the Ulticom MAP Gateway.

Configuration and startup of NavisRadius
We have tested the NavisRadius version 4.5.6 and 4.5.2 with our application. The installation procedure and the basic configuration of the server is well documented in the Quick Start Guide and in the more complete Server Management Tool Manual, which follows the downloadable software build. One of the important steps is to remember to add the RADIUS client to the “clients” configuration file. This section will describe the extra steps necessary to configure the EAP-SIM support.

The server's configuration files are located in the *run* directory, and in order to start the AuthEapSim plug-in we need to change some of these. The configuration files we are going to start from is located in the *run/samples/eap-sim* directory, copy them to the *run* directory. The "auth_methods" configuration file contains the choice of the default GSM triplet method and details for each of these. To start using GSM triplets method 2 and 3 it is not necessary to do any configurations in this file, for the ReadMapGateway plug-in it is necessary to configure the gateway address. We tested the EAP-SIM plug-in with the ReadGsmTripletFile plug-in. In order to authenticate a user the server needs a binary file containing blocks of 28 byte triplets, a triplet is a 16 byte RAND, a 4 byte SRES, and an 8 byte Kc in that order. The server software also comes with a small MS Windows application called "triplettool" (located in the *run/samples/eap-sim* directory), which is necessary in the generation of the triplet file. To use the tool, open a Command Prompt window, place a SIM card in a PC/SC compatible smart card reader, and type: triplettool -pin <PIN>. This will generate a tripletfile with the name of SIM's IMSI, copy this file into the *run* directory.

To start NavisRadius you can use the UNIX executable in the *bin* directory, using the command: *#./nr start all.*
On windows you can start the server from the start menu, using the program "Start Server".

NavisRadius logs important events into the navisradius.log file, located in the *run* directory. If you want to have more information to the log file, you can change the logging rules through the configuration file "log_rules".

When the RADIUS server receives EAP-SIM authentication request, it looks up the permanent identity of the request and the realm part decide what kind of GSM triplet method is to be used. The default configuration file uses the ReadGsmTripletFile plug-in when the read.triplet.file.com realm is found, the GenerateGsmTriplet plug-in when the read.sim.key.com realm is found and the ReadMapGateway plug-in for all other realms.

FreeRADIUS
FreeRADIUS [77] is a free premiere open source RADIUS server. The server is released under the GNU General Public License (GPL), which means that it is free to download and install. FreeRADIUS is developed in C and should work on all UNIX systems. FreeRADIUS was chosen because of its flexible module system and the support for EAP-SIM authentication, it also provides the capability for you to easily create your own modules. The EAP-SIM plug-in is called "rlm_eap_sim", currently it only supports gathering of GSM triplets through a text file. This is supported by the plug-in "rlm_sim_files" or from the "users" file, if one wants to fetch triplets from the HLR then a module needs to be developed.

Configuration and startup of FreeRADIUS
We have tested the FreeRADIUS Version 1.1.0 with our application. The installation procedure and the basic configuration of the server are well documented in the INSTALL file and in the Wiki at the webpage. Many of the initial configuration steps are similar as with NavisRadius. This section will describe the extra steps necessary to configure the EAP-SIM support.

The server's configuration files are located in the *<installed directory>/etc/raddb*, where the "radiusd.conf" is the main configuration file. The EAP authentication methods are configured through the "eap.conf" configuration file. To add support for EAP-SIM you must change the parameter *default_eap_type* to be *sim*, and also add *sim{}* somewhere in the file. The EAP-

SIM module requires that triplet attributes are set before it is called. The attributes can be set through the users file or with help from the module "sim_files". One example of a users file that sets the required attribute can be found in the location:
<source-directory>/src/tests/eapsim-03.
We have developed a small application called TripletGenerator. It connects to a PC/SC reader with a SIM card and generates a specified number of triplets, which are shown in the format understood by the sim_files module.

To use the "sim_files" module you need to install FreeRADIUS with support for experimental-modules. To configure the module you must edit the "radiusd.conf" file. Add *sim_files {}* to the modules{} section and add *sim_files* to the authorize{} section, both places you must add this before eap is added.

To start FreeRADIUS you use the executable in the location *<installed directory>/sbin/,* using the command: *#./radiusd*

FreeRADIUS logs important events into the radius.log file, located in the *<installed directory>/var/log/radius* directory.

5.4 Services

5.4.1 Service Provider using JSP

The Prototype includes a simple service provider (SP) application that gives an example of how the authentication service could be integrated into a web application. The SP was implemented using HTML and JavaServer Pages (JSP). JSP is a technology that allows Java code and certain pre-defined actions to be embedded into static HTML content.

The user can navigate to the SP using a normal web browser, and all communication with the SP uses HTTPS communication. See Figure 50 for a screenshot from the demo SP.

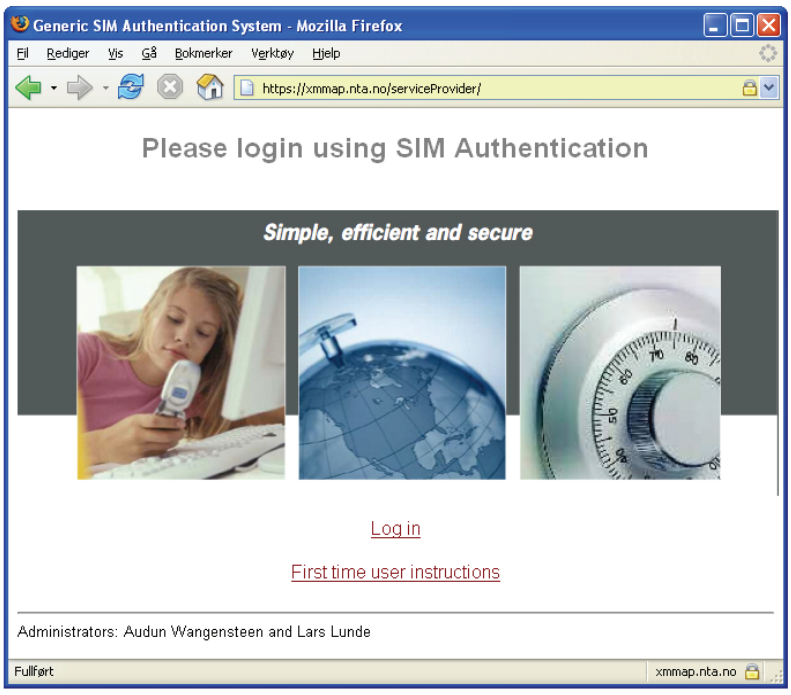


Figure 50: Prototype – Main Page

When the user navigates to the "Log In" page, the supplicant Java Applet will be loaded, and the user can start the authentication with the Authenticator. First the user needs to choose which type of communication device it has available (see Figure 51, left screenshot). If the user chose the Smart Card Reader type, the application will ask for the Pin code (see Figure 51, right screenshot)

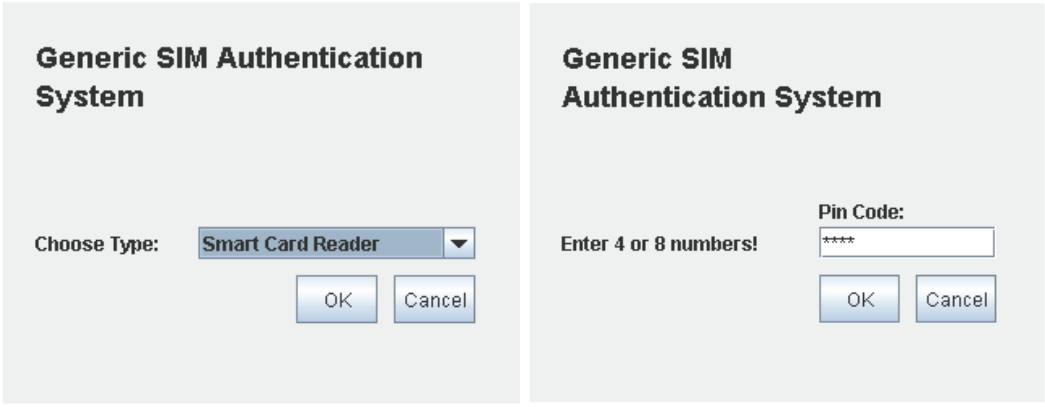


Figure 51: Generic SIM Authentication System Supplicant

In a successful authentication the user will be given a special token, which is presented to the SP. The SP will then check this token with the Authenticator communicating over a secure socket using EAP messages. When the SP has ensured that the user was authenticated, the "Welcome" page is displayed (see Figure 52).

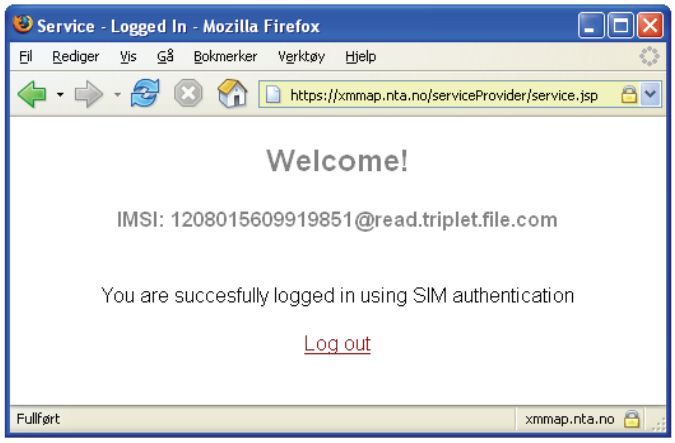


Figure 52: Prototype – Welcome Page

Apache Tomcat

Apache Tomcat, formerly known as the Apache Jakarta Project, implements the Servlet and the JSP specifications from Sun Microsystems that provides an environment for Java code to run together with a web server. Tomcat actually works as a standalone web server. The Prototype using jsp is deployed using Apache Tomcat [78] as a web container.

Before starting the Tomcat server it needs to be configured to support HTTPS connections. The configuration file is located in the *<tomcat directory>/conf* directory. The server.xml file contains the main configuration, and the HTTPS option is commented out by default. Add to the definition after: *<!-- Define a SSL Coyote HTTP/1.1 Connector on port 8443 -->*

keystoreFile="<address to the private keystore> "
keystorePass="<password to the private keystore>"

When HTTPS is configured, start the Tomcat server by using the shell script *startup.sh* in the directory *<tomcat directory>/bin.*

To deploy the service provider simply copy the serviceprovider.war file into the directory *<tomcat directory>/webapps* and the application will automatically be unpacked and ready to process users.

5.4.2 MyService, Service Provider using PHP

The "MyService" is another example of how the authentication service could be integrated into a web portal. MyService also includes an example of how the service provider can control the registration of new users and link up with their SIM identity. MyService was implemented using HTML and PHP: Hypertext Preprocessor (PHP). PHP is a popular scripting language that is especially suited for Web development.

All communication with MyService uses HTTPS communication.

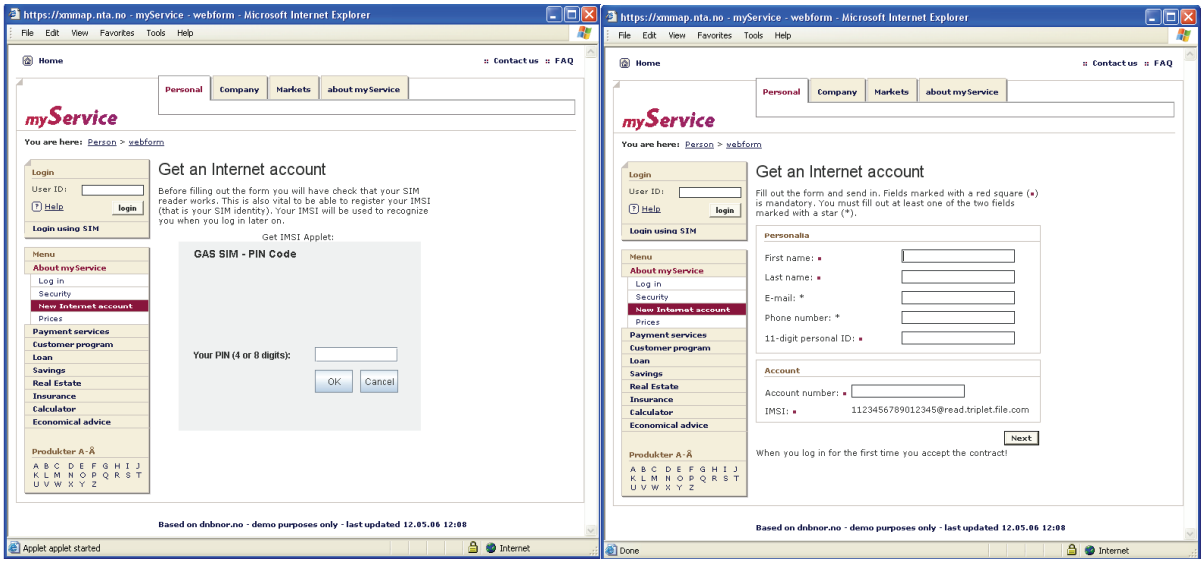


Figure 53: MyService – New Internet Account

Figure 53 shows the two windows included in the registration process of a new Internet account customer of myService. The left part of the figure contains the Java Applet. It is necessary to perform a SIM authentication to extract IMSI and to confirm the authentication for the SP. The next step (shown in the right part of the figure) is to register the new user with the information needed by the SP. The SP can after the registration process automatically open an account for the user or choose to confirm the information elsewhere before opening the account. This is entirely up to the SP and is important in those cases where the SP wants to have control.

The log in phase looks like the left part of Figure 53, but will result in the User's personal myService page. To be able to confirm an authentication, the SP will use an additional application that contacts the Authenticator. The code uses some classes of the GAS prototype, but has one additional class used for the confirmation. The code is on the CD as referred to in Appendix G. To be able to execute the Java Application, the PHP version supported by the Apache server must not be configured for safe mode. The function used in PHP to execute a file is: *#shell_exec (string cmd)*

MyService uses mySQL to store the personal information. To be able to use mySQL, PHP must be configured to use mySQL.

Apache HTTP Server with PHP and SSL
Apache HTTP Server [79] is the most popular open source web server, which includes support for PHP by installing an extra module.

To setup SSL with Apache2, use the following steps:

1. Enable the ssl module: *#a2enmod ssl*
2. Add the ssl port (443) to *<apache dir>/ports.conf*
3. Create and enable the SSL site:
   #cp <apache dir>/sites-available/default <apache dir>/sites-available/ssl
4. Add the following lines to the new SSL site: (Inside the tag <virtual host>)
   SSLEngine On
   SSLCertificateFile <location of the certificate>
5. Finally enable the SSL site with: *#a2ensite ssl*

5.4.3 Captive Portal, Web Based Login to GasSpot

A captive portal is a Web page that the user of a public-access network is obliged to view and interact with before access is granted. Captive portals are typically used by business centers, airports, hotel lobbies, coffee shops, and other venues that offer free Wi-Fi hot spots for Internet users, defined by Whatis.com[9] dictionary.

A Captive Portal technique forces a Web browser on a network to see a special web page, which can be used to authenticate the user before access to Internet is given. Once a user is authenticated their IP and MAC (Media Access Control) address are allowed to pass the gateway.

The Prototype includes a Captive Portal, called GasSpot, which shows how to integrate the Generic SIM Authentication System to authenticate users to a Captive Portal. Figure 54 illustrates the entities needed with GasSpot. Chilli is the gateway that controls the access to Internet; it is implemented using a software called ChilliSpot that will be described later in this section.

The following scenario describes how a user will experience the GasSpot:

1. The user (client) connects to the wireless network GasSpot
2. Chilli assigns the new client an internal IP address
3. When the user opens a web page in the browser, he/she gets redirected to the Generic SIM Authentication System page (see Figure 50).
4. When the user navigates to the "Log In" page, the supplicant Java Applet will be loaded, and the user can start the authentication with the Authenticator (see Figure 51).
5. When the user is authenticated Chilli gives access to Internet, and redirects the browser to the welcome page

The Authenticator uses Chilli as the RADIUS server, which functions as a proxy for the actual RADIUS server (Authentication Server). The Authentication Server uses the GSM Gateway to retrieve credentials for the user.

[9] Whatis.com, http://whatis.techtarget.com/

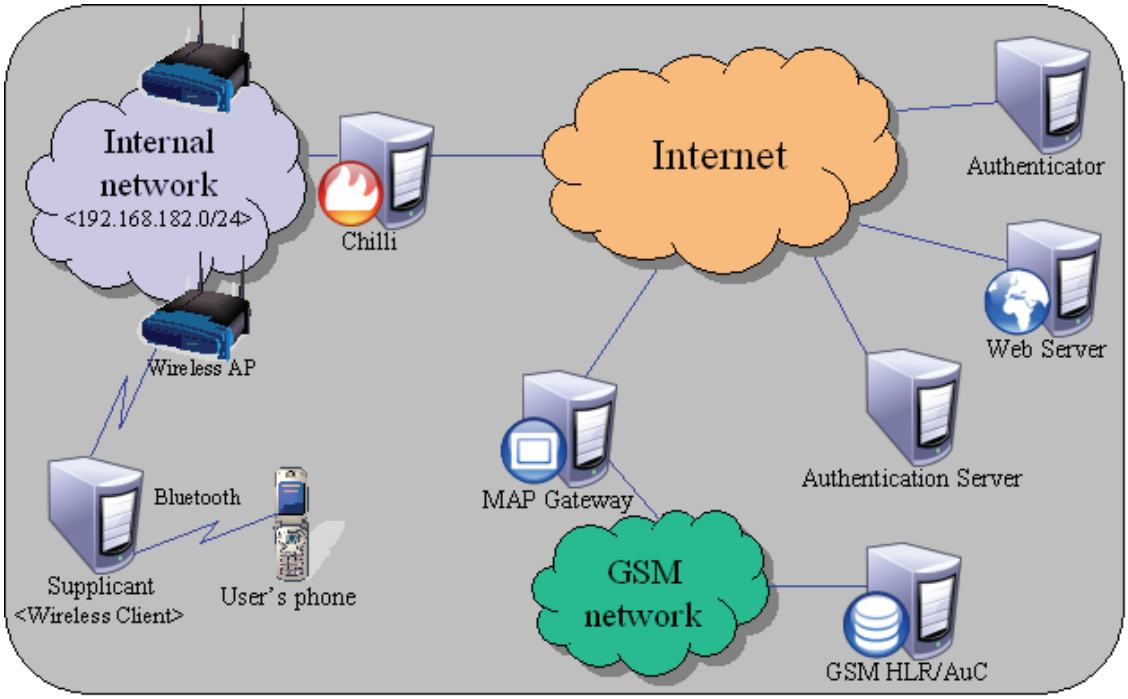


Figure 54: Web Based Login to GasSpot using Chilli

ChilliSpot – Captive Portal

ChilliSpot [80] is an open source captive portal or wireless LAN access point controller. It can be used to authenticate users of a wireless or wired LAN. ChilliSpot supports two different authentication modes: web based login and Wireless Protected Access login (WPA). GasSpot uses a combination of the two modes, where the web based login starts the Generic SIM Authentication System, which use the ChilliSpot gateway as a RADIUS proxy (WPA mode). The computer that runs Chilli needs 2 network interface cards and run the Linux platform.

Chilli can be installed using a binary download or build from source code. To support the GasSpot mode, Chilli needs some small changes to the source code. This is necessary to get Chilli to forward the last response from the RADIUS server, i.e. Access-Accept and Access-Reject.

The changes is to the file *chilli.c*, in the methods*:*

- *int static dnprot_accept(struct app_conn_t *appconn) {*
  Under: *case DNPROT_UAM:*
  Add: *radius_access_accept(appconn);*
- *int static dnprot_reject(struct app_conn_t *appconn) {*
  Under: *case DNPROT_UAM:*
  Add: *radius_access_reject(appconn);*

After the small changes to the source code, follow the instructions in the INSTALL file on how to build the software.

