Radboud University Nijmegen

Bachelor Thesis

Encrypted SMS, an analysis of the theoretical necessities and implementation possibilities

Author:
Lars Lockefeer

Supervisors:
Engelbert Hubbers
Roel Verdult

July 19, 2010

Abstract

With the emerge of the mobile phone over the course of the years, *Short Message Service* (SMS) has been widely adopted as the standard for quick and easy communication. The implementation of SMS however, lacks support of security services exposing messages that are sent through the system to adversaries. In this thesis, we will have an in-depth look at the current implementation of SMS. Then, we will investigate the aspects that are of importance for secure communication. Thereafter, we will describe the vulnerabilities within the current implementation of SMS, and identify the threats that the system is exposed to as a result of these vulnerabilities. We will then have a look at some existing implementations, to see that they contain flaws and do not address all threats. Then, we will propose two protocols that can easily be implemented to facilitate secure communication using the current implementation, explain how these protocols can be implemented and why they do address the threats sufficiently. Finally, we will end with a conclusion on securing SMS, and give some pointers for further research.

Contents

Chapter 1

Introduction

With the emerge of the mobile phone over the course of the years, *Short Message Service* (SMS) has been widely adopted as the standard for quick and easy communication. While some only use it to communicate low-profile information that, except for the receiver, nobody is interested in, others have adopted the standard to share confidential information. The devices used by some of these people have implemented extra security measures but it is not likely that these measures are available to be used by anyone that shares confidential information over SMS.

While the relatively open character of SMS only leads to slight discomfort for the larger part of SMS traffic, it is not difficult to think of confidential messages that might lead to serious risks when read by the wrong people. It is not unlikely, for example, that a couple of ministers of a country decide that they immediately have to gather secretly, and agree upon a meeting place through SMS. Now, when the press would be tapping their communication, the meeting would not be so secret at all.

It gets even worse, when SMS is used for authentication, which is done by one of the major banks in The Netherlands. The following example describes an attack scenario, that would make the system much more vulnerable to misuse by criminal organisations.

Several studies have shown vulnerabilities in online banking systems. Due to the rather open character of modern browsers, malicious browser extensions can be developed, and then installed on the machines of non-suspicious users. When using the online banking system, these extensions will constantly modify the communication between the user and the bank, and rewrite the pages shown by the banking system. By doing this, the developers hope to interfere with the transaction, transferring money to an account they own instead of the determined account, without making the user suspicious. To prevent this, all banks have implemented a phase into the transaction, in which the user has to check and confirm the amount transferred through a separate device.
However, when this confirmation is done through SMS, as is the case for one of the major banks in The Netherlands, the attack can be enhanced by monitoring the SMS traffic of a user. The SMS that is sent from the bank to the user, which contains the amount of money transferred in the transaction and a confirmation code, can then be intercepted, changed and forwarded to the user. The user will not be suspicious at all and confirm the transaction.

In this thesis, we will first take a look at the current implementation of SMS (Chapter 2). Then, we will investigate the various aspects that are of vital importance when communicating securely over a certain channel (Chapter 3). After this, we will investigate the vulnerabilities of SMS and define the threats that the current implementation is exposed to (Chapter 4). In Chapter 5, we will have a look at some existing implementations, to see that they contain flaws and address only some of the threats identified in Chapter 4. Therefore, we propose two protocols that facilitate secure communication through SMS, details on how to implement them, and explain why these protocols do address the threats sufficiently. We will end this thesis with some general conclusions on the matter, and some pointers for further research (Chapter 6).

Abbr.	**Full name**	**Function**
MS	Mobile Station	An entity, such as a cellphone or PDA, that is used to participate in GSM traffic.
NSS	Network Switching Subsystem	The component of a GSM network that manages the communication between a mobile station and other peers, such as other Mobile Stations or telephones that are connected through a landline.
BSS	Base Station Subsystem	Those entities within the network that are responsible for handling traffic between a Mobile Station and the Network Switching Subsystem. The BSS consists of several Base Transceiver Stations that are controlled by a Base Station controller.
BTS	Base Transceiver Station	A Base Transceiver Station is an entity within the GSM network that receives messages sent by Mobile Stations and delivers them to a Base Station Controller, or broadcasts messages that it receives from a Base Station Controller to Mobile Stations. Messages between the MS and the BTS are transferred over the air.
BSC	Base Station Controller	A Base Station Controller controls several Base Transceiver Stations, forwarding messages between the BTS and the MSC. One Base Station Controller controls several Base Stations.
SMSC	Short Message Service Center	The entity within the GSM network, that is responsible for the delivery of short messages to the receiving Mobile Station. Messages sent by a Mobile Station that is subscribed to the network are first stored, and then forwarded to the receiving Mobile Station.
MSC	Mobile Switching Center	An entity within the GSM network, that is responsible for the routing of calls and messages to the correct Mobile Station or Service Center
HLR	Home Location Register	A database within the GSM network, that contains general information on the subscribers to that network, such as their identity, the services they have subscribed to etc. The HLR also contains the location of the subscriber.
VLR	Visitor Location Register	A database, coupled to a specific Base Station, that contains the location and information obtained from the HLR for all subscribers that are present in the service area of that Base Station.
SIM	Subscriber Identity Module	A chip that contains, among other data, the International Mobile Subscriber Identity (IMSI) number of a subscriber. This unique number is used to identify the user across the GSM network.
	Location Area	An area comprising of several Base Transceiver Stations controlled by a Base Station Controller.
	Message	A 140-byte message entity that is transmitted between two users.
	Concatenated message	Two or more messages that are sent to the user at the same time. Note that these messages always have length that is a multitude of 140 bytes.