The configuration of Chilli is well documented in the "Release Notes" and "FAQ" found on the ChilliSpot web page [80]. Chilli's configuration file is located at */etc/chilli.conf*. The configuration used in GasSpot can be found in Appendix E.2.

5.5 Motivation for the Technology Choices

In this section, the motivation for choosing the different SIM communication methods, transport protocols, and Authentication Server will be elaborated.

But first of all the reason for using Java will be described. Java is mainly used because of two reasons. First, it is the main programming language used at the Norwegian University of Science and Technology (NTNU). Second, Java ensures cross platform compatibility because the Java Virtual Machine (JVM) makes it possible for Java byte code to run on various systems. This emphasizes the goal to make a generic authentication system. Further, Java is a well-known programming language in the telecommunication business and many open source components exist for Java.

5.5.1 Mobile Terminal Communication

Several possibilities emerged when studying ways of communicating with the SIM through the mobile terminal; AT +CSIM, SATSA (JSR-177), Bluetooth SAP, and using JavaCard with push and pull SMS.

AT +CSIM would have been the genius solution if all phones had implemented it. Since no phones supported +CSIM, this solution was discharged.

SATSA could have been used in conjunction with a Java MIDlet on the mobile phone, but since no phones could be found that supported the APDU optional package needed to retrieve SRESs, it was discharged as well.

The JavaCard solution was only a last resort since it requires a special SIM with an installed application. That resulted in the Bluetooth SAP solution. 25 Nokia models and 11 BenQ-Siemens models were found supporting Bluetooth SAP, and this worked smoothly without implementing anything on the phone. It also worked with all kinds of SIMs.

5.5.2 Smart Card Reader Communication

The choice of a smart card reader interface was discussed in section 5.2.1, and the selection of the PC/SC API and the JPCSC library was explained. The OpenCard Framework (OCF) was also considered used as the Java interface to PC/SC. However since we only need to use the PCSC10CardTerminal and send APDU commands, OCF is unnecessary complex. JPCSC is more lightweight, and therefore more suitable for our purposes.

5.5.3 EAP transport choices

The EAP specification only discusses usage within a point-to-point protocol (PPP), which is a low layer protocol (the “Data Link” layer of the OSI model). The encapsulation of EAP messages in higher layer protocols is not covered by the specification.

This section describes two different approaches to how the EAP message could be represented when it is transported, bytes or XML. This section will also discuss the different transport choices that were considered for this project: UDP, HTTP, TCP, and SOAP. It was important that the transport protocols utilized the already installed TCP/IP protocol suite, and did not require that new communication protocols had to be installed.

Transport mechanisms

User Datagram Protocol (UDP)
UDP is a transport protocol that makes it possible to send short messages (datagrams) to other computers. UDP does not provide reliability and ordering guarantees, however this makes UDP faster and more efficient for lightweight purposes. We need a reliable connection between the Supplicant and the Authenticator, which means that when using UDP the applications need to provide the reliability.

Transmission Control Protocol (TCP)
TCP is a transport protocol that let computers create connections to one another, over which they can exchange the EAP messages. Since TCP gives reliability and ordering guarantees, applications do not have to implement retries and reliability functions thus making them less complex.

Transport Layer Security (TLS) and Secure Sockets Layer (SSL)
TLS is the successor to SSL, which both provide secure communication between computers. Using cryptography they provide endpoint privacy to prevent eavesdropping and spoofing. TLS runs above the TCP transport protocol, which means that it also provides reliable communication. To use TLS the Authenticator need to have a public key certificate, this is used by the Supplicant to establish the secure context.

HyperText Transfer Protocol (HTTP)
HTTP is a well-known request and response protocol used on the Internet between clients and servers. The benefit of using HTTP is that this is a well-defined application level protocol that is well understood. HTTP runs on top of TCP, which add reliability to the protocol, it is also possible to run HTTP on top of TLS called Secure HTTP (HTTPS). A Web Server or a Web Browser normally performs the parsing of HTTP headers. This will require the Authenticator to run a Web Server or implement a complex parser, we chose to drop HTTP transportation because of the extra requirements.

Simple Object Access Protocol (SOAP) and Web Services
SOAP is a simple, lightweight XML-based protocol for exchanging structured and typed information on the Web, and it is part of the web services stack. Web Services is a popular solution nowadays, especially because of the interoperability between various software applications running on different platforms. For our purpose Web Services would require a more rigid implementation that also needs an application server to run at the Authenticator.

Representation

Byte
The EAP specifications describe the message format as byte-oriented, where each field has a specified length and value. When the EAP message is added as an attribute to the RADIUS packet the message has to be byte-oriented. To save the Authenticator from doing translation between two different EAP representation formats, we chose to keep the byte-oriented representation between the Supplicant and the Authenticator also.

Extensible Markup Language (XML)
XML is a text-based general-purpose format capable of describing many different kinds of data. The benefit of using XML as the EAP representation is that it is human-readable (easier to debug) and it is easier to implement the parsing of the message. The disadvantage for our purpose is that we have to map the EAP message format into a XML document. It is also not as efficient as a byte-oriented representation, because the syntax is fairly verbose and sent as text.

5.5.4 RADIUS Server

There are many alternatives when choosing a RADIUS server today. The requirements to the chosen RADIUS server are that it needs to support EAP-SIM authentication, and should also have the possibility for extensions.

NavisRadius was chosen because Telenor R&D already had a testing license, and the proof-of-concept shown at 3GSM [6] showed that NavisRadius worked with EAP-SIM and the Ulticom MAP Gateway. FreeRadius was chosen because it is a free open source RADIUS server with a rich feature support and a modular design. The configuration of the two RADIUS servers was explained in section 5.3.4.

5.6 Evaluation and Testing

Due to time constraints on this thesis some important steps, such as unit testing and user testing, have been left out from the testing of the Prototype.

Under the development and testing of the Prototype a self-signed certificate has been used, which means that browsers do not recognize the certificate. This means that the user must approve the certificate each time a new authentication session is started. In a production system this will be more convenient when an approved certificate is used, typically from VeriSign or Thawte.

5.6.1 Testing Environment for the Prototype

The deployment of the different components in the Prototype (i.e. Authenticator, Radius Server and Service Provider) can be performed in many ways. They can all be on the same host, or spread out on different hosts. The deployment setup is shown in Figure 55. The Authenticator is located at jaguar.stud.ntnu.no, the service provider at tiger.stud.ntnu.no and the RADIUS server at the Telenor IP network on xmmap.nta.no. The RADIUS server uses the MAP Gateway also inside the Telenor IP network.

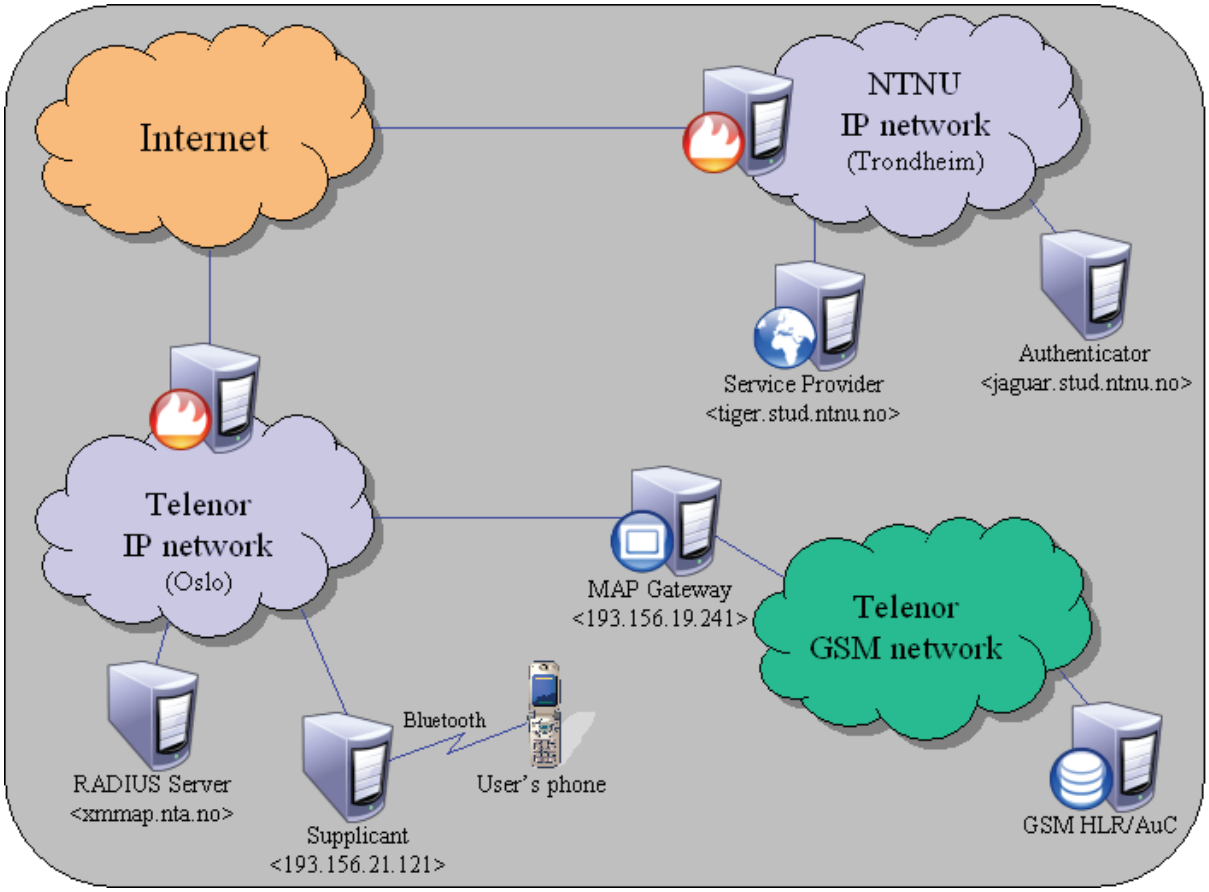


Figure 55: Testing Environment for the Prototype

This deployment setup ensures that the communication is tested outside the local network. Especially when the Supplicant is located in Oslo, the communication goes to the Authenticator in Trondheim, which communicates with the RADIUS server in Oslo. The Authenticator has also been tested extensively, i.e. many simultaneously authentication attempts and running for more than a month.

A packet trace captured with Ethereal (described in 5.3.1) is shown in Appendix F. The trace shows captured network data from a normal run of the Prototype, where only the fields of interest are included.

5.6.2 Software and Hardware Testing

The Prototype has been tested both at the module level and in the final version. First the different modules (e.g. Authenticator, ServiceSupplicant, SimSupplicant) were tested separately, before they were put together. The Prototype was tested with different software and hardware to ensure interoperability. The different components will be explained in the remaining part of this section.

SIM cards

To prove that the Prototype supports all types of SIM cards, the system has been tested with many different cards. The cards used in the testing are produced by: Telenor, Tele2, Netcom, Chess, Axalto (JavaCard), Cingular.

Smart Card readers
The system has been tested with three different types of smart card readers, all supporting PC/SC. The readers tested are: Omnikey CardMan Desktop USB 3121 and 2020, Omnikey CardMan Dongle USB 6121.

Mobile phones
The SIM access function is only tested with two mobile phones, due to the missing support of SAP on many phones. The system is tested on:

- Nokia N91 (pre-release version)
- Nokia 6230i

When testing the Bluetooth Inquiry module the "Dial-up networking service" was used instead of the "SIM access", because most mobile phones support this service.

Bluetooth USB Dongle
The Prototype is tested with following Bluetooth USB dongles:
Works:

- Jensen Blue:Link 101, Broadcom Blutonium BCM2035 (v 1.2)
- EPoX BT-DG06, Cambridge Silicon Radio Ltd (v 1.2)
- EpoX BT-DG07A+, Cambridge Silicon Radio Ltd (v 2.0)
- CNet CBD-220, Cambridge Silicon Radio Ltd (v 2.0)

Do not work:

- EPoX BT-DG03 (v 1.1), Cambridge Silicon Radio Ltd
- CNet CBD-101, Silicon Wave Bluetooth Wireless Adapter (v 1.2)

Operating Systems
To ensure that the Prototype is portable across different operating system, it has been tested on the following platforms: Windows XP, Mac OS X Tiger and Linux (Ubuntu 5.10 and 6.06, SuSE 10.0, Fedora Core 4).

Web Browsers
To guarantee that the service provider and the Java Applet supplicant work on different browsers, it was tested on the most popular available today: Internet Explorer, Firefox, Opera and Safari.

5.7 Summary

In this chapter an example of a deployment scheme has been shown for a realization of a Prototype for the Generic SIM Authentication System. The implementation and configuration phase could be used to improve the Prototype, as advices in another similar implementation, or when implementing or configuring one or more of the elements elaborated in this chapter.

A produced similar system should take into considerations the points mentioned in the evaluation and further test the system beyond the testing done in relation with this thesis.

For further comments on choices made and security considerations, chapter 6 should be read.

6 Critical Review

This chapter starts by giving a critical evaluation of the methodology used in this thesis. The generality of the implemented prototype is discussed in section 6.2. A generic authentication system like the one implemented in the Prototype also have some security concerns connected to it. The security considerations will be elaborated in section 6.3.

6.1 Methodology

The methodology used in this thesis was described in section 1.6, where the 5 different phases of Design Research was described. Due to time constraints on this thesis one important step in phase 4 (Evaluation) have been left out from the methodology. The functional criteria of the testing were not made in the suggestion phase.

The Prototype has been extensively tested during the development phase, however without using a formalized testing framework. An implementation of the full system should follow a testing framework, such as JUnit [81] or similar.

No standardized software development process was followed in the analysis and design phase. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process. Most of the models were found to be too complicated and heavy. Nevertheless a development process used internally at Telenor R&D provided a good starting point for the analysis and development. This development methodology is based on the Unified Software Development Process but is much lighter to enable more efficient development. The Unified Software Development Process uses the Unified Modeling Language (UML) [62], and is use-case driven, architecture-centric, iterative and incremental.

6.2 The Generality of the Implemented Prototype

This section discusses the current realization of the Prototype with respect to availability and reliability, scalability and openness.

6.2.1 Availability and Reliability

The availability and reliability of the system is concerned with system failure and its associated consequences. If the authentication system fails the users will not get connected with the wanted services, even if the service is available. The authentication system must have mechanisms to ensure:

- Detection of system failures
- Notification when a failure has occurred
- Specified procedures if a failure occurs
- Prevention of failures

The Prototype has some mechanisms to detect system failures. If one of the components does not respond in the communication, a timeout will occur and the user will be notified with an error message. When an error occurs at the Supplicant the user will be notified and the system exits. When an error occurs at the Authenticator an error message will be written to the log file, and the thread handling the connection is killed. Only when a major error occurs the Authenticator will shutdown.

6.2.2 Scalability

The authentication system must be built to scale gracefully in a distributed environment. Scaling gracefully means that the application maintains its functional integrity and uses available capacity effectively as it is distributed across multiple computers. This will ensure that the backend of the system can support large numbers of clients and services. When an application is distributed on many servers different load balancing techniques will be used, this will make sure that the load is shared on servers and over CPU on different computers. Each server should be stateless in the sense that the architecture does not store state information related to the transaction of each client. A stateless architecture can be scaled up to include multiple copies for each server type to handle increased load.

A scalable authentication system will also help against denial-of-service (DoS) attacks. DoS attacks are a very common type of attack and are aimed at breaking servers or network resources that are necessary to provide services.

In the current deployment of the Prototype the Authenticator and RADIUS server runs on a single server. This may be a bottleneck for the authentication system, however this is only a Prototype and a real deployment of the system should distribute the authenticator and RADIUS server on many servers. To improve the reliability of the service the servers should also be geographically distributed in case of a network failure.

The messages being sent in GAS are relatively short, so the most important factor for the scalability is the number of messages and the calculation needed for each message. Given EAP-SIM, 10 messages are sent between the Supplicant and Authenticator, 6 between Authenticator and Authentication Server, and 2 between Authentication Server and GSM network. There are some calculations involved, per message, but the slowest part of the GAS is actually between Supplicant and SIM. Nevertheless, this does not affect the scalability of the system. This means that a production system should have more Authenticators than Authentication Servers and only enough GSM Gateways to have redundancy.

6.2.3 Openness

The Generic SIM Authentication system uses open standard communication protocols and open source libraries. This should make the system portable and ensure interoperability.

6.3 Security Considerations

6.3.1 Bluetooth SAP

The first security concern with Bluetooth is the passkey. As with any key, long keys are more secure than short ones. If a hacker/cracker is able to discover the passkey, he/she can calculate possible initiation keys, and then from that, calculate the authentication key. Making the passkey long will make it much harder to accomplish the first step.

The mandatory 16 digits passkey in SAP results in a 128 bits authentication key used in the authentication process between the devices. The authentication key is used when creating the encryption keys and it's important that this key is at least as big as the encryption key. Each time encryption is activated; a new encryption key shall be generated. Thus, the lifetime of the encryption key does not necessarily correspond to the lifetime of the authentication key. The algorithms for the different keys are out of the scope of this thesis, but can be found in the Bluetooth specification [54].

The Bluetooth SAP specification [55] specifies that authentication between the SAP devices is mandatory, together with minimum 64 bits encryption, and either security mode 2 or 3 (see section 2.7.1). New Bluetooth devices support up to 128 bits encryption, and the highest encryption possible is recommended and used if possible. The messages sent throughout an authentication process are few and short. This emphasizes that the Bluetooth encryption is relatively safe, even if 64 bits encryption is regarded rather insecure.

Nevertheless, a few pointers to the users of such a system should be given:

- Never use fixed passkeys, should be randomly generated.
- Class 1 Bluetooth equipment (range of up to 100 meters) has a bigger security risk, because hidden hackers may try to access your system.
- Keep your system updated to be able to use the newest security capacities (recommend equipment supporting 128 bit encryption).
- Turn off SAP access on the phone when not using it, preferably also Bluetooth.

Near Field Communication (NFC), second generation Radio Frequency Identity (RFID), could be used to set up a secure Bluetooth connection. NFC only works in theory up to about 10 centimeters (in a real environment only a couple of centimeters) and will further improve the security of Bluetooth SAP.

6.3.2 EAP-SIM

The authentication in EAP-SIM is mainly based on the GSM authentication with some improvements. In this section some weaknesses and possible solutions will be discussed.

A3- /A8-algorithm

The biggest problem with GSM is that the network does not need to authenticate itself to the MS. Thus, it is possible for an attacker to set up a false base station with the same Mobile Network Code (MNC) as the subscriber's network. The attacker could just send an IDENTITY REQUEST (type=IMSI), and it will get the IMSI in clear text.

The most common implementation of A3 and A8 is rolled into a single algorithm called COMP-128. Comp-128 has a flaw that has shown that four of the output bytes are directly linked to four of the input bytes (2 bytes from the Ki and 2 bytes from the RAND). For carefully chosen RAND values it is possible using what is called a 2R attack to find the Ki with 2^{17} (a conservative number) RAND attempts. With a false BS, it is possible to send AUTHENTICATION REQUESTs (RAND) and get SRESs in response. Within 9 hours, the attacker will have gained the Ki using a 2R attack, and can impersonate the subscriber. With more sophisticated methods (Dejan Kaljevic's SIM Scan, which uses 2R, 3R, 4R, and 5R attacks) it could be possible to gain the Ki in one hour. It will also be possible to eavesdrop on an encrypted conversation, because the Kc can be calculated when the Ki is known. Newer versions of COMP-128, namely COMP128-2 and COMP128-3, are on the market and has made the cloning of SIMs unfeasible. The information above is found in [82].

Dejan Kaljevic's SIM Scan has been tested. On the fourth trial, a SIM card from 1999 was successfully broken and the Ki was retrieved.

Pseudonyms
EAP-SIM includes optional identity privacy support that protects the privacy of the subscriber identity against passive eavesdropping. The EAP-SIM specification only specifies a mechanism to deliver pseudonyms from the server to the peer as part of an EAP-SIM exchange. Hence, an Authentication Server that has not yet performed any EAP-SIM exchanges does not typically have a pseudonym available. This section is an excerpt from the EAP-SIM protocol.

Mutual Authentication and Triplet Exposure
The mutual authentication is based on the Kc, and care should be taken not to expose the Kc keys to attackers when they are stored or handled by the Authentication server. It is also important to have secure communication between the HLR and the Authentication Server and between the SIM and the Service Supplicant where the Kcs are sent. If the same SIM credentials are also used for GSM traffic, the triplets could be revealed in the GSM network. But given the size of RAND, the probability is small given that the size of RAND is 128 bits.

In GSM, the network is allowed to re-use the RAND challenge in consecutive authentication exchanges. This is not allowed in EAP-SIM. The EAP-SIM server is mandated to use fresh triplets (RAND challenges) in consecutive authentication exchanges. EAP-SIM does not mandate any means for the peer to check if the RANDs are fresh, so the security of the scheme leans on the secrecy of the triplets.

The UMTS AKA, which is used in EAP-AKA, has improved the re-use of authentication vectors risk and should be used when the triplet re-use properties of EAP-SIM are not sufficient.

The strength of the keys
The information below is mainly excerpts from the EAP-SIM protocol. EAP-SIM combines several GSM triplets in order to generate stronger keying material and stronger AT_MAC values. The actual strength of the resulting keys depends, among other things, on operator-specific parameters including authentication algorithms, the strength of the Ki key, and the quality of the RAND challenges. For example, some SIM cards generate Kc keys with 10 bits set to zero. Such restrictions may prevent the concatenation technique from yielding strong session keys. Because the strength of the Ki key is 128 bits, the ultimate strength of any derived secret key material is never more than 128 bits.

There is no known way to obtain complete GSM triplets by mounting an attack against EAP-SIM. But, a passive eavesdropper can learn n*RAND and AT_MAC and may be able to link this information to the subscriber identity. An active attacker that impersonates a GSM subscriber can easily obtain n*RAND and AT_MAC values from the EAP server for any given subscriber identity. However, calculating the Kc and SRES values from AT_MAC would require the attacker to reverse the keyed message authentication code function HMAC-SHA1-128.

As EAP-SIM does not expose any values calculated from an individual GSM Kc keys, it is not possible to mount a brute force attack on only one of the Kc keys in EAP-SIM.

The problem with the keys, as stated in [83], is that if two or three similar RANDs are sent in the challenge, the strength of the key is reduced to 64 bits. The probability of two similar RAND is 1/(2^128) → equal zero, but an Impersonator may do this on purpose.

Summation of EAP-SIM security considerations
The EAP-SIM has some weaknesses, but most of them are easy to correct. The Generic SIM Authentication System will include these solutions to address some of the possible security holes:

- Control that all of the RANDs are different.
- Control that 2 or 3 RANDs are used.
- Include authentication of the Authenticator on transport level (TLS). Require a signed certificate from a trusted CA.
- Require that the communication between the Supplicant and the Authenticator is encrypted.
- Secure the communication between the SIM and the Service Supplicant.

The biggest problem is the communication between the HLR and the Authentication Server where a proprietary protocol is used. This communication channel should be secure.

6.3.3 Transport Layer Security (TLS)

To secure the communication between the Supplicant and the Authenticator, TLS has been used. TLS is the successor of SSL. SSL uses public-key cryptography to exchange a set of shared keys, and then uses standard shared-key encryption to exchange data. The shared keys are used both for encrypting the data (making it unreadable by others) and for authenticating the data (ensuring that it hasn't come from an impostor).

Some early implementation of SSL used short keys (40 bits), but TLS and other newer version uses at least 128 bit in the symmetric key encryption. This is considered safe, and given the short duration of the authentication process; hackers will have little data to work on. This only improves safety.

TLS 1.1 has now been released RFC 4279 [84]. The main reason for the new version is a modified format for the encrypted RSA pre-master secret (part of the client key-exchange message (if RSA is used)) to use PKCS#1 v 2.1. This is done to protect against an attack discovered by Daniel Bleichenbacher [85], which can be launched against TLS 1.0 servers, using PKCS#1 v 1.5. Unfortunately, TLS 1.0 seems to be used in Java SSL communication.

The SSL communication is no safer than its implementation, and therefore TLS (the newest SSL algorithm) is chosen with RSA as the algorithm. RSA, devised by Ron Rivest, Adi Shamir, and Len Adleman, is considered unfeasible to break as long as the key is bigger than n=1024. Nevertheless, n=1024 is reckoned to be broken in the future and n=2048 or bigger is therefore recommended. The default keysize in Java Keytool is 1024 and should be changed if used commercially.

Even though a few potentially security holes exist, TLS is considered very safe.

6.3.4 Cryptographic Algorithms

There has been some writing about weaknesses in the SHA-1 algorithm. This is an excerpt from the NIST's Computer Security Resource Center (CSRC), last updated March 30, 2006: "*NIST was recently informed that researchers had discovered a way to "break" the current Federal Information Processing Standard SHA-1 algorithm, which has been in effect since 1994. The researchers have not yet published their complete results, so NIST has not confirmed these findings. However, the researchers are a reputable research team with*

expertise in this area. ... Due to advances in computing power, NIST already planned to phase out SHA-1 in favor of the larger and stronger hash functions (SHA-224, SHA-256, SHA-384 and SHA-512) by 2010." The reported attacks require an estimated work factor of 2^69 (approximately 590 billion billion) hash computations to find a collision (reported February 2005). While this is well beyond what is currently feasible using a normal computer, this is potentially feasible for attackers who have specialized hardware. These stronger hash functions are therefore recommended. There is however nothing that indicates that it will affect other application using the SHA-1, like the HMAC.

The MD-5 is considered weaker than the SHA-1, and in [86] there has been shown how to find collisions in MD-5 and break it. It reports that using an IBM P690 it uses a little more than an hour to have two different messages produce the same hash value (a collision). But, because of the way hash functions are used in the HMAC construction, the techniques used in the attacks do not apply to break HMAC.

Once a collision has been found, additional collisions can be found trivially by concatenating data to the matching messages. This passes for all hash functions.

According to Cryptography Research [87], TLS or SSL 3.0 should not be affected by any of these attacks. There has been some concern about CAs. A devastating attack would be one that enabled adversaries to obtain a legitimate server certificate with a collision to one containing a wildcard for the domain name and an expiration date far in the future

7 Conclusion

This master thesis has proposed a novel design of a generic authentication system based on SIM, together with a detailed description of a prototype. The system has been tested end-to-end and provides a strong authentication mechanism. New services can easily be supported, such that these can benefit from strong authentication. By gradually implementing more authentication mechanisms (e.g. OTP and PKI) on the SIM, it will be able to support several levels of security. This will result in a generic authentication system approving security needs for nowadays and also for the future.

The advantages of the Generic SIM Authentication System can be summarized as follows:

- User do not need to remember many username and password combinations
- User friendliness (uses an existing device (the mobile phone with a SIM))
- Secure and tamper-resistant
- The GSM system has a large existing customer base
- Simple deployment and administration of tokens
- Easy for service providers to add strong authentication to their service (outsourcing of authentication function)

7.1 Achievements and Results

The Generic SIM Authentication System (GAS) builds upon the existing GSM authentication infrastructure, thus allow re-use of GSM expertise from the mobile operators. The non-existing components of the system are the Supplicant and the Authenticator. It also might be necessary to implement plug-ins to the Authentication Server. The main functionality of the Authenticator is to locate a suitable Authentication Server, act as a translator from EAP to RADIUS and back, store information about authorized Supplicants, and keep track of identities when for instance temporary identities are used. It is the Authentication Server that authenticates the Supplicants on behalf of the Authenticator. The Authentication Server uses a GSM gateway to communicate with the GSM network and the HLR/AuC. The Ulticom MAP Gateway has been tested towards a real GSM network and operates as expected.

The communication between the Supplicant and the other components in the authentication system is based on EAP, which is a general authentication protocol supporting multiple authentication methods. It runs on top of TCP between the Supplicant and the Authenticator, and on top of UDP/RADIUS between the Authenticator and the Authentication Server. When performing the GSM authentication the EAP-SIM method is used.

To exchange credentials with the SIM, the SIM must be located in a SIM reader. The SIM reader communicates with the SIM by sending and receiving APDUs. There are two main classes of SIM readers, the integrated reader in the mobile phone and the smart card reader. To communicate with the SIM through the mobile phone the phone must support a SIM access interface, for instance the Bluetooth SAP. The PC/SC interface has become the de-facto standard when communicating with the smart card reader.

A Prototype has been developed in Java to demonstrate the GAS, and includes both a client (Supplicant) and a server (Authenticator) part. The Supplicant has two versions; a Java Applet intended to be used with web applications and a standalone Java Application to be used with other IP-based services (e.g. e-mail, SIP). Different SIM access interfaces (AT+CSIM, SATSA, and SAP) were considered and tested during the development of the Prototype.

However, because of availability and security reasons SAP seems like the most promising technology for the future and is the one implemented in the Prototype. The Prototype has been tested thoroughly with deployment of the components on different locations and operating systems; it has also been tested with different hardware to ensure interoperability.

Three different services have been developed to demonstrate how easily the SIM authentication can be integrated. The first demo service shows how to integrate the authentication with JSP technology and Apache Tomcat. The second service, MyService, is another example of how the authentication service could be integrated into a web portal using PHP to demonstrate that the Prototype is independent of the service implementation language. MyService also illustrates how the service provider can control the registration of new users and link up with their SIM identity. The last service, GasSpot, shows how to integrate the GAS to authenticate users to a Captive Portal. The access is controlled by the gateway, which is implemented using ChilliSpot.

Based on the master thesis, the authors have written a paper called "A Generic Authentication System based on SIM". The paper has been accepted for the IARIA International Conference on Internet Surveillance and Protection (ICISP'06), in Cap Esterel, Côte d'Azur, France, August 26-29, 2006, shown in Appendix H.

7.2 Future Work

The GAS can be easily extended as an authentication mechanism in a Single Sign-On (SSO) system. Liberty Alliance and their federated identity management have been considered. By integrating the Authenticator with an Identity Provider (IDP) and extending the Service Providers according to Liberty Alliance it is possible to have an SSO where SPs trust an IDP, which vouches for the Supplicant after an authentication.

The GAS should include a standardized Service Provider API, and be integrated and tested with more services, like an SSO system, SIP, e-mail, access control etc.

Other authentication mechanisms should be implemented to support several security levels. OTP and PKI have been tested on SIMs, and they are already in use on smart card, which is the same technology as SIM. An implementation with EAP-AKA and USIM will be even more secure, and should be considered.

To make the communication between the PC and the mobile phone more protected, NFC should be implemented instead of Bluetooth.

A Supplicant version that is integrated on a Pocket PC/PDA, i.e. the SIM is located with the Supplicant, should also be developed.

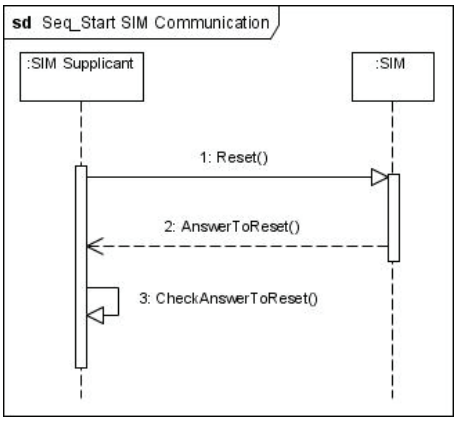
Appendix A UML Diagrams

This section shows the additional different UML 2.0 diagrams used in the design of the Generic SIM Authentication System (not used in chapters 3 and 4).

A.1 Sequence Diagrams

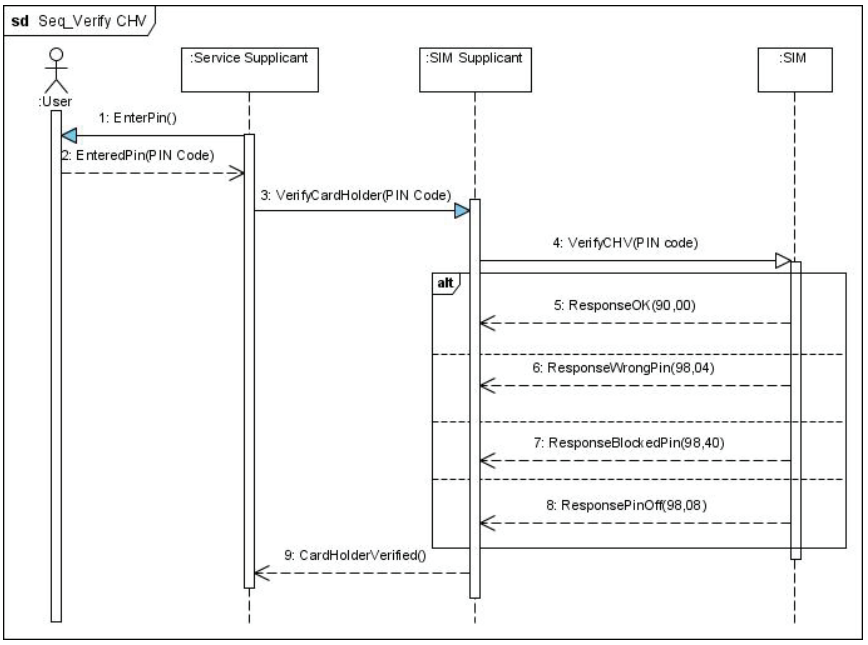
Start SIM Communication

Resets the SIM, and interprets the Answer To Reset (ATR) that includes information about the SIM.



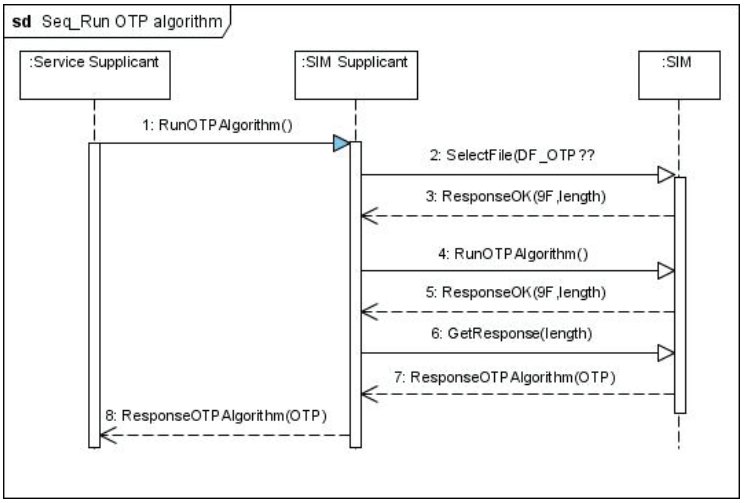
Verify CHV

The user enters the PIN and it is sent to the SIM to give the Supplicant access to PIN protected files and algorithms, e.g. Run GSM Algorithm.



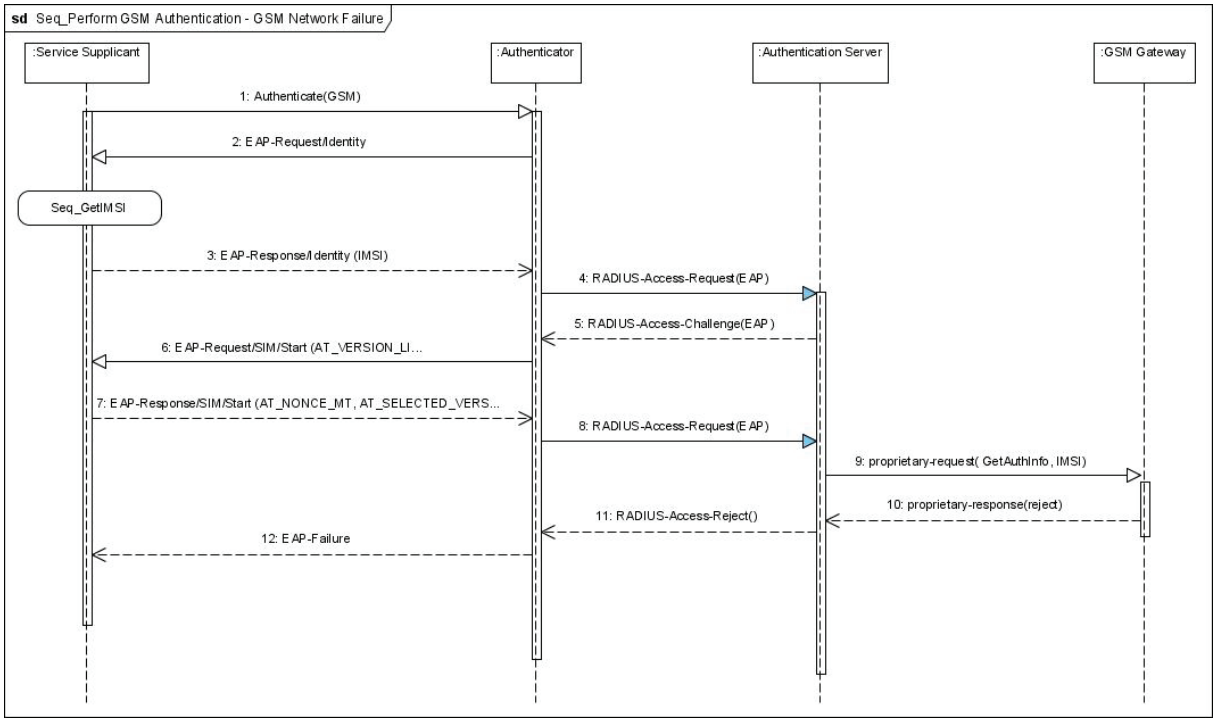
Run OTP Algorithm

The diagram shows a plausible scheme of how to extract an OTP from the SIM.



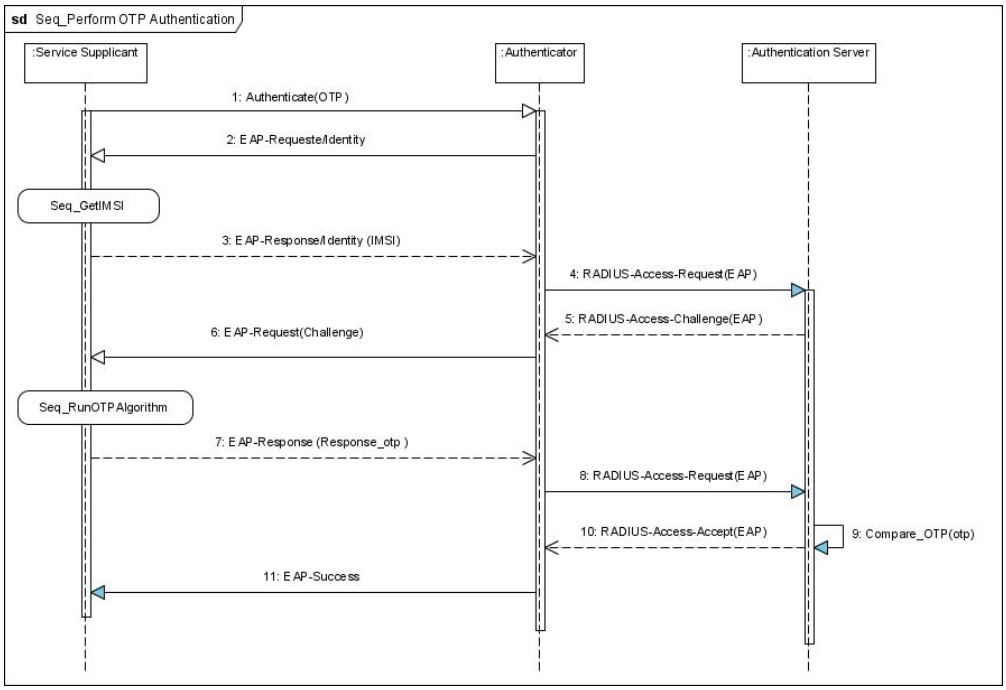
Perform GSM Authentication – GSM Network Failure

The diagram shows a plausible failure during a GSM Authentication. The Authentication Server is not able to communicate with the GSM Network (HLR/AuC) through the GSM Gateway. The Authentication Server either act on a time-out or on a failure message from the GSM Gateway.