Table 1.1: Entities in SMS transmission

Chapter 2

SMS

For this thesis, the following aspects of the workflow are important:

1. The transmission of an SMS over the air, from sender to receiver
2. The storage of sent/received messages on the cellphone of both sender and receiver

In this chapter, we will describe both aspects of the workflow in detail. Most of the information presented in this chapter is adapted from [20, 9, 8].

2.1 SMS transmission

Short Message Service (SMS) provides a connectionless transfer of messages. A single message consists of 140 bytes, which means an SMS can contain 160 7-bit or 140 8-bit characters. The structure of such a message is given in figure 2.1. Messages can pass in both directions, and are transported on the GSM signalling links which allows messages to be received while the subscriber is calling. Multiple messages to one receiver can be sent at once, and will be concatenated by the software in most mobile phones. Note that messages are always sent in a multitude of 140 bytes. Throughout this thesis, whenever we refer to a *message* we mean a single, 140 byte message that is transmitted between two users. Multiple messages sent at once are referred to as *concatenated messages*. There are two types of messages: a cell broadcast message, that is sent to all subscribers in a specific area and a point to point message that is sent to a specific subscriber. Since this thesis focuses on secure communication between two people over SMS, we will only describe those details that involve point to point messaging. For the sake of clarity, all entities and their function that are introduced during this chapter are listed in table 1.1 on page 4.

Instructions to air interface	Instructions to SMSC	Instructions to Mobile Station	Instructions to SIM (optional)	Message body

Figure 2.1: The Structure of a Message - adapted from [14]

2.1.1 Flow of the message

When a message is sent from a Mobile Station to another Mobile Station, it passes several entities before it is delivered. Below, we give a short overview of the message path. An overview of this path is also given in figure 2.2.

1. The message is delivered from the *Mobile Station* (MS) of the sender, such as a cellphone or PDA, to a *Mobile Switching Center* (MSC). Often, this MSC is referred to as the Gateway MSC. The delivery is done through the *Base Station Subsystem* (BSS). The BSS transports GSM Data Units over the air, from Mobile Stations to the Network Switching Subsystem (NSS).
2. The Mobile Switching Center, that is situated in the GSM network, is connected to a *Short Message Service Center* (SMSC), to which the message is delivered.

3. The Short Message Service Center is connected to the GSM network and determines the Mobile Switching Center to which the receiver of the message is currently subscribed, by querying the *Home Location Register* (HLR). It forwards the message to that Mobile Switching Center through the GSM Roaming Protocol.
4. The Mobile Switching Center determines the exact location to deliver the message by querying the *Visitor Location Register* (VLR), and sends the message to the BSS. The BSS then delivers the message at the Mobile Station of the receiver.