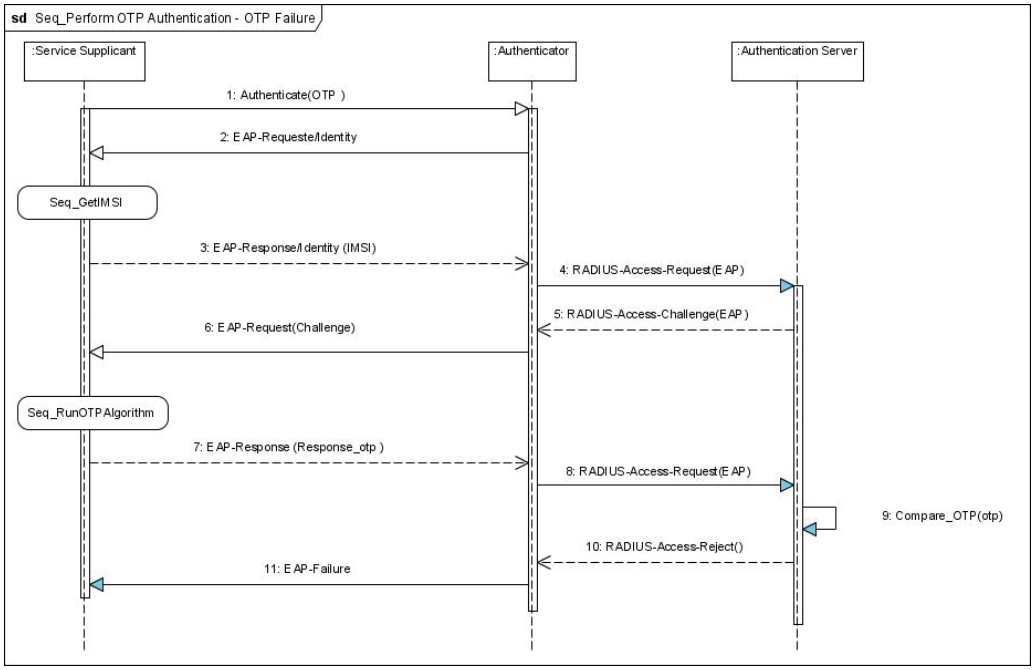
Perform OTP Authentication

Shows how an OTP authentication could be performed towards an Authentication Server supporting the OTP authentication method.



Perform OTP Authentication – OTP Failure

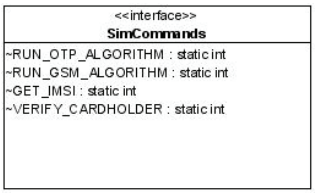
Shows what happens when the OTP sent to the Authentication Server is not correct. The Access-Reject and EAP-failure messages will also be accurate when a wrong MAC is presented during an SIM Authentication.



A.2 Class Diagrams

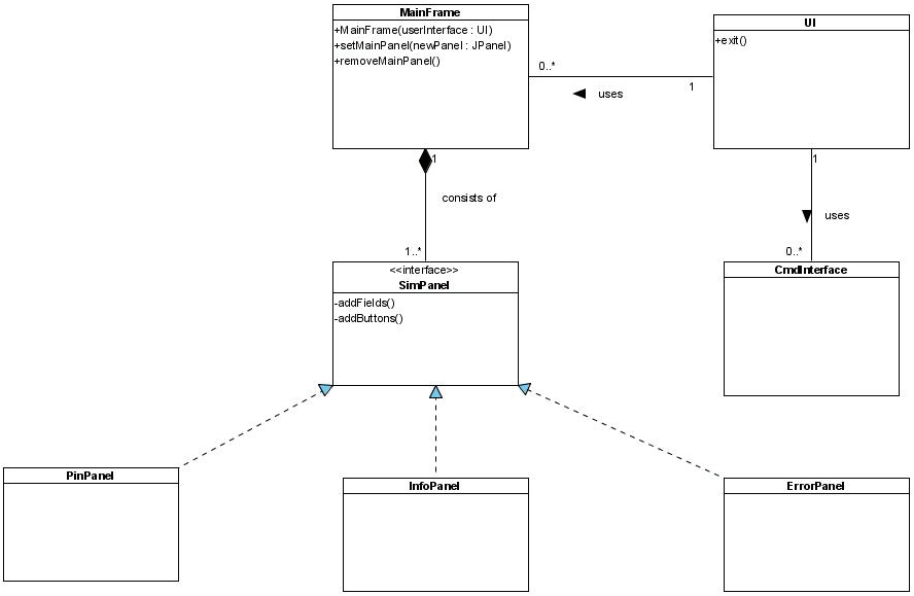
gas.supplicant.SimCommands

Contains constants used by the SIM Supplicant and the Service Supplicant to identify which methods to perform.



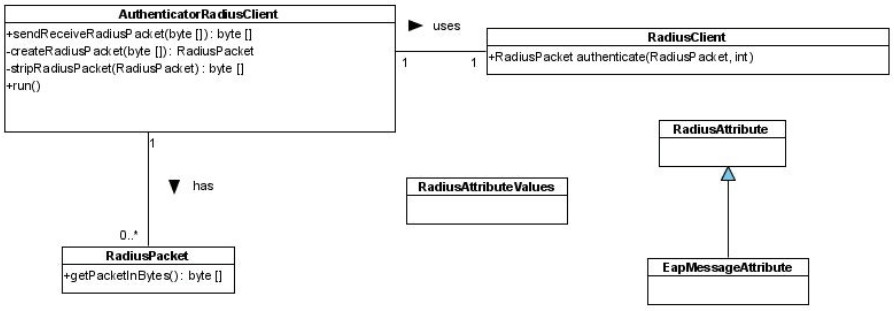
gas.supplicant.serviceSupplicant.userInterface

The package contains the classes responsible for communicating with the user, either via a Graphic User Interface (GUI) or via the command interface. The MainFrame-class and the Panel-classes contains the GUI representation.



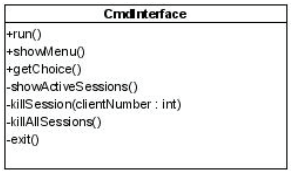
gas.authenticator.radiusClient

This package is responsible for communicating with an Authentication Server, and in this case a RADIUS server. This package wraps and unwraps EAP-messages into RADIUS packets and sends/receives them to/from the server.



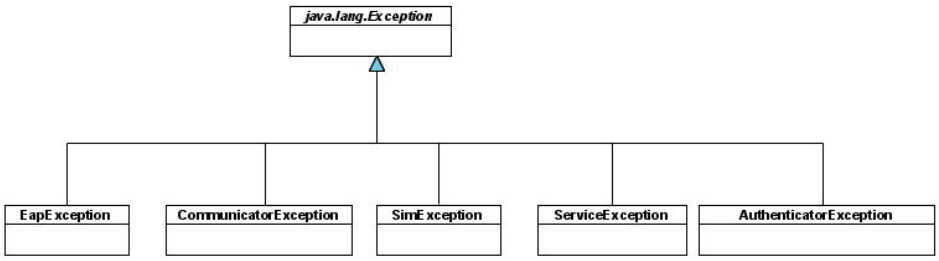
gas.authenticator.userInterface

The package contains a CmdInterface-class that gives the user a possibility to interact with ongoing processes at the Authenticator. It support showing active sessions, killing of sessions, etc, and is meant for a typical administrator of the Authenticator system.



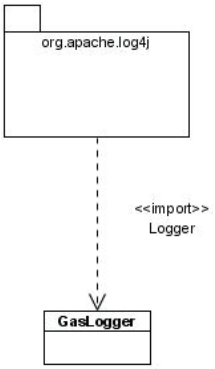
gas.exception

Contains the range of exception-classes used in the Generic SIM Authentication System.



gas.log

Contains a GasLogger imported by log4j that handles info-, debug-, warning-, and error-messages. These messages could be written to a log-file or directly to the command interface with the possibility of adding the point in time and different color for the different messages.



Appendix B Use Case Template

The Use Case templates used in this thesis are based on [61].

Normal use case and included use case

Use Case	*Use Case identifier and modification history.* <Number>. <Name> **History** created <date>, modified <date>
Description	*Goal to be achieved by use case and sources for requirement.*
Actors	*List of actors involved in use case.*
Assumptions	*Conditions that must be true for use case to terminate successfully.*
Steps	*Interactions between actors and system that are necessary to achieve goal. On the form:* 1. Interaction A 2. IF something A THEN 2.1 Interaction B ELSEIF something B THEN 2.2 Interaction C 3. RETURN something C
Variations	*Any variations in the steps of a use case. On the form:* #1 Interaction A fails, RETURN failure
Issues	*List of issues that remain to be resolved.*

Use Case extension

Use Case Extension	*Extension and Use Case identifier, and modification history.* <extension identifier> **extends** <use case identifier> **History** created <date>, modified <date>
Description	*Goal to be achieved by use case and sources for requirement.*
Condition	*The condition that must be satisfied if the extension is to take place.*
Steps	*Changes to use case steps.*
Variations	*Any variations in the steps of a use case.*
Issues	*List of issues that remain to be resolved.*

Appendix C The Cryptographic Java Implementation

This section shows the implementation of the Crypto-class that belongs to the gas.communicator-package.

```
/**
* Generic SIM Authentication System (GAS)
* Copyright (C) 2006, Lars Lunde & Audun Wangensteen
*
* This file is part of GAS.
*
* GAS is free software; you can redistribute it and/or modify it under the terms of the GNU General Public
* License  as published by the Free Software Foundation; either version 2 of the License, or (at your option) any
* later version.
*
* This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without
* even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License along with this program; if not, write to
* the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
*/
package gas.communicator;
import java.math.BigInteger;
import gas.exception.EapException;

/**
* Crypto provides all of the cryptographic algorithms used in the Generic SIM Authentication System.
* It includes, HMAC-128, MD-5, SHA-1 (160), and EAP-SIM Pseudo-Random Number Function (PRNG).
*
* @author Lars Lunde
* @author Audun Wangensteen
* @since 22.feb.2006
*/
public class Crypto {
/** A standard variables for the G Function (SHA-1 and MD-5) */
private final static int h0 = 0x67452301;
/** A standard variables for the G Function (SHA-1 and MD-5) */
private final static int h1 = 0xEFCDAB89;
/** A standard variables for the G Function (SHA-1 and MD-5) */
private final static int h2 = 0x98BADCFE;
/** A standard variables for the G Function (SHA-1 and MD-5) */
private final static int h3 = 0x10325476;
/** A standard variables for the G Function (SHA-1) */
private final static int h4 = 0xC3D2E1F0;
/** Every method in the Crypto class uses a block size of 64 bytes*/
private final static int BLOCK_SIZE = 64;
/** outer pad in hmac */
private final static int IPAD = 0x36;
/** inner pad in hmac */
private final static int OPAD = 0x5C;
/** Internal value telling HMAC to use SHA-1 as the hash-function */
public final static int SHA1 = 1;
/** Internal value telling HMAC to use MD-5 as the hash-function */
public final static int MD5 = 2;
```

```
/**
* MD5 makes a hash message digest of 160 bits = 20 bytes. <br />MD-5 is a hash function, it is specified in
* RFC1321. The first part of the method pads the incoming text into a multiple of BLOCK_SIZE (64 bytes). All
the
* internal shifts is done in the md5Function().
*
* @param text the byte array that needs to be hashed
* @return 128 bits hashed value in a byte array
* @throws EapException If some internal calls fails
*/
public synchronized static byte[] md5(byte[] text) throws EapException {
    int textLength = text.length;
    // Checks if any remaining bytes according to n*BLOCK_SIZE
    int n = (int)(textLength % BLOCK_SIZE);
    //Checks if it is room for padding of 10000000 (1 byte) and 64 bits (8 bytes) of length
    int totalLength;
    if ((BLOCK_SIZE-n) < 9)
        totalLength = textLength + BLOCK_SIZE + BLOCK_SIZE - n;
    else
        totalLength = textLength + BLOCK_SIZE - n;

    /* Pad remaining bytes in buffer */
    // STEP 1: puts the last n bytes of byte[] text into the empty array, all zeroes
    //this means automatically zeroes in every spot not specified
    byte[] tmpArray = new byte[totalLength];
    System.arraycopy(text, 0, tmpArray, 0, textLength);
    // STEP 2: padding is always binary 1 followed by binary 0s followed by original length (last 64 bits)
    tmpArray[textLength] = (byte) 0x80;
    //STEP 3: the last 64 bits (8 bytes) are the length of the original message we are counting bits, so
    //multiplying with 8=2^3
    long bits = textLength << 3;
    tmpArray[totalLength-8] = (byte) bits; tmpArray[totalLength-7] = (byte)(bits >>> 8);
    tmpArray[totalLength-6] = (byte)(bits >>> 16); tmpArray[totalLength-5] = (byte)(bits >>> 24);
    tmpArray[totalLength-4] = (byte)(bits >>> 32); tmpArray[totalLength-3] = (byte)(bits >>> 40);
    tmpArray[totalLength-2] = (byte)(bits >>> 48); tmpArray[totalLength-1] = (byte)(bits >>> 56);
    /* Padding ends */

    if((totalLength % BLOCK_SIZE) != 0)
        throw new EapException("Something is wrong with the padding to MD-5!");
    byte[] buffer = new byte[BLOCK_SIZE];

    int[] gResult;
    //initiates the standard variables
    int hh0 = h0; int hh1 = h1; int hh2 = h2; int hh3 = h3;

    //Stepping through n*M(512 bits = 64 bytes), using MD5function
    for (int i = 0; i < totalLength; i+= BLOCK_SIZE) {
        System.arraycopy(tmpArray, i, buffer, 0, BLOCK_SIZE);
        gResult = md5Function(buffer, hh0, hh1, hh2, hh3);
        hh0 = gResult[0]; hh1 = gResult[1]; hh2 = gResult[2]; hh3 = gResult[3];
    }
    text = new byte[] {
        (byte) hh0, (byte)(hh0 >>> 8), (byte)(hh0 >>> 16), (byte)(hh0 >>> 24),
        (byte) hh1, (byte)(hh1 >>> 8), (byte)(hh1 >>> 16), (byte)(hh1 >>> 24),
        (byte) hh2, (byte)(hh2 >>> 8), (byte)(hh2 >>> 16), (byte)(hh2 >>> 24),
        (byte) hh3, (byte)(hh3 >>> 8), (byte)(hh3 >>> 16), (byte)(hh3 >>> 24)
    };

    return text;
}
```

```
/**
* SHA1 makes a hash message digest of 160 bits (20 bytes). <br />SHA-1 or SHA-160 is a hash function, it is
* specified in NIST Federal Information Processing Standards (FIPS) Publication 180-2.
*
* @param text the byte array that needs to be hashed
* @return 160 bits hashed value in a byte array
* @throws EapException If some internal calls fails
*/
public synchronized static byte[] sha1(byte[] text) throws EapException{
	int textLength = text.length;
	// Checks if any remaining bytes according to n*BLOCK_SIZE
	int n = (int)(textLength % BLOCK_SIZE);
	//Checks if it is room for padding of 10000000 (1 byte) and 64 bits (8 bytes) of length
	int totalLength;
	if ((BLOCK_SIZE-n) < 9)
		totalLength = textLength + BLOCK_SIZE + BLOCK_SIZE - n;
	else
		totalLength = textLength + BLOCK_SIZE - n;

	/* Pad remaining bytes in buffer */
	//STEP 1: puts the last n bytes of byte[] text into the empty array, all zeroes
	//this means automatically zeroes in every spot not specified
	byte[] tmpArray = new byte[totalLength];
	System.arraycopy(text, 0, tmpArray, 0, textLength);
	// STEP 2: padding is always binary 1 followed by binary 0s followed by original length (last 64 bits)
	tmpArray[textLength] = (byte) 0x80;
	//STEP 3:the last 64 bits (8 bytes) are the length of the original message
	//we are counting bits, so multiplying with 8=2^3
	long bits = textLength << 3;
	tmpArray[totalLength-8] = (byte)(bits >>> 56); tmpArray[totalLength-7] = (byte)(bits >>> 48);
	tmpArray[totalLength-6] = (byte)(bits >>> 40); tmpArray[totalLength-5] = (byte)(bits >>> 32);
	tmpArray[totalLength-4] = (byte)(bits >>> 24); tmpArray[totalLength-3] = (byte)(bits >>> 16);
	tmpArray[totalLength-2] = (byte)(bits >>> 8); tmpArray[totalLength-1] = (byte) bits;
	/* Padding ends */
	if((totalLength % BLOCK_SIZE) != 0)
		throw new EapException("Something is wrong with the padding to MD-5!");

	//initiates the standard variables
	int hh0 = h0; int hh1 = h1; int hh2 = h2; int hh3 = h3; int hh4 = h4; int[] gResult;
	byte[] buffer = new byte[BLOCK_SIZE];
	//Stepping through n*M(512 bits = 64 bytes), suing G function
	for (int i = 0; i < totalLength; i+= BLOCK_SIZE) {
		System.arraycopy(tmpArray, i, buffer, 0, BLOCK_SIZE);
		gResult = gFunction(buffer, hh0, hh1, hh2, hh3, hh4);
		hh0 = gResult[0]; hh1 = gResult[1]; hh2 = gResult[2]; hh3 = gResult[3]; hh4 = gResult[4];
	}
	//transforming the int array into a byte array
	text = new byte[] {
		(byte)(hh0 >>> 24), (byte)(hh0 >>> 16), (byte)(hh0 >>>8), (byte) hh0,
		(byte)(hh1 >>> 24), (byte)(hh1 >>> 16), (byte)(hh1 >>>8), (byte) hh1,
		(byte)(hh2 >>> 24), (byte)(hh2 >>> 16), (byte)(hh2 >>>8), (byte) hh2,
		(byte)(hh3 >>> 24), (byte)(hh3 >>> 16), (byte)(hh3 >>>8), (byte) hh3,
		(byte)(hh4 >>> 24), (byte)(hh4 >>> 16), (byte)(hh4 >>>8), (byte) hh4
	};
	return text;
}
```

```
/**
* HMAC truncates into 128 bits (16 bytes).
* <br />HMAC is specified in NIST Federal Information Processing Standards (FIPS) Publication 198-a.
* This variant uses either SHA-1 or MD5 as a hash function.
* SHA-1 is specified in NIST Federal Information Processing Standards (FIPS) Publication 180-2.
* MD-5 is a hash function, it is specified in RFC1321.
*
* @param text the byte array that needs to have a message authentication code
* @param key the key used to generate the MAC
* @param hashFunction a integer telling which hashing function to use
* @return a 128 bits MAC in a byte array
* @throws EapException If some internal calls fails
*/
public static byte[] hmac(byte[] text, byte[] key, int hashFunction) throws EapException{
	//digest is the truncated value to be returned (128 bits = 16 bytes)
	byte[] digest = new byte[16];
	//a temporary byte array used to store hashed values of 160 bits = 20 bytes
	byte[] tmp = null;
	if(hashFunction != SHA1 && hashFunction != MD5)
		throw new EapException("Wrong hash function has been chosen in Crypto HMAC!");

	//B Block size (in bytes) of the input to the Approved hash function.
	//H An Approved hash function.
	//Key (initially) Secret key shared between the originator and the intended receiver(s).
	//Key (during method) The key K after any necessary pre-processing to form a B byte key.
	//L Block size (in bytes) of the output of the Approved hash function.
	//t The number of bytes of MAC.
	//text The data on which the HMAC is calculated; text does not include the padded key.
	// The length of the text is n bits, where n < 2B - 8B.

	//Step 2: If the length of K > B: hash K to obtain an L byte string, then append (B-L)
	//zeros to create a B-byte string K0 (i.e., K0 = H(K) || 00...00). Go to step 4.
	if (key.length > BLOCK_SIZE){
		//the zeroes will be appended in step 3 here.
		if(hashFunction == SHA1)
			key = sha1(key);
		else if (hashFunction == MD5)
			key = md5(key);
	}
	//Step 3:If the length of K < B: append zeros to the end of K to create a B-byte string K0
	//(e.g., if K is 20 bytes in length and B = 64, then K will be appended with 44 zero bytes 0x00).
	if (key.length < BLOCK_SIZE){
		byte[] tmpKey = new byte[BLOCK_SIZE];
		System.arraycopy(key, 0, tmpKey, 0, key.length);
		key = tmpKey;
	}
	//Step 4: Exclusive-Or Key with ipad: Key XOR ipad (inner pad; the byte 0x36 repeated B times.)
	byte[] ipadArray = new byte[BLOCK_SIZE];
	for(int i=0; i<BLOCK_SIZE; i++){
		ipadArray[i] = (byte) (key[i]^IPAD);
	}
	//Step 5: Append the stream of data 'text' to the string resulting from step 4: (Key xor ipad) || text.
	byte[] tmpArray = new byte[BLOCK_SIZE+text.length];
	System.arraycopy(ipadArray, 0, tmpArray, 0, BLOCK_SIZE);
	System.arraycopy(text, 0, tmpArray, BLOCK_SIZE, text.length);
	//Step 6: Apply H to the stream generated in step 5: H((Key xor ipad) || text).
	if(hashFunction == SHA1)
		tmp = sha1(tmpArray);
	else if (hashFunction == MD5)
		tmp = md5(tmpArray);
```

```
        //Step 7: Exclusive-Or Key with opad: Key XOR opad( Outer pad; the byte 0x5c repeated B times)
        byte[] opadArray = new byte[BLOCK_SIZE];
        for(int i=0; i<BLOCK_SIZE; i++){
            opadArray[i] = (byte) (key[i]^OPAD);
        }
        //Step 8: Append the result from step 6 to step 7: (Key xor opad) || H((Key xor ipad) || text).
        tmpArray = new byte[tmp.length+BLOCK_SIZE];
        System.arraycopy(opadArray, 0, tmpArray, 0, BLOCK_SIZE);
        System.arraycopy(tmp, 0, tmpArray, BLOCK_SIZE, tmp.length);
        //Step 9: Apply H to the result from step 8: H((Key xor opad )|| H((Key or ipad) || text)).
        if(hashFunction == SHA1)
            tmp = sha1(tmpArray);
        else if (hashFunction == MD5)
            tmp = md5(tmpArray);
        //Step 10: Select the leftmost t (t= 128) bytes of the result of step 9 as the MAC.
        System.arraycopy(tmp, 0, digest, 0, 16);

        return digest; //return a 128 bits message digest
    }

    /**
    * EAP-SIM Pseudo-Random Number Function (PRNG), used in the key generation.
    * <br />Key derivation is based on the random number generation specified in NIST
    * Federal Information Processing Standards (FIPS) Publication 182-2.
    * The Pseudo-random number generator is specified in the change notice 1.
    *
    * @param mk the master key
    * @return 1280 bits byte array that can be partitioned into suitable-sized keys.
    * @throws EapException If some internal calls fails
    */
    public static byte[] eapSimPrng(byte[] mk) throws EapException{
        byte[] xkey = mk;
        ByteArray x = new ByteArray();
        byte[] m; int[] gResult;
        int hh0;int hh1;int hh2;int hh3;int hh4;

        for (int i = 0; i < 8; i++) {
            //Padds the xkey with zeros up to 512 bits
            m = new byte[BLOCK_SIZE];
            System.arraycopy(xkey, 0, m, 0, xkey.length);
            //Using the gFunction
            gResult = gFunction(m, h0, h1, h2, h3, h4);
            hh0 = gResult[0]; hh1 = gResult[1]; hh2 = gResult[2]; hh3 = gResult[3]; hh4 = gResult[4];
            byte[] w = new byte[] {
                (byte)(hh0 >>> 24), (byte)(hh0 >>> 16), (byte)(hh0>>> 8), (byte) hh0,
                (byte)(hh1 >>> 24), (byte)(hh1 >>> 16), (byte)(hh1>>> 8), (byte) hh1,
                (byte)(hh2 >>> 24), (byte)(hh2 >>> 16), (byte)(hh2>>> 8), (byte) hh2,
                (byte)(hh3 >>> 24), (byte)(hh3 >>> 16), (byte)(hh3>>> 8), (byte) hh3,
                (byte)(hh4 >>> 24), (byte)(hh4 >>> 16), (byte)(hh4>>> 8), (byte) hh4
            };
            x.write(w);

            //Finding the next xkey
            BigInteger xkeyBig = new BigInteger(xkey); BigInteger wBig = new BigInteger(w);
            BigInteger one = new BigInteger(""+1); BigInteger exp = new BigInteger(""+2);
            exp = exp.pow(160);
            xkeyBig = xkeyBig.add(one); xkeyBig = xkeyBig.add(wBig);
            xkeyBig = xkeyBig.mod(exp);
            xkey = xkeyBig.toByteArray();
```

```
            if(xkey.length > 20){
                byte[] tmpXKey = new byte[20];
                System.arraycopy(xkey, 1,tmpXKey, 0, 20);
                xkey = tmpXKey;
            }
        }
        return x.getArray();
}

/**
* The implementation of the G function derived from the SHA-1 algorithm.
* <br />The implementation follows the one specified in 186-2 Change Notice #1,
* when used in conjunction with eap-sim pad the input with only zeroes
*
* @param input the input to the GFunction, has to be 512 bits
* @param hh0 constant from the previous iteration or equal to h0 the first time
* @param hh1 constant from the previous iteration or equal to h1 the first time
* @param hh2 constant from the previous iteration or equal to h2 the first time
* @param hh3 constant from the previous iteration or equal to h3 the first time
* @param hh4 constant from the previous iteration or equal to h4 the first time
* @return 160 bits byte array
* @throws EapException if the input to the GFunction is not 512 bits
*/
private synchronized static int[] gFunction(byte[] input, int hh0, int hh1, int hh2, int hh3, int hh4)
                                                                    throws EapException {
        if(input.length != BLOCK_SIZE)
            throw new EapException("The input to the GFunction has to be512 bits");
        int a = hh0; int b = hh1; int c = hh2; int d = hh3; int e = hh4;
        int T = 0; int offset = 0; int[] W = new int[80];

        for (int i = 0; i < 16; i++) {
            W[i] = input[offset++] << 24 | (input[offset++] & 0xFF) << 16 |
            (input[offset++] & 0xFF) << 8 | (input[offset++] & 0xFF);
        }
        for (int i = 16; i < 80; i++) {
            T = W[i-3] ^ W[i-8] ^ W[i-14] ^ W[i-16];
            W[i] = T << 1 | T >>> 31;
        }
        // rounds 0-19
        for (int i = 0; i < 20; i++) {
            T = (a << 5 | a >>> 27) + ((b & c) | (~b & d)) + e + W[i] + 0x5A827999;
            e = d; d = c; c = b << 30 | b >>> 2; b = a; a = T;
        }
        // rounds 20-39
        for (int i = 20; i < 40; i++) {
            T = (a << 5 | a >>> 27) + (b ^ c ^ d) + e + W[i] + 0x6ED9EBA1;
            e = d; d = c; c = b << 30 | b >>> 2; b = a; a = T;
        }
        // rounds 40-59
        for (int i = 40; i < 60; i++) {
            T = (a << 5 | a >>> 27) + (b & c | b & d | c & d) + e + W[i] + 0x8F1BBCDC;
            e = d; d = c; c = b << 30 | b >>> 2; b = a; a = T;
        }
        // rounds 60-79
        for (int i = 60; i < 80; i++) {
            T = (a << 5 | a >>> 27) + (b ^ c ^ d) + e + W[i] + 0xCA62C1D6;
            e = d; d = c; c = b << 30 | b >>> 2; b = a; a = T;
        }
        return new int[] {hh0+a, hh1+b, hh2+c, hh3+d, hh4+e};
}
```

```
/**
* The method performs the shift operation needed in the MD-5function,
* according to the specification (RFC1321).
*
* @param input a 64 byte array
* @param hh0 constant from the previous iteration or equal to h0 the first time
* @param hh1 constant from the previous iteration or equal to h1 the first time
* @param hh2 constant from the previous iteration or equal to h2 the first time
* @param hh3 constant from the previous iteration or equal to h3 the first time
* @return an int array of 128 bits
*/
private synchronized static int[] md5Function(byte[] input, int hh0, int hh1, int hh2, int hh3) {
      int offset = 0; int[]X = new int[16];
      for(int i=0; i<16; i++){
            X[i]=(input[offset++] & 0xFF) | (input[offset++] & 0xFF) << 8 |
                  (input[offset++] & 0xFF) << 16 | input[offset++] << 24;
      }
      int a = hh0; int b = hh1; int c = hh2; int d = hh3;
      // round 1
      a += ((b & c) | (~b & d)) + X[0] + 0xD76AA478;  a = b + (a << 7 | a >>> -7);
      d += ((a & b) | (~a & c)) + X[1] + 0xE8C7B756;  d = a + (d << 12 | d >>> -12);
      c += ((d & a) | (~d & b)) + X[2] + 0x242070DB;  c = d + (c << 17 | c >>> -17);
      b += ((c & d) | (~c & a)) + X[3] + 0xC1BDCEEE;  b = c + (b << 22 | b >>> -22);

      a += ((b & c) | (~b & d)) + X[4] + 0xF57C0FAF;  a = b + (a << 7 | a >>> -7);
      d += ((a & b) | (~a & c)) + X[5] + 0x4787C62A;  d = a + (d << 12 | d >>> -12);
      c += ((d & a) | (~d & b)) + X[6] + 0xA8304613;  c = d + (c << 17 | c >>> -17);
      b += ((c & d) | (~c & a)) + X[7] + 0xFD469501;  b = c + (b << 22 | b >>> -22);

      a += ((b & c) | (~b & d)) + X[8] + 0x698098D8;  a = b + (a << 7 | a >>> -7);
      d += ((a & b) | (~a & c)) + X[9] + 0x8B44F7AF;  d = a + (d << 12 | d >>> -12);
      c += ((d & a) | (~d & b)) + X[10] + 0xFFFF5BB1; c = d + (c << 17 | c >>> -17);
      b += ((c & d) | (~c & a)) + X[11] + 0x895CD7BE; b = c + (b << 22 | b >>> -22);

      a += ((b & c) | (~b & d)) + X[12] + 0x6B901122; a = b + (a << 7 | a >>> -7);
      d += ((a & b) | (~a & c)) + X[13] + 0xFD987193; d = a + (d << 12 | d >>> -12);
      c += ((d & a) | (~d & b)) + X[14] + 0xA679438E; c = d + (c << 17 | c >>> -17);
      b += ((c & d) | (~c & a)) + X[15] + 0x49B40821; b = c + (b << 22 | b >>> -22);
      // round 2
      a += ((b & d) | (c & ~d)) + X[1] + 0xF61E2562;  a = b + (a << 5 | a >>> -5);
      d += ((a & c) | (b & ~c)) + X[6] + 0xC040B340;  d = a + (d << 9 | d >>> -9);
      c += ((d & b) | (a & ~b)) + X[11] + 0x265E5A51; c = d + (c << 14 | c >>> -14);
      b += ((c & a) | (d & ~a)) + X[0] + 0xE9B6C7AA;  b = c + (b << 20 | b >>> -20);

      a += ((b & d) | (c & ~d)) + X[5] + 0xD62F105D;  a = b + (a << 5 | a >>> -5);
      d += ((a & c) | (b & ~c)) + X[10] + 0x02441453; d = a + (d << 9 | d >>> -9);
      c += ((d & b) | (a & ~b)) + X[15] + 0xD8A1E681; c = d + (c << 14 | c >>> -14);
      b += ((c & a) | (d & ~a)) + X[4] + 0xE7D3FBC8;  b = c + (b << 20 | b >>> -20);

      a += ((b & d) | (c & ~d)) + X[9] + 0x21E1CDE6;  a = b + (a << 5 | a >>> -5);
      d += ((a & c) | (b & ~c)) + X[14] + 0xC33707D6; d = a + (d << 9 | d >>> -9);
      c += ((d & b) | (a & ~b)) + X[3] + 0xF4D50D87;  c = d + (c << 14 | c >>> -14);
      b += ((c & a) | (d & ~a)) + X[8] + 0x455A14ED;  b = c + (b << 20 | b >>> -20);

      a += ((b & d) | (c & ~d)) + X[13] + 0xA9E3E905; a = b + (a << 5 | a >>> -5);
      d += ((a & c) | (b & ~c)) + X[2] + 0xFCEFA3F8;  d = a + (d << 9 | d >>> -9);
      c += ((d & b) | (a & ~b)) + X[7] + 0x676F02D9;  c = d + (c << 14 | c >>> -14);
      b += ((c & a) | (d & ~a)) + X[12] + 0x8D2A4C8A; b = c + (b << 20 | b >>> -20);
```

```
        // round 3
        a += (b ^ c ^ d) + X[5] + 0xFFFA3942;  a = b + (a << 4 | a >>> -4);
        d += (a ^ b ^ c) + X[8] + 0x8771F681;  d = a + (d << 11 | d >>> -11);
        c += (d ^ a ^ b) + X[11] + 0x6D9D6122; c = d + (c << 16 | c >>> -16);
        b += (c ^ d ^ a) + X[14] + 0xFDE5380C; b = c + (b << 23 | b >>> -23);

        a += (b ^ c ^ d) + X[1] + 0xA4BEEA44;  a = b + (a << 4 | a >>> -4);
        d += (a ^ b ^ c) + X[4] + 0x4BDECFA9;  d = a + (d << 11 | d >>> -11);
        c += (d ^ a ^ b) + X[7] + 0xF6BB4B60;  c = d + (c << 16 | c >>> -16);
        b += (c ^ d ^ a) + X[10] + 0xBEBFBC70; b = c + (b << 23 | b >>> -23);

        a += (b ^ c ^ d) + X[13] + 0x289B7EC6; a = b + (a << 4 | a >>> -4);
        d += (a ^ b ^ c) + X[0] + 0xEAA127FA;  d = a + (d << 11 | d >>> -11);
        c += (d ^ a ^ b) + X[3] + 0xD4EF3085;  c = d + (c << 16 | c >>> -16);
        b += (c ^ d ^ a) + X[6] + 0x04881D05;  b = c + (b << 23 | b >>> -23);

        a += (b ^ c ^ d) + X[9] + 0xD9D4D039;  a = b + (a << 4 | a >>> -4);
        d += (a ^ b ^ c) + X[12] + 0xE6DB99E5; d = a + (d << 11 | d >>> -11);
        c += (d ^ a ^ b) + X[15] + 0x1FA27CF8; c = d + (c << 16 | c >>> -16);
        b += (c ^ d ^ a) + X[2] + 0xC4AC5665;  b = c + (b << 23 | b >>> -23);
        // round 4
        a += (c ^ (b | ~d)) + X[0] + 0xF4292244;  a = b + (a << 6 | a >>> -6);
        d += (b ^ (a | ~c)) + X[7] + 0x432AFF97;  d = a + (d << 10 | d >>> -10);
        c += (a ^ (d | ~b)) + X[14] + 0xAB9423A7; c = d + (c << 15 | c >>> -15);
        b += (d ^ (c | ~a)) + X[5] + 0xFC93A039;  b = c + (b << 21 | b >>> -21);

        a += (c ^ (b | ~d)) + X[12] + 0x655B59C3; a = b + (a << 6 | a >>> -6);
        d += (b ^ (a | ~c)) + X[3] + 0x8F0CCC92;  d = a + (d << 10 | d >>> -10);
        c += (a ^ (d | ~b)) + X[10] + 0xFFEFF47D; c = d + (c << 15 | c >>> -15);
        b += (d ^ (c | ~a)) + X[1] + 0x85845dd1;  b = c + (b << 21 | b >>> -21);

        a += (c ^ (b | ~d)) + X[8] + 0x6FA87E4F;  a = b + (a << 6 | a >>> -6);
        d += (b ^ (a | ~c)) + X[15] + 0xFE2CE6E0; d = a + (d << 10 | d >>> -10);
        c += (a ^ (d | ~b)) + X[6] + 0xA3014314;  c = d + (c << 15 | c >>> -15);
        b += (d ^ (c | ~a)) + X[13] + 0x4E0811A1; b = c + (b << 21 | b >>> -21);

        a += (c ^ (b | ~d)) + X[4] + 0xF7537E82;  a = b + (a << 6 | a >>> -6);
        d += (b ^ (a | ~c)) + X[11] + 0xBD3AF235; d = a + (d << 10 | d >>> -10);
        c += (a ^ (d | ~b)) + X[2] + 0x2AD7D2BB;  c = d + (c << 15 | c >>> -15);
        b += (d ^ (c | ~a)) + X[9] + 0xEB86D391;  b = c + (b << 21 | b >>> -21);
        return new int[] {hh0+a, hh1+b, hh2+c, hh3+d};
}//end of method}//end of class
```

Appendix D Mobile Phones Supporting SAP

Mobile phones supporting Bluetooth SIM Access Profile (SAP)

Nokia	
N71	6021
N80	6103
N91	6111
N92	6125
E60	6131
E61	6230
E70	6230i
3250	6233
7370	6270
8800	6280
9300	6810
9300i	6820
	6822

BenQ-Siemens
SK65 Burlwood
SL75 Escada
SXG75
SP65
SL75
SK65
S65
S68
S75
M75
CX75

Appendix E Configurations

E.1 JPCSC Mac OS X Port

This Appendix will describe the small changes that are needed to port the JPCSC library to Mac OS X.

Makefile changes:
Changes in the Makefile in the directory *<jpcsc-directory>src/jpcsc/*:
Under:
else
#a unix target
Change to:
DSTLIB=libjpcsc.jnilib
Change to:
LINK=${CC} –dynamiclib –o ${ARCHDIR}/${DSTLIB} ${COBJS} ${LDFLAGS}

Source code changes:
Changes in jpcsc.h in the directory *<jpcsc-directory>src/jpcsc/*:
Change to:
#include <PCSC/winscard.h>
#include <PCSC/wintypes.h>

E.2 ChilliSpot

This Appendix will show the configuration file used with the GasSpot service. The configuration file is normally located at the location */etc/chilli.conf*.

 # Radius parameters
radiusserver1 <IP address of RADIUS server, (use 127.0.0.1 for localhost)>
radiusserver2 <IP address of RADIUS server, (use 127.0.0.1 for localhost)>
radiussecret <Shared RADIUS secret>

 # Radius proxy parameters
proxylisten <Internet IP address of local host>
proxyport <RADIUS proxy port, used by Authenticator>
proxyclient <IP address of the Authenticator>
proxysecret <Shared secret between Chilli and the Authenticator>

 # DHCP Parameters
dhcpif <Internal network card, normally eth0 or eth1>

 # Universal access method (UAM) parameters
uamserver https://<Internet IP address of the Chilli Web Server>
uamallowed 192.168.182.1,<Internet IP address of local host>,<IP address of Authenticator>,<IP address of RADIUS server>

Appendix F Captured Network Data from Prototype

This Appendix shows captured network data from a normal run of the Prototype. The data is captured using Ethereal; only the fields of interest are included.