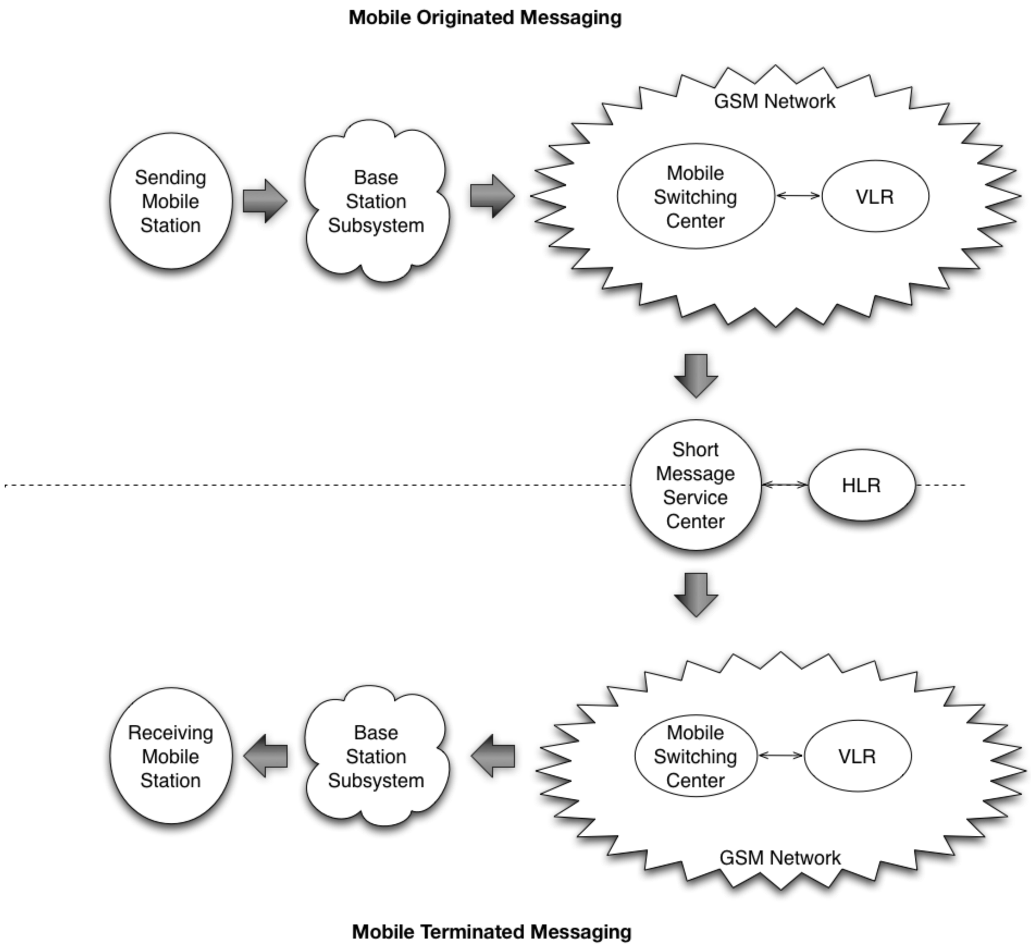

Figure 2.2: The Flow of a Short Message from Sender to Receiver

To be able to receive SMS, the Mobile Station of the receiver must contain special software to enable the messages to be processed and stored upon receiving. Storing the messages is usually done on the *Subscriber Identity Module* (SIM) of the MS. In section 2.4, we will elaborate more on this matter.

2.2 The Protocol hierarchy

Several protocols interact with each other during SMS transmission. In this section, we will discuss the most important layers of the protocol hierarchy and the Data Units they utilise for communication. An overview of the hierarchy can be found in figure 2.3.

2.2.1 The Short Message Transfer Layer

The Short Message Transfer Layer (SM-TL) is used to transfer messages that are generated at the Short Message Application Layer and their delivery reports. Within the layer, each message is identified using a *Short Message Identifier* (SMI) that is generated once a message is registered with the layer. The SM-TLs of an MS and a SMSC communicate with each other using the *Short Message Service Transfer Protocol* (SMS-TP). This protocol consists of four types of Data Units (TPDU):

1. **SMS-SUBMIT**
   A Data Unit that transports a message from the Mobile Station to the Short Message Service Center. Optionally, a validity period is specified by this Data Unit, so that the message can be buffered in the Short Message Service Center if it cannot be delivered immediately.

2. **SMS-DELIVER**
   A Data Unit that transports a message from the Short Message Service Center to the Mobile Station of the recipient. This Data Unit includes a time stamp, that is used by the Short Message Service Center to inform the recipient Mobile Station about the time that the message arrived at the Short Message Transfer Layer of the Short Message Service Center. Furthermore, this Data Unit contains a parameter that indicates whether there are one or more messages waiting in the Short Message Service Center to be delivered to the recipient Mobile Station.

3. **SMS-STATUS-REPORT**
   A Data Unit that transports a report from the Short Message Service Center to the Mobile Station of the sending entity. This report describes the status of the message that was previously sent from this Mobile Station. This Data Unit can report two types of errors:

   (a) Permanent errors
   Errors that have to do with the expiration of a message, or an incompatible destination.

   (b) Temporary errors
   Errors that are temporary, eg. when the network is congested.

   An SMS-STATUS-REPORT is only sent when this is requested by the SMS-SUBMIT TPDU sent earlier.

4. **SMS-COMMAND**
   A Data Unit that transports a command from the Mobile Station to the Short Message Service Center. These commands typically deal with things such as a query about the previously submitted message, cancellation of the status report or deletion of the submitted message.