F.1 Between Supplicant and Authenticator

Network data sent between Supplicant (193.156.21.121) and Authenticator (jaguar.stud.ntnu.no):

No. Time Source Destination Protocol Info
1 0.012207 193.156.21.121 jaguar.stud.ntnu.no TCP [TCP segment of a reassembled PDU]

Frame 4 (61 bytes on wire, 61 bytes captured)
Ethernet II, Src: Aopen_57:45:de (00:01:80:57:45:de), Dst: Cisco_bb:78:00 (00:30:96:bb:78:00)
Internet Protocol, Src: 193.156.21.121 (193.156.21.121), Dst: 129.241.56.181 (129.241.56.181)
Transmission Control Protocol, Src Port: 1231 (1231), Dst Port: 8080 (8080), Seq: 1, Ack: 1, Len: 7
Extensible Authentication Protocol
Code: Request (1)
Id: 0
Length: 7
Type: Experimental usage [RFC3748] (255)
subtype: 153 (Get method list)
Min sec level (1 byte): 1

No. Time Source Destination Protocol Info
2 0.108767 jaguar.stud.ntnu.no 193.156.21.121 TCP [TCP segment of a reassembled PDU]

Frame 6 (61 bytes on wire, 61 bytes captured)
Ethernet II, Src: Cisco_bb:78:00 (00:30:96:bb:78:00), Dst: Aopen_57:45:de (00:01:80:57:45:de)
Internet Protocol, Src: 129.241.56.181 (129.241.56.181), Dst: 193.156.21.121 (193.156.21.121)
Transmission Control Protocol, Src Port: 8080 (8080), Dst Port: 1231 (1231), Seq: 1, Ack: 8, Len: 7
Extensible Authentication Protocol
Code: Response (2)
Id: 0
Length: 7
Type: Experimental usage [RFC3748] (255)
subtype: 153 (Get method list)
Method list (1 byte): 18 (EAP-SIM Nokia IP smart card authentication)

No. Time Source Destination Protocol Info
3 0.115859 193.156.21.121 jaguar.stud.ntnu.no TCP [TCP segment of a reassembled PDU]

Frame 7 (61 bytes on wire, 61 bytes captured)
Ethernet II, Src: Aopen_57:45:de (00:01:80:57:45:de), Dst: Cisco_bb:78:00 (00:30:96:bb:78:00)
Internet Protocol, Src: 193.156.21.121 (193.156.21.121), Dst: 129.241.56.181 (129.241.56.181)
Transmission Control Protocol, Src Port: 1231 (1231), Dst Port: 8080 (8080), Seq: 8, Ack: 8, Len: 7
Extensible Authentication Protocol
Code: Request (1)
Id: 0
Length: 7
Type: Experimental usage [RFC3748] (255)
subtype: 154 (Chosen Method)
Chosen Method (1 byte): 18 (EAP-SIM Nokia IP smart card authentication)

No. Time Source Destination Protocol Info
4 0.125281 jaguar.stud.ntnu.no 193.156.21.121 TCP [TCP segment of a reassembled PDU]

Frame 9 (60 bytes on wire, 60 bytes captured)
Ethernet II, Src: Cisco_bb:78:00 (00:30:96:bb:78:00), Dst: Aopen_57:45:de (00:01:80:57:45:de)
Internet Protocol, Src: 129.241.56.181 (129.241.56.181), Dst: 193.156.21.121 (193.156.21.121)
Transmission Control Protocol, Src Port: 8080 (8080), Dst Port: 1231 (1231), Seq: 8, Ack: 15, Len: 5
Extensible Authentication Protocol
Code: Request (1)
Id: 0
Length: 5
Type: Identity [RFC3748] (1)

No. Time Source Destination Protocol Info
5 0.149633 193.156.21.121 jaguar.stud.ntnu.no TCP [TCP segment of a reassembled PDU]

Frame 10 (97 bytes on wire, 97 bytes captured)
Ethernet II, Src: Aopen_57:45:de (00:01:80:57:45:de), Dst: Cisco_bb:78:00 (00:30:96:bb:78:00)
Internet Protocol, Src: 193.156.21.121 (193.156.21.121), Dst: 129.241.56.181 (129.241.56.181)
Transmission Control Protocol, Src Port: 1231 (1231), Dst Port: 8080 (8080), Seq: 15, Ack: 13, Len: 43
Extensible Authentication Protocol
Code: Response (2)
Id: 0
Length: 43
Type: Identity [RFC3748] (1)
Identity (38 bytes): 1208015609919851@read.triplet.file.com

No. Time Source Destination Protocol Info
6 0.239684 jaguar.stud.ntnu.no 193.156.21.121 TCP [TCP segment of a reassembled PDU]

Frame 12 (74 bytes on wire, 74 bytes captured)
Ethernet II, Src: Cisco_bb:78:00 (00:30:96:bb:78:00), Dst: Aopen_57:45:de (00:01:80:57:45:de)
Internet Protocol, Src: 129.241.56.181 (129.241.56.181), Dst: 193.156.21.121 (193.156.21.121)
Transmission Control Protocol, Src Port: 8080 (8080), Dst Port: 1231 (1231), Seq: 13, Ack: 58, Len: 20
Extensible Authentication Protocol
Code: Request (1)
Id: 148
Length: 20
Type: EAP-SIM Nokia IP smart card authentication [Haverinen] (18)
subtype: 10 (Start)
Reserved: 0
Attribute: AT_VERSION_LIST
Attribute: AT_FULLAUTH_ID_REQ

No. Time Source Destination Protocol Info
7 0.245754 193.156.21.121 jaguar.stud.ntnu.no TCP [TCP segment of a reassembled PDU]

Frame 13 (130 bytes on wire, 130 bytes captured)
Ethernet II, Src: Aopen_57:45:de (00:01:80:57:45:de), Dst: Cisco_bb:78:00 (00:30:96:bb:78:00)
Internet Protocol, Src: 193.156.21.121 (193.156.21.121), Dst: 129.241.56.181 (129.241.56.181)
Transmission Control Protocol, Src Port: 1231 (1231), Dst Port: 8080 (8080), Seq: 58, Ack: 33, Len: 76
Extensible Authentication Protocol
Code: Response (2)
Id: 148
Length: 76
Type: EAP-SIM Nokia IP smart card authentication [Haverinen] (18)
subtype: 10 (Start)
Reserved: 0
Attribute: AT_NONCE_MT
Attribute: AT_IDENTITY
Attribute: AT_SELECTED_VERSION

No. Time Source Destination Protocol Info
8 0.279265 jaguar.stud.ntnu.no 193.156.21.121 TCP [TCP segment of a reassembled PDU]

Frame 15 (134 bytes on wire, 134 bytes captured)
Ethernet II, Src: Cisco_bb:78:00 (00:30:96:bb:78:00), Dst: Aopen_57:45:de (00:01:80:57:45:de)
Internet Protocol, Src: 129.241.56.181 (129.241.56.181), Dst: 193.156.21.121 (193.156.21.121)
Transmission Control Protocol, Src Port: 8080 (8080), Dst Port: 1231 (1231), Seq: 33, Ack: 134, Len: 80
Extensible Authentication Protocol
Code: Request (1)
Id: 149
Length: 80
Type: EAP-SIM Nokia IP smart card authentication [Haverinen] (18)
subtype: 11 (Challenge)
Reserved: 0
Attribute: AT_RAND
Attribute: AT_MAC

No. Time Source Destination Protocol Info
9 0.465761 193.156.21.121 jaguar.stud.ntnu.no TCP [TCP segment of a reassembled PDU]

Frame 17 (82 bytes on wire, 82 bytes captured)
Ethernet II, Src: Aopen_57:45:de (00:01:80:57:45:de), Dst: Cisco_bb:78:00 (00:30:96:bb:78:00)
Internet Protocol, Src: 193.156.21.121 (193.156.21.121), Dst: 129.241.56.181 (129.241.56.181)
Transmission Control Protocol, Src Port: 1231 (1231), Dst Port: 8080 (8080), Seq: 134, Ack: 113, Len: 28
Extensible Authentication Protocol
Code: Response (2)
Id: 149
Length: 28
Type: EAP-SIM Nokia IP smart card authentication [Haverinen] (18)
subtype: 11 (Challenge)
Reserved: 0
Attribute: AT_MAC

No.	Time	Source	Destination	Protocol	Info
10	**0.492297**	**jaguar.stud.ntnu.no**	**193.156.21.121**	**TCP**	**[TCP segment of a reassembled PDU]**

Frame 18 (60 bytes on wire, 60 bytes captured)
Ethernet II, Src: Cisco_bb:78:00 (00:30:96:bb:78:00), Dst: Aopen_57:45:de (00:01:80:57:45:de)
Internet Protocol, Src: 129.241.56.181 (129.241.56.181), Dst: 193.156.21.121 (193.156.21.121)
Transmission Control Protocol, Src Port: 8080 (8080), Dst Port: 1231 (1231), Seq: 113, Ack: 162, Len: 4
Extensible Authentication Protocol
Code: Success (3)
Id: 150
Length: 4

F.2 Between Authenticator and RADIUS server

Network data sent between Authenticator (jaguar.stud.ntnu.no) and RADIUS server (xmmap.nta.no):

No. Time Source Destination Protocol Info
1 0.000000 jaguar.stud.ntnu.no xmmap.nta.no RADIUS Access-Request(1) (id=0, l=135)

Frame 1 (177 bytes on wire, 177 bytes captured)
Ethernet II, Src: Cisco_f7:30:c0 (00:13:7f:f7:30:c0), Dst: AsustekC_18:53:e4 (00:0e:a6:18:53:e4)
Internet Protocol, Src: 129.241.56.181 (129.241.56.181), Dst: 193.156.19.170 (193.156.19.170)
User Datagram Protocol, Src Port: 33478 (33478), Dst Port: radius (1812)
Radius Protocol
Code: Access-Request (1)
Packet identifier: 0x0 (0)
Length: 135
Authenticator: FFEEDE8DB7F905485FBA6487AC9A561D
Attribute Value Pairs
AVP: l=40 t=User-Name(1): 1208015609919851@read.triplet.file.com
AVP: l=6 t=Service-Type(6): Framed-User(2)
AVP: l=45 t=EAP-Message(79) Last Segment[1]
EAP fragment
Extensible Authentication Protocol
Code: Response (2)
Id: 0
Length: 43
Type: Identity [RFC3748] (1)
Identity (38 bytes): 1208015609919851@read.triplet.file.com
AVP: l=6 t=NAS-IP-Address(4): 129.241.56.181
AVP: l=18 t=Message-Authenticator(80): F0A25718C1FA5B2D7CCCB74E3D7B3865

No. Time Source Destination Protocol Info
2 0.000655 xmmap.nta.no jaguar.stud.ntnu.no RADIUS Access-challenge(11) (id=0, l=78)

Frame 2 (120 bytes on wire, 120 bytes captured)
Ethernet II, Src: AsustekC_18:53:e4 (00:0e:a6:18:53:e4), Dst: Cisco_f7:30:c0 (00:13:7f:f7:30:c0)
Internet Protocol, Src: 193.156.19.170 (193.156.19.170), Dst: 129.241.56.181 (129.241.56.181)
User Datagram Protocol, Src Port: radius (1812), Dst Port: 33478 (33478)
Radius Protocol
Code: Access-challenge (11)
Packet identifier: 0x0 (0)
Length: 78
Authenticator: B10EAFBEBDACF42315F32332197D8F67
Attribute Value Pairs
AVP: l=22 t=EAP-Message(79) Last Segment[1]
EAP fragment
Extensible Authentication Protocol
Code: Request (1)
Id: 148
Length: 20
Type: EAP-SIM Nokia IP smart card authentication [Haverinen] (18)
subtype: 10 (Start)
Reserved: 0
Attribute: AT_VERSION_LIST
Attribute: AT_FULLAUTH_ID_REQ
AVP: l=18 t=Message-Authenticator(80): F6CA0590CA3DAFE2174E8DB812D89BF7
AVP: l=18 t=State(24): 8C9F092E15B266816CAEB33B8577F64A

No. Time Source Destination Protocol Info
3 0.035013 jaguar.stud.ntnu.no xmmap.nta.no RADIUS Access-Request(1) (id=1, l=186)

Frame 3 (228 bytes on wire, 228 bytes captured)
Ethernet II, Src: Cisco_f7:30:c0 (00:13:7f:f7:30:c0), Dst: AsustekC_18:53:e4 (00:0e:a6:18:53:e4)
Internet Protocol, Src: 129.241.56.181 (129.241.56.181), Dst: 193.156.19.170 (193.156.19.170)
User Datagram Protocol, Src Port: 33478 (33478), Dst Port: radius (1812)
Radius Protocol
Code: Access-Request (1)
Packet identifier: 0x1 (1)
Length: 186
Authenticator: 53371E00C6C829ABD8FCA845F77739D8
Attribute Value Pairs
AVP: l=40 t=User-Name(1): 1208015609919851@read.triplet.file.com
AVP: l=6 t=Service-Type(6): Framed-User(2)
AVP: l=78 t=EAP-Message(79) Last Segment[1]
EAP fragment
Extensible Authentication Protocol
Code: Response (2)
Id: 148
Length: 76
Type: EAP-SIM Nokia IP smart card authentication [Haverinen] (18)
subtype: 10 (Start)
Reserved: 0
Attribute: AT_NONCE_MT
Attribute: AT_IDENTITY
Attribute: AT_SELECTED_VERSION
AVP: l=6 t=NAS-IP-Address(4): 129.241.56.181
AVP: l=18 t=State(24): 8C9F092E15B266816CAEB33B8577F64A
AVP: l=18 t=Message-Authenticator(80): C505F626DB91B08E018777A0D69A203A

No. Time Source Destination Protocol Info
4 0.035474 xmmap.nta.no jaguar.stud.ntnu.no RADIUS Access-challenge(11) (id=1, l=138)

Frame 4 (180 bytes on wire, 180 bytes captured)
Ethernet II, Src: AsustekC_18:53:e4 (00:0e:a6:18:53:e4), Dst: Cisco_f7:30:c0 (00:13:7f:f7:30:c0)
Internet Protocol, Src: 193.156.19.170 (193.156.19.170), Dst: 129.241.56.181 (129.241.56.181)
User Datagram Protocol, Src Port: radius (1812), Dst Port: 33478 (33478)
Radius Protocol
Code: Access-challenge (11)
Packet identifier: 0x1 (1)
Length: 138
Authenticator: 2EE7F01435FB7A24BC31DB4C54B7421E
Attribute Value Pairs
AVP: l=82 t=EAP-Message(79) Last Segment[1]
EAP fragment
Extensible Authentication Protocol
Code: Request (1)
Id: 149
Length: 80
Type: EAP-SIM Nokia IP smart card authentication [Haverinen] (18)
subtype: 11 (Challenge)
Reserved: 0
Attribute: AT_RAND
Attribute: AT_MAC
AVP: l=18 t=Message-Authenticator(80): F91AC871FFE8E5A0B85341550620027E
AVP: l=18 t=State(24): 4BE09F6D64416A3A04EDB25CC1F56D09

No. Time Source Destination Protocol Info
5 0.237251 jaguar.stud.ntnu.no xmmap.nta.no RADIUS Access-Request(1) (id=2, l=138)

Frame 5 (180 bytes on wire, 180 bytes captured)
Ethernet II, Src: Cisco_f7:30:c0 (00:13:7f:f7:30:c0), Dst: AsustekC_18:53:e4 (00:0e:a6:18:53:e4)
Internet Protocol, Src: 129.241.56.181 (129.241.56.181), Dst: 193.156.19.170 (193.156.19.170)
User Datagram Protocol, Src Port: 33478 (33478), Dst Port: radius (1812)
Radius Protocol
Code: Access-Request (1)
Packet identifier: 0x2 (2)
Length: 138
Authenticator: DA5446ECDECAD25DD7EDFCB5F160F189
Attribute Value Pairs
AVP: l=40 t=User-Name(1): 1208015609919851@read.triplet.file.com
AVP: l=6 t=Service-Type(6): Framed-User(2)
AVP: l=30 t=EAP-Message(79) Last Segment[1]
EAP fragment
Extensible Authentication Protocol
Code: Response (2)
Id: 149
Length: 28
Type: EAP-SIM Nokia IP smart card authentication [Haverinen] (18)
subtype: 11 (Challenge)
Reserved: 0
Attribute: AT_MAC
AVP: l=6 t=NAS-IP-Address(4): 129.241.56.181
AVP: l=18 t=State(24): 4BE09F6D64416A3A04EDB25CC1F56D09
AVP: l=18 t=Message-Authenticator(80): 1D236E69ECC99A3C0C9BB2174E0678AC

No. Time Source Destination Protocol Info
6 0.241502 xmmap.nta.no jaguar.stud.ntnu.no RADIUS Access-Accept(2) (id=2, l=200)

Frame 6 (242 bytes on wire, 242 bytes captured)
Ethernet II, Src: AsustekC_18:53:e4 (00:0e:a6:18:53:e4), Dst: Cisco_f7:30:c0 (00:13:7f:f7:30:c0)
Internet Protocol, Src: 193.156.19.170 (193.156.19.170), Dst: 129.241.56.181 (129.241.56.181)
User Datagram Protocol, Src Port: radius (1812), Dst Port: 33478 (33478)
Radius Protocol
Code: Access-Accept (2)
Packet identifier: 0x2 (2)
Length: 200
Authenticator: 5C9B17100195DD6F7DB248E853AC99E5
Attribute Value Pairs
AVP: l=58 t=Vendor-Specific(26) v=Microsoft(311)
AVP: l=58 t=Vendor-Specific(26) v=Microsoft(311)
AVP: l=6 t=EAP-Message(79) Last Segment[1]
EAP fragment
Extensible Authentication Protocol
Code: Success (3)
Id: 150
Length: 4
AVP: l=18 t=Message-Authenticator(80): 598B68BD6223B0C22D23A28A24BC3EB5
AVP: l=40 t=User-Name(1): 1208015609919851@read.triplet.file.com

Appendix G Enclosed CD

There is a index.html-file on the CD. Use this file to be guided through the CD. The following directories can be found on the enclosed CD:

Generic SIM Authentication System (GAS)

Client (Stand-alone and Applet)
To start the stand-alone client you can either double-click the GASClient.jar file, or you can execute the command at the command line*: #java -jar GASClient.jar*

Authenticator
To start the authenticator you can either double-click the GASAuthenticator.jar file, or you can execute the command at the command line: *#java -jar GASAuthenticator.jar*
To start the Authenticator in daemon mode, you can use the unix shell scripts: *startDaemon.sh* and *stopDaemon.sh*

TripletGenerator
To start the TripletGenerator you can either double-click the GASAuthenticator.jar file, or you can execute the command at the command line: *#java -jar TripletGenerator.jar*
The TripletGenerator will store the triplets in a text file called simtriplets in the same directory.

JavaDoc

This directory includes two versions of JavaDoc:

- GAS API, Javadoc for members with visibility public
- Javadoc for members with visibility private

Master Thesis – this thesis in pdf format

Source Code

GAS
The source code of the GAS Prototype

TripletGenerator
The source code of the TripletGenerator

ChilliProvider
The Web Pages used in the Chilli Provider Demo

jspDemo
The source code of the jsp Demo

phpDemo
The source code of the php Demo (myService)

Tools

This directory includes the different tools used in the realization of the Prototype:

- FreeRadius
- NavisRadius
- Chilli
- AvetanaBluetooth
- Jpcsc
- Apache Ant
- Apache Tomcat
- Apache HTTP Server
- Ethereal
- Java 2 Platform Standard Edition 5.0 (External link)

Appendix H Accepted Paper to ICISP 2006

The paper "A Generic Authentication System based on SIM" was accepted for the IARIA (International Academy, Research, and Industry Association) conference International Conference on Internet Surveillance and Protection (ICISP'06).

The email shown below is from the Manager of ICISP'06 that reports that the paper has been accepted, and gives the reviewer's comments. The final version of the paper is shown at the end of this Appendix.

```
From: "Manager of ICISP'06" <CGI.script.-.do.not.reply@iariapapers.org>
Date: 7. mai 2006 17.17.18 GMT+02:00
To: larslund@stud.ntnu.no
Subject: Your ICISP'06 paper 26
Reply-To: Manager.of.ICISP'06@iariapapers.org

Dear Lars Lunde,

On behalf of the Program Committee, we are happy to inform you that your
paper 26 ("A Generic Authentication System based on SIM") has been accepted
for publication and presentation at the co-located events ICISP 2006 or
ICDT 2006.

The acceptance of your paper is made with the understanding that each
accepted paper will be registered and at least one author will attend the
conference to present the paper (preferably with PowerPoint slides).

First, please consider the reviewers' comments or guidelines when editing
your final version.

The Proceedings are published by the IEEE Computer Society Press. Please
read carefully the "Manuscript Preparation Instructions" on the conference
web site when submitting the final version. All the registration related
issues must be addressed to oana@vicov.com.

Note that a paper will be published on the IEEE Xplore and Conference
Proceedings only after the paper is registered, i.e., the registration form
is sent in the due time and successfully processed. Please fax the
registration for before uploading the paper. The hotel booking can be sent
later, using the same registration form. However, as there is a hot season,
please book the conference hotel as soon as possible; on May 20 our
contractors must release the pre-booked extra rooms.

The registration form is available from the conference web site: see
"Registration form"

A registered attendee for an event, has access to all sessions of the
collocated event, as a bonus. However, each paper must be separately
registered.

The deadline for the submission of registration forms to the organizers is
20 May 2006. Otherwise, the paper is considered withdrawn and it will
neither appear in the proceedings of the conference nor published on the
IEEE site. You can still update the originally and timely submitted paper
on the IEEE site later on, as shown in "Manuscript Preparation
Instructions"

Thanking once again for your active participation, we are looking forward
for your cooperation to successfully finalize the event.
```

Again, please accurately consider the review comments when finalizing your camera ready version. The chairs will check for the conformance with the reviewers' comments.

Please fill the registration form as soon as possible. Flight information can be sent later, using the same registration form.

Sincerely yours, Events's Chairs ----- Reviews ------

* How well does the paper fit into the scope of the conference ? 3 (3=acceptable)

* How new are the contributions of the paper? 3 (3=acceptable)

* Is the paper technically sound? 3 (3=acceptable)

* How well the paper is presented? 2 (2=poor)

* Overall Recommendation 3 (3=acceptable)

* Please assess your competence in the subject matter of the paper: 2 (2=Medium)

* Comments to Authors I would have liked to see at least some basic analysis of the security properties of this authentication system, including which threats are considered and how they are addressed by the system. The abstract talks about certain applications that "require strong authentication", from which I conclude that your system claims to provide that. While strong authentication sounds indeed useful, I am very skeptical about the overall security of a system that just forces the use of a strong authentication mechanism somehow/somewhere; it is important to see the context in which strong authentication is used, how easy it would be to make people perform strong authentication for a purpose other than what they think, etc.

The organization of the paper increases the difficulty for me to understand these security implications, because only in section 6 there are some more concrete hints about how the system will be used. I could not grasp that from the "use cases" in section 3. It would have been helpful for me to see a concrete description of how this system could be used up front, including how the user interacts with the Applet as part of an application access (login) procedure, and how she interacts with her (or her neighbor's :-) SIM/Smartcard reader. Some of this is described in section 6, but too late and too hard to put in context.

At first I found "Authenticator" to be a strange name for the component that receives requests from the supplicant - it doesn't do any authentication itself (that is done at the "Authentication Server", which sounds like just a fancier name for an "authenticator"), and this impression is amplified when you write: "The main functionality of the Authenticator is to act as a translator from EAP to RADIUS and back." But in fact the "Authenticator" seems to be more than just a relay/protocol converter: it has to locate a suitable Authentication Server, and somehow it synthesizes a new identity (useful for the Supplicant, or for the authorization question at hand) from the identity assertion from that Authentication Server, combined with the "Authenticator"'s interpretation of identities at that server (defined by some form of trust).
----------*----------

A Generic Authentication System based on SIM

Audun Wangensteen - *Norwegian University of Science and Technology - audunw@stud.ntnu.no*
Lars Lunde - *Norwegian University of Science and Technology - larslund@stud.ntnu.no*
Ivar Jørstad - *Norwegian University of Science and technology - ivar@ongx.org*
Do van Thanh - Telenor R&D - thanh-van.do@telenor.com

Abstract

Today the Internet is mostly used for services that require low or none security. The commercial and governmental applications have started to emerge but met problems since they require strong authentication, which is both difficult and costly to realize. The SIM card used in mobile phones is a tamper resistant device that contains strong authentication mechanisms, and if Internet services could use the strong authentications of the SIM card it would be very convenient and cost-efficient.

This paper presents an analysis and a design of a generic authentication system based on SIM. The proposed system supports different types of authentication methods to target the various security requirements of the services; OTP authentication, GSM/UMTS authentication, and PKI authentication. Another feature of the proposed system is that it can easily be extended as the authentication mechanism used in a Single Sign-On system.

1. Introduction

The emergence and growth of Internet usage leads to the need for security and enhanced privacy. With an increasing number of service providers requiring authentication of different strength, it becomes difficult for the user to manage all these identities. The number of user identities should therefore be kept at a minimum to accomplish user simplicity. Throughout this paper, different protocols, interfaces, and security problems during an end-to-end authentication will be investigated to find a solution reducing the number of user identities.

The investigation is based on Subscriber Identity Module (SIM) usage and existing technologies. The idea is to combine existing technologies to meet the security requirements (i.e. support strong authentication) of today and at the same time make the system user-friendly. Strong authentication is a process controlling the authenticity of the users identity on the basis of at least two of the three following factors:

- "Something You Know", e.g. user name and password.
- "Something You Have", e.g. a token device.
- "Something You Are", e.g. fingerprints

This paper presents an analysis and a design of a generic authentication system based on SIM, together with the gained experiences from making a prototype of the system. The paper starts by summarizing the state-of-the-art solutions for strong authentication and their limitations in related work. A description of the developed prototype, the Generic SIM Authentication System (GAS), is then presented to give an understanding of the system. Section 4 shows the overall architecture while components and their implementation are explained in section 5. Planned extensions are explained in future work.

The user-friendliness of a generic SIM authentication system is dependent on the possibility of using the mobile phone in the authentication process. The retrieval of SIM credentials and to use these in a secure system has been a big challenge. The main features of the different SIM connection methods will be described together with the solution chosen in GAS.

2. Related work

A proof-of-concept service called "SIM strong authentication" was presented at the 3GSM World Congress in Barcelona, February 2006 [1]. The goal of this proof-of-concept was to demonstrate the possibility of implementing innovative service in a heterogeneous environment using Liberty Alliance Federation Standard [2]. The implementation used elements such as Java Card, Active-X supplicant, IDP Java Authenticator, Vital AAA server and Signalware gateway. The weakness with this proof-of-concept was that the Supplicant implementation required new SIM cards with a preinstalled Java Card application and Windows XP SP2/Internet Explorer with Active-X support enabled.

Similar authentication services that use smart cards and public key infrastructure (PKI) are

deployed in many countries. BankID[10] is an example of a standardized and coordinated electronic identification issued by a Norwegian bank, which is used on the Internet for identification and digital signatures. To be issued a BankID one needs to be customer of a bank that supports it. This solution differs from the one described in this paper on many different points. The BankID needs two extra devices (smart card and offline reader), it has only one choice of security level and it is not designed for use in a single sign-on system.

3. Generic SIM Authentication System

GAS could be used with several different services like SIP, e-mail, access control, and Internet services. Figure 1 shows the deployment of the prototype that has been developed. Two types of client software have been developed, a standalone Java Application and a Java Applet, both named the Supplicant. The functionality related to the authentication is the same, but the Applet is provided through web browser using https to secure the communication.

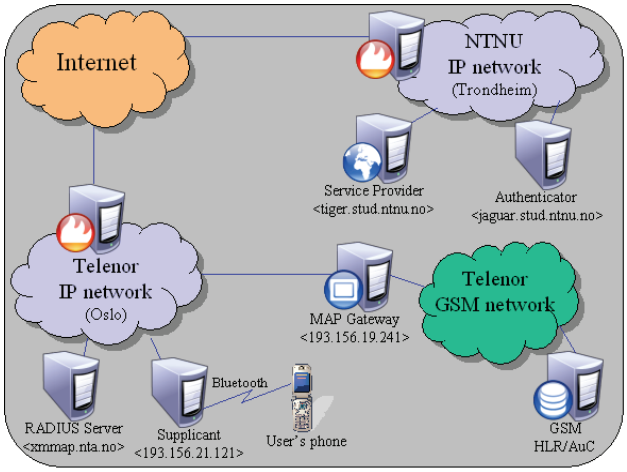


Figure 1. Deployment of the prototype

The Supplicant communicates with a SIM reader (here shown as the User's mobile phone) and the Authenticator. The Authenticator is responsible for authenticating the Supplicant on behalf of the Service Provider (SP). If the authentication was successful, the Authenticator vouches for the Supplicant, and the SP gives access to the service. The Authenticator uses Authentication Servers (here shown as a RADIUS server) to perform parts of the authentication. The Authentication Server uses the MAP Gateway to communicate with the GSM Network during the authentication. The authentication process will be elaborated in section 4 and a more thorough description of the components will be described in section 5.

Specific for the developed prototype, the user will be guided through a few steps during the authentication. When trying to login, the user is prompted if he/she wants to login using a smart card reader or a Bluetooth device having SIM Access Profile (SAP). If SAP is chosen, a list of earlier used devices emerges and the user can either select the appropriate device or start a new Bluetooth device discovery. The Supplicant then sets up a connection to the SIM (via the chosen method) and the user is prompted for the PIN. When the Supplicant has been authorized to use the SIM (PIN code is verified), a connection towards the Authenticator is set up using a secure socket and the authentication process towards the Authenticator will start.

The Applet is downloaded automatically when accessing the web page of either the Service Provider or the Authenticator, and does not need any additional files or installations, maybe except from hardware drivers. It is tested successfully on Linux, Mac OS X, and Windows XP with browsers like Firefox, Opera, Internet Explorer, and Safari. The tests are performed end-to-end including the GSM MAP Gateway.

A user-friendly alternative to using the mobile phone as the reader is to have dual or triple SIM subscriptions, where one SIM is located on a USB dongle, a 3G PC Card, or integrated with the Supplicant to maintain easy access.

4. Overall architecture

This section starts by introducing a sequence diagram used to identify messages and components needed in the GAS. The overall architecture is shown using a high-level component diagram. This section clarifies which components that exist and which that do not exist.

The specific messages shown in this section are based on EAP between the Service Supplicant and the Authenticator, and EAP over RADIUS between the Authenticator and the Authentication Server. RADIUS is chosen because it is most widespread, but it could just as easily be performed with for instance DIAMETER. The EAP protocol is extended with EAP-SIM in the GSM authentication specific messages, according to the EAP-SIM specification RFC 4186 [3]. If another authentication protocol is chosen instead of GSM authentication, the correct EAP type specific protocol will be used (e.g. EAP-TLS for PKI and EAP-OTP for OTP).

Figure 2 shows the GSM SIM authentication using EAP-SIM during a full authentication. When

[10] BankID, http://www.bankid.no

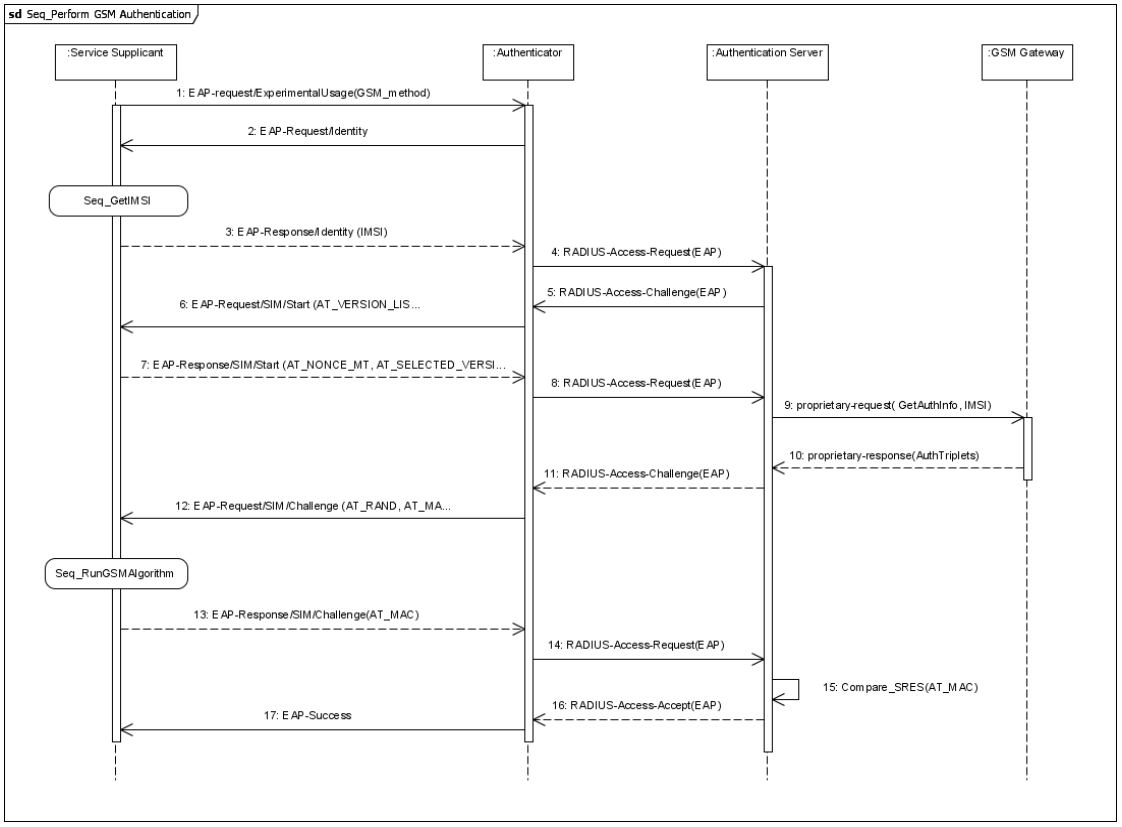


Figure 2. Sequence diagram authenticating a user

the Authenticator is initialized it produces a list of possible authentication methods based on the different Authentication Servers (and which methods they offer) the Authenticator has mutual trust with. A User accessing a service triggers the process supported by GAS. The Service initializes the Supplicant (e.g. by an applet download) with the minimum required security level needed by the service. The Supplicant asks the Authenticator for a list of supported authentication methods that satisfies the given security level, and when the Supplicant chooses a suitable method from that list (here shown by the first message) the authentication starts. Seq_GetIMSI and Seq_RunGSMAlgorithm is the communication towards the SIM where the ID (IMSI) and SRES/Kc[11] are retrieved, respectively.

Now, the different components are recognized. Figure 3 presents the components of GAS. The Supplicant consists of both a Service Supplicant and a SIM supplicant, where the Service Supplicant communicates with the Authenticator over EAP and the SIM Supplicant communicates with the SIM with Application Protocol Data Units (APDUs) over T=0 [4]. The Authenticator communicates with the Authentication Server over RADIUS and uses RADIUS' interfaces. The communication between the Authentication Server and the GSM gateway is depending on the implementation of the GSM Gateway used by the Authentication Server. The GSM Gateway communicates with the GSM Network using MTP3 over SS7.

The Supplicant does not exist and needs to be designed and implemented. The Service Supplicant needs a communication channel towards the Authenticator and the SIM Supplicant, hence an Authenticator Client and a SIM Client. The SIM Supplicant must have an interface giving the Service Supplicant the possibility to retrieve IMSI and SRES/Kc, and be able to communicate with the SIM, i.e. a communication channel allowing GSM SIM related APDUs to be sent.

The Authenticator does not exist and needs to be designed and implemented. It needs to have a listener ready for incoming connections from Supplicants and be able to interpret the received EAP-messages. Depending on the Authentication Server, the Authenticator must have an Authentication Server Client (e.g. a RADIUS Client) to be able to communicate with the Server. The Authenticator should store some information to be able to confirm the successful authentications to Service Providers.

[11] SRES (Signed RESponse): used to authenticate a GSM subscriber
Kc: GSM encryption key

In the backend, GAS has an Authentication Server that authenticates the Supplicants. A RADIUS server will be used as the Authentication Server because it is currently the de-facto standard for remote authentication. Hence, the Authentication Server exists, but it needs plug-ins. To perform the GSM/UMTS authentication the RADIUS server will use the EAP-SIM Plug-in (some servers does not have the EAP-SIM plug-in, hence it needs to be implemented) and a GSM Gateway Client. For other types of authentication additional plug-ins (i.e. EAP-OTP Plug-in) are used.

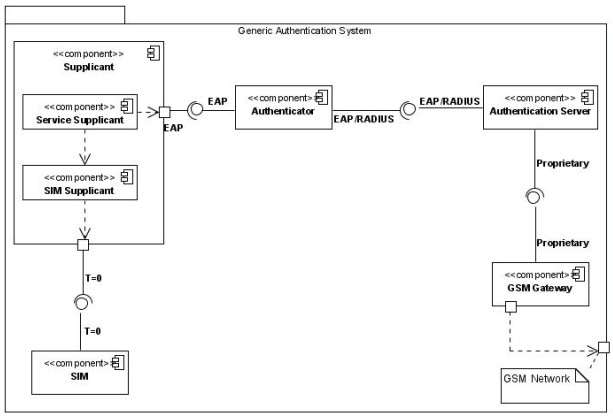


Figure 3. The components of GAS

The SIM is ideally not changed, i.e. no implementation or installation of a SIM application is required (thus e.g. JavaCard is not needed).

The GSM Gateway will not be explored any more since the GSM Gateway interfaces are proprietary.

5. Detailed description of the components

The GAS builds upon the existing GSM authentication infrastructure, thus allows re-use of GSM expertise from the mobile operators.

This chapter examines the non-existing components of the system.

5.1. SIM communication

The SIM Supplicant may be located on either the same unit as the Service Supplicant or on different units. The Service Supplicant asks for credentials and the SIM Supplicant performs the communication towards the SIM via a SIM reader. There are two main classes of SIM readers, the integrated reader in the mobile phone and the smart card reader. They incorporate different security mechanisms, where the former is stricter and less implemented. Both readers communicate with the SIM over GSM 11.11 [5], which means to transfer APDUs from the computer to the SIM.

To communicate with the SIM card through the mobile phone the phone must support a SIM access interface. The phone manufacturers are often holding back the interface to the SIM card because of security reasons, and therefore many of the standardized access mechanisms are not implemented by current mobile phones. Three such possibilities will be mentioned below.

AT-commands (Hayes Command Set) are a set of commands that mobile phones implement, but unfortunately no phone has been found which supports the AT+CSIM (gives generic access to the SIM).

The Security and Trust Services APIs (SATSA) is an optional profile to Java Platform, Micro Edition (Java ME), and provide smart card access and cryptographic capabilities to applications running on small devices. No phone with the optional APDU package that gives access to the credentials has been found.

The Bluetooth SIM Access Profile (SAP) seems like the most promising technology for the future. By April 2006, phones supporting SAP has been found on 11 BenQ-Siemens models and on 25 Nokia models (see list at [6]). SAP does not need the SIM Supplicant to be installed in the phone, and the SIM Supplicant will therefore be located with the Service Supplicant.

Before the SAP connection can be started the user needs to turn on Bluetooth and allow remote SIM connections on the mobile phone. To use the SAP the SIM Access Client (SIM Supplicant) needs to perform a Bluetooth service discovery to find the channel number that the SIM Access Server (mobile phone) is using. The SAP service name is “SIM Access” and the service class id is 0x112D. The implementation of the Bluetooth client uses the Avetana library in Java, which supports the built in Bluetooth stacks on most operating systems.

In order to ensure secure communication between the client and the server the SIM Access Profile specification identifies several mandatory security demands:

- The passkey used in the pairing of the devices must be minimum 16-digit long.
- The link between the client and server shall be encrypted using Bluetooth baseband encryption with a minimum encryption key length of 64 bits.

Given that newer Bluetooth devices uses 128 bits encryption (considered safe) and the long passkey, the Bluetooth link is hard to exploit.

To communicate with the smart card reader the SIM Supplicant need to use the readers API. Today

there exist different types of standardized smart card interface APIs; the two most known are PC/SC and OCF. The PC/SC API has become the de-facto standard implemented by most smart card readers on the market today. Because of this the PC/SC interface was a natural choice for the implementation. The PC/SC interface works on many platforms, either through the Win32 API (Windows) or the PCSC Lite tool (Mac and Linux). The communication with the smart card reader from Java needs a library that implements an interface between Java and the lower level PC/SC on the operating system. In this project an open source library called JPCSC [7] is used, which implements a Java Native Interface (JNI) wrapper to allow for the access to PC/SC functions from Java. JPCSC offers an API to establish a connection with the SIM card and communication through APDUs. JPCSC supports both Windows and Linux, and with some small changes it also works on Mac OS X.

5.2. EAP over secure socket

The EAP specification only discusses usage within a point-to-point protocol (PPP), which is a low layer protocol (the Data Link layer of the OSI model). The encapsulation of EAP messages in higher layer protocols is not specified in the specification. The message format is byte-oriented and the EAP attribute of the RADIUS packet uses this format. An alternative representation is to use the eXtensible Markup Language (XML), but this would require extra translation at the Authenticator. The transportation between the Supplicant and the Authenticator can be UDP, TCP, HTTP or SOAP

To have a generic and secure communication between the Authenticator and the Supplicant EAP over TCP/IP was chosen with TLS (Transport Layer Security) to maintain the integrity and the confidentiality. The authentication is resolved by EAP at the Application Layer.

EAP should be interpreted and understood by the Supplicant, Authenticator, and the Authentication Server. A correct implementation according to the specifications is crucial in all parts of the system. EAP is implemented according to RFC 3748 [8]. The EAP-packet is a shell containing an authentication procedure. The focus on the prototype of the system is on the authentication protocol EAP-SIM.

EAP-SIM specifies an EAP-based mechanism for mutual authentication and session key agreement using the SIM card. The method includes enhancements to the GSM authentication, such as mutual authentication and larger random numbers and ciphering keys. To provide mutual authentication, the EAP-SIM client issues a random number (NONCE) to the network that is used to verify correct identity of the authentication server. To provide a larger ciphering key, the EAP-SIM method uses two or more authentication triplets (random number (RAND), SRES, Kc) to generate the corresponding amount of ciphering keys, which are combined into one large key. It is crucial that GAS checks that the RANDs are different and that at least two are being used to obtain 128 bits security. The security is limited to 128 bits because the Ki (the secret key used in GSM) is 128 bits long.

EAP-SIM uses several cryptographic algorithms when generating keys and exchanging credentials, among them SHA-1, HMAC using SHA-1, and a special variant of SHA-1 with the message padded only with zeroes. These algorithms have also been implemented in the prototype. These algorithms are considered safe, even though SHA-1 is recommended changed in the future due to newly discovered weaknesses. HMAC-SHA1 is not affected by the SHA-1 weakness.

5.3. RADIUS client

The RADIUS client is a part of the Authenticator and is used to communicate with the Authentication Server. There is mutual trust between the Authenticator and the Authentication Server due to a shared secret. The main functionality of the Authenticator is to locate a suitable Authentication Server, act as a translator from EAP to RADIUS and back, store information about authorized Supplicants, and keep track of identities when for instance temporary identities are used. When the Authenticator receives EAP messages from the Supplicant, it will forward these inside a RADIUS packet to the RADIUS server. All RADIUS packets received from the RADIUS server is checked for validity and forwarded to the correct Supplicant.

RFC 3579 [9] specifies additional attributes for providing EAP support within RADIUS; EAP-Message and Message-Authenticator. The EAP-Message attribute is assigned the type number 79. The EAP-message is placed in the value field; if the EAP-message is larger than 253 bytes it can be divided into multiple EAP-Message attributes and the RADIUS server will concatenate them at reception. If a RADIUS server receives an EAP messages that it does not understand it should return an Access-Reject.

The implementation of the RADIUS client uses code from the open source project JRadiusClient [10], which is an implementation of a RADIUS client intended to be used as a library. The JRadiusClient library does not have EAP support, to support this extension an entire new client was developed from

scratch using the source code from the JRadiusClient. This was necessary because of the way the library was implemented; it was difficult to add this extension without changing the core of the library itself. The Radius client is compliant with the RFC 2865 [11] (RADIUS spec.) and RFC 3579.

The prototype is tested with two different RADIUS servers that both support EAP-SIM; FreeRadius and NavisRadius. The EAP-SIM plug-in in FreeRadius is called "rlm_eap_sim", currently it only supports gathering of GSM triplets through a text file.

The NavisRadius plug-in is called "AuthEapSim", and the gathering of GSM triplets supports three different methods:

1. Retrieve triplets from a HLR using a MAP gateway towards the GSM network
2. Read triplets from a triplet file
3. Generate triplets at the RADIUS server

Method 2 and 3 is considered to be most useful for testing purposes, where method 1 is the most realistic in a deployed system. The ReadMapGateway (method 1) plug-in needs contact with a GSM network to communicate with the HLR. Since NavisRadius operates on an IP network and an HLR operates on a SS7 network, requests for the HLR need to be made through a special GSM Gateway, e.g. NavisRadius supports using the Ulticom MAP Gateway. Nevertheless, the system has been tested with method 1 towards a real GSM network, to verify that all components interoperate as expected.

6. Future work

The GAS can be easily extended as an authentication mechanism in a Single Sign-On (SSO) system. Liberty Alliance and their federated identity management have been considered. By integrating the Authenticator with an Identity Provider (IDP) and extending the Service Providers according to Liberty Alliance it is possible to have an SSO where SPs trust an IDP, which vouches for the Supplicant after an authentication.

The GAS will be integrated and tested with different services, like an SSO system, SIP, e-mail, access control etc. Other authentication mechanisms on the SIM will be implemented to support several security levels. OTP and PKI have been tested on SIMs, and they are already in use on smart card, which is the same technology (UICC – Universal Integrated Circuit Card) as SIM.

The Supplicant will also be integrated on a Pocket PC/PDA, i.e. the SIM is located with the Supplicant.

7. Conclusion

This paper has proposed a novel design of a generic authentication system based on SIM, together with a detailed description of a prototype. The system has been tested end-to-end and provides a strong authentication mechanism. New services can easily be supported, such that these can benefit from strong authentication. By gradually implementing more authentication mechanisms (e.g. OTP and PKI) on the SIM, it will be able to support several levels of security. This will result in a generic authentication system approving security needs for nowadays and also for the future.

8. References

[1] T. van Do, et al, "Offering SIM Strong Authentication to Internet Services", White Paper, 3GSM World Congress, Barcelona, February 2006.

[2] T. Wason, et al., "Liberty ID-FF Architecture Overview; Version: 1.2-errata-v1.0", Liberty Alliance Project, 2005.

[3] H. Haverinen, J. Salowey, "EAP-SIM Authentication", RFC 4186, IETF, January 2006.

[4] ISO, "ISO 7816 Part 4: Organizations, security and commands for interchange", ISO, 2005.

[5] "Digital cellular telecommunications system (Phase 2+); Specification of the Subscriber Identity Module-Mobile Equipment (SIM-ME) interface", GSM 11.11 version 5.0.0, ETSI, December 1995.

[6] "Nokia 616 Compatibility", Nokia, April 2006: <http://www.nokia.no/phones/enhancements/616/compatibility.php>

[7] IBM, "JPC/SC Java API 0.8.0", MUSCLE, April 2006: <http://www.linuxnet.com/middle.html>

[8] B. Aboba, et al., "Extensible Authentication Protocol (EAP)", RFC 3748, IETF, June 2004.

[9] B. Aboba, P. Calhoun, "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)", RFC 3579, IETF, September 2003.

[10] A. Abouchi, B. Loihl, "JRadiusClient 2.0.0", March 2006: <http://www.java-radius-client.fr.fm>

[11] C. Rigney, et al., "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, IETF, June 2000.

References

[1] Entrust, “Survey Finds Identity Theft Negatively Impacting Consumer Use of the Internet”, Entrust Press Release, October 2004

[2] H. Haverinen, J. Salowey, “EAP-SIM Authentication”, RFC 4186, IETF, January 2006.

[3] Gemplus, “Gemplus launches GEMobileIT - a SIM-Secure solution for access to data over public WLAN”, Press Release, Gemplus, February 2004.

[4] V. Khu-Smith, C.J. Mitchell, “Using GSM to Enhance E-Commerce Security”, In proceedings of the 2nd ACM International Workshop on Mobile Commerce (WMC ’02), New York, 2002, pp. 75-81.

[5] A. Pashalidis, C. Mitchell, “Using GMS/UMTS for Single Sign-On”, Mobile Future and Symposium on Trends in Communications (SympoTIC '03), IEEE Press, Bratislave, 2003, pp.138-145.

[6] T. van Do, A. Nachef, J.D. Aussel, et al, “Offering SIM Strong Authentication to Internet Services”, White Paper, 3GSM World Congress, Barcelona, February 2006.

[7] S.T. March, G.F. Smith, “Design and natural science research on information technology”, Decision Support Systems 15, Elsevier Science, 1995, pp. 251-266.

[8] A.R. Hevner, S.T. March, J. Park, et al., ”Design Science in Information Systems Research”, MIS Quarterly, 28(1), March 2004, pp. 75-105.

[9] V. Vaishnavi, B. Kuechler, “Design Research in Information System”, Association for Information System, June 2005. October 2005: <http://www.isworld.org/Researchdesign/drisISworld.htm>

[10] L. Lunde, “Using SIM for strong end-to-end application authentication”, Project Assignment, NTNU, December 2005.

[11] A. Wangensteen, “Using SIM for strong end-to-end application authentication”, Project Assignment, NTNU, December 2005

[12] P. Congdon, B. Aboba, A. Smith, et al., “IEEE 802.1X Remote Authentication Dial-In User Service (RADIUS) Usage Guidelines”, RFC 3580, IETF, September 2003.

[13] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[14] N. Haller, C. Metz, P. Nesser, et al., ”A One-Time Password System”, RFC2289, IETF, February 1998.

[15] C. Adams and S. Lloyd, *Understanding PKI 2nd edition: Concepts,Standards, and Deployment Considerations*, Pearson Education Inc., 2003.

[16] OATH, "An Industry Roadmap for Open Strong Authentication", White Paper.

[17] Bankenes Betalingssentral AS, "BankID COI White Paper: Version v1.0", Bankenes Betalingssentral AS, September 2005.

[18] 3GPP, "Specification of the Subscriber Identity Module - Mobile Equipment (SIM-ME) Interface v.4.15.0", TS 51.011, 3GPP, June 2005.

[19] 3GPP, "Specification of the SIM Application Toolkit for the Subscriber Identity Module - Mobile Equipment (SIM - ME) interface v.4.5.0", TS 51.014, 3GPP, January 2005.

[20] International Numbering Plans. October 2005: <http://www.numberingplans.com/>

[21] 3GPP, "Security-related network functions v.8.1.0", TS 03.20, 3GPP, June 2000.

[22] 3GPP, "Security mechanisms for the SIM application toolkit v.8.9.0", TS 03.48, 3GPP, June 2005.

[23] 3GPP, "Subscriber Identity Module Application Programming Interface (SIM API) for Java Card v.8.5.0", TS 03.19, 3GPP, September 2002.

[24] I'M Technologies, "How do UMTS, UICC, USIM and USAT-(I) fit together?", White Paper

[25] 3GPP, "Characteristics of the USIM application v.7.1.0", TS 31.102, 3GPP, June 2005.

[26] 3GPP, "3G security; Security architecture v.6.3.0", TS 33.102, 3GPP, December 2004.

[27] 3GPP, "3G Security; Specification of the MILENAGE algorithm set; Algorithm specification v.6.0.0", TS 35.206, 3GPP, January 2005.

[28] V. Niemi, K. Nyberg, *UMTS Security*, John Wiley & Sons, 2003.

[29] Bluetooth Special Interest Group, "SIM Access Profile", version 1.0, May 2005

[30] ISO, "Identification cards -- Integrated circuit cards -- Part 4: Organization, security and commands for interchange", 7816-4, ISO, 2005.

[31] PC/SC Workgroup, "PC/SC Workgroup Specifications 2.01, all parts", September 2005.

[32] M.U.S.C.L.E., "Movement for the Use of Smart Cards in a Linux Environment", <http://www.musclecard.com>

[33] Smart Card Alliance, “Feature of the Month, Smart Cards and the IT Infrastructure”, Smart Card Talk, 10(2), February 2005.

[34] IEC, “Global System for Mobile Communication”, The Internet Engineering Consortium, 2000.

[35] Clint Smith and David Collins, “3G Wireless Networks”, McGrew-Hill, 2002.

[36] C.-C. Lee, M.-S- Hwang and W.-P. Yang, “Extension of authentication protocol for GSM”, Communications, IEEE Proceedings, 150(2), April 2003, pp. 91-95.

[37] B. Aboba, et al., “Extensible Authentication Protocol (EAP)”, RFC 3748, IETF, June 2004.

[38] H. Haverinen, J. Salowey, “EAP-SIM Authentication”, RFC 4186, IETF, January 2006.

[39] J. Arkko, H. Haverinen, “Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA)”, Internet-Draft, IETF, December 2004.

[40] B. Aboba, D. Simon, “PPP EAP TLS Authentication Protocol”, RFC 2716, IETF, October 1999.

[41] M. Nystroem, “The Protected One-Time Password Protocol (EAP-POTP)”, Internet Draft, IETF, October 2005.

[42] RSA Security, “EAP-POTP: An Extensible Authentication Protocol (EAP) Method for OTP Algorithms”, Presentation, RSA Conference, February 2005.

[43] P. Urien, et al., “EAP Smart Card Protocol (EAP-SC)”, Internet-Draft, IETF, March 2005.

[44] P. Calhoun et al., “Remote Authentication Dial In User Service (RADIUS)”, RFC 2865, IETF, June 2000.

[45] B. Aboba, P. Calhoun, “RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)”, RFC 3579, IETF, September 2003.

[46] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, IETF, February 1997.

[47] Ericsson Research, “Diameter vs. Radius”, Ericsson AB, 2003. September 2005: <http://www.3gpp.org/ftp/tsg_sa/WG3_Security/TSGS3_31_Munich/Docs/PDF/S3-030736.pdf>

[48] P. Calhoun, J. Loughney, E. Guttman, et al., “Diameter Base Protocol”, RFC 3588, IETF, September 2003.

[49] IEEE, “IEEE 802.15 WPAN Task Group 1 (TG1) – Web Site”. April 3, 2006: <http://ieee802.org/15/pub/TG1.html>

[50] ”The Official Bluetooth Membership Site”. April 3, 2006: <http://www.bluetooth.org>

[51] SIG, “Bluetooth Specification version 1.1 – Serial Port Profile”, Bluetooth Special Interest Group, February 2001, pp. 172-196.

[52] “Terminal Equipment to Mobile Station (TE-MS) multiplexer protocol v.7.2.0”, TS 07.10, 3GPP, March 2002.

[53] Andrew S. Tanenbaum, *Computer Networks 4th edition*, Prentice Hall, Upper Saddle River, 2003.

[54] SIG, ”Specification of the Bluetooth System Version 2.0 + EDR [vol0]”, Bluetooth Special Interest Group, November 2004.

[55] SIG, ”SIM Access Profile, Interoperability Specification v 10r00”, Car WG, Bluetooth Special Interest Group, May 2005.

[56] 3GPP, “AT command set for GSM Mobile Equipment (ME) v.7.8.0”, TS 07.07, 3GPP, March 2003.

[57] “Visual Paradigm, Visual Paradigm for UML 5.2 Modeler Edition”, <http://www.visual-paradigm.com>

[58] R. Malan, D. Bredemeyer, “Functional Requirements and Use Cases”, White Paper, 2001.

[59] R. Malan, D. Bredemeyer, “Defining Non-Functional Requirements”, White Paper, 2001.

[60] Object Management Group, “Unified Modeling Language: Superstructure version 2.0”, Specification, August 2005.

[61] D. Coleman, “A Use Case Template: Draft for discussion”, June 1998.

[62] OMG, “Unified Modeling Language: Superstructure v.2.0”, OMG – Object Management Group, August 2005.

[63] IBM, “JPC/SC Java API 0.8.0”, <http://www.linuxnet.com/middle.html>

[64] BlueZ Project, <http://www.bluez.org>

[65] Avetana, “Avetana JSR-82 implementation, version 200510223”. April 2006: <http://www.avetana-gmbh.de/avetana-gmbh/produkte/jsr82.eng.xml>

[66] “Digital Signature Standard (DSS)”, FIPS PUB 186-2 Change Notice #1, US Department of Commerce/National Institute of Standards and Technology, January 2000.

[67] “Secure Hash Standard (SHS)”, FIPS PUB 180-2 Change Notice #1, US Department of Commerce/National Institute of Standards and Technology, August 2002.

[68] NIST, “The Keyed-Hash Message Authentication Code (HMAC)”, FIPS PUB 198-a, US Department of Commerce/National Institute of Standards and Technology, March 2002.

[69] A. Abouchi, B. Loihl, ”JRadiusClient 2.0.0”, <http://www.java-radius-client.fr.fm>

[70] The Apache Ant Project, “Apache Ant, version 1.6.5”, The Apache Software Foundation, <http://ant.apache.org>

[71] The Apache Ant Project, “Apache Ant 1.6.5 Manual”, The Apache Software Foundation. April 2006: <http://ant.apache.org>

[72] Sun Microsystems, “keytool - Key and Certificate Management Tool”, Sun Microsystems. April 2006: <http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/keytool.html>

[73] Greg Travis, “Using JSSE for secure socket communication”, IBM, April 2002.

[74] Sun Microsystems, “jarsigner - JAR Signing and Verification Tool”, Sun Microsystems. April 2006: <http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/jarsigner.html>

[75] Ethereal, “Ethereal - Network Protocol Analyzer 0.99.0”, <http://www.ethereal.com>

[76] Lucent Technologies, ”NavisRadius 4.5.6”, <http://www.lucentradius.com>

[77] FreeRADIUS Development Team, ”FreeRADIUS 1.1.0”, <http://www.freeradius.org/>

[78] The Apache Software Foundation, “Apache Tomcat, version 5.0.28”. April 2006: < http://tomcat.apache.org>

[79] The Apache Software Foundation, “Apache HTTP Server, version 2.0.54”. May 2006: < http://httpd.apache.org>

[80] ChilliSpot, “ChilliSpot 1.0”, < http://www.chillispot.org>

[81] JUnit, < http://junit.org>

[82] J. Quirke, “Security in the GSM system”, AusMobile, May 2004.

[83] S. Patel, “Analysis of EAP-SIM Session Key Agreement”, Lucent Technologies, 2004.

[84] P. Eronen and H. Tschofenig, ”Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)”, RFC 4279, December 2005.

[85] D. Bleichenbacher, “Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1”, CRYPTO'98, LNCS 1462, pp. 1-12, 1998.

[86] X. Wang, D. Feng, X. Lai, et al., ” Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD”, August 2004.

[87] Cryptography Research, “Hash Collision Q&A”, Cryptography Research, February 2005. April 2006: <http://www.cryptography.com/cnews/hash.html>