2.2.2 The Short Message Relay Layer

The Short Message Relay Layer (SM-RL) is used to transfer TPDUs and the corresponding reports, and is positioned underneath the Short Message Transfer Layer. Within this layer, each message is again identified using a *Short Message Identifier* (SMI). This SMI is not the same as in the Short Message Transfer Layer, so a mapping is made in this layer, between both SMIs. The Short Message Relay Layers on both sides of the connection communicate with each other over the Short Message Relay Protocol, a protocol that is used for the networking functions between the Mobile Station and the Short Message Service Center, and that interworks with Mobile Application Part Protocol (MAP) in the Mobile Switching Center. The protocol consists of four types of Data Units (RPDU):

1. **RP-DATA**
   A Data Unit that transports a TPDU and its control information from the Mobile Station to the network. Each RPDU contains the originating address, terminating address and user-data.

2. **RP-SM-MEMORY-AVAILABLE**
   A Data Unit that transports control information from the Mobile Station to the network. The control information that is passed, contains information on whether or not this MS has memory available to receive one or more messages.

3. **RP-ACK**
   A Data Unit that acknowledges the reception of a specific RP-DATA or RP-SM-MEMORY-AVAILABLE RPDU.

4. **RP-ERROR**
   A Data Unit that reports an error in the transmission of a specific RP-DATA RPDU.

2.2.3 The Connection Management Sublayer

The Connection Management Sublayer (CM-Sub) serves as a support layer for the Short Message Relay Layer. Each Mobile Station has two kinds of Short Message Control entities, one for Mobile Originated (MO) messaging, and one for Mobile Terminated (MT) messaging. Before any of the messages on this layer is delivered, a Mobility Management (MM) connection must first be established between the MS and the MSC. As soon as this connection is established, the RPDU is transferred over the connection. After the delivery of the RPDU has completed, the MM-connection is released by the Short Message Control, with a flag indicating whether or not the transmission was successful. The Short Message Control entities on both sides of the connection communicate with each other through the Short Message Control Protocol (SM-CP). The protocol consists of the following types of Data Units (SM-CPDU):

1. **CP-DATA**
   A Data Unit that delivers an RPDU between the Mobile Station and the Mobile Switching Center.

2. **CP-ACK**
   A Data Unit that acknowledges the reception of a specific CP-DATA SM-CPDU.

3. **CP-ERROR**
   A Data Unit that reports an error in the transmission of a specific CP-DATA SM-CPDU.

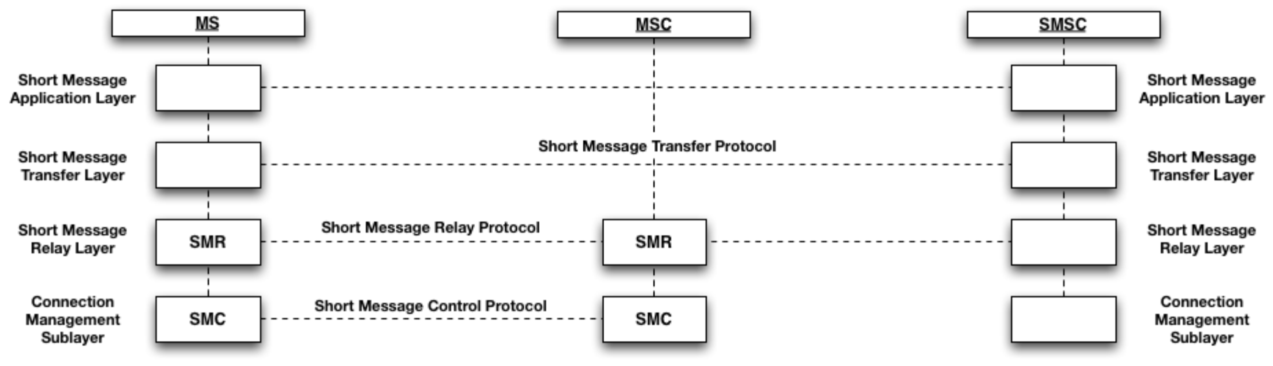
MS
MSC
SMSC
Short Message Application Layer
Short Message Application Layer
Short Message Transfer Protocol
Short Message Transfer Layer
Short Message Transfer Layer
Short Message Relay Layer
SMR
Short Message Relay Protocol
SMR
Short Message Relay Layer
Connection Management Sublayer
SMC
Short Message Control Protocol
SMC
Connection Management Sublayer

Figure 2.3: The SMS Protocol Hierarchy

2.3 A typical transmission scenario

As stated above, we will focus this thesis on the point to point Short Message Service. A typical transmission of a message over this service, consists of two parts. First, the message is sent from the Mobile Station of the sender to the Short Message Service Center. This part of the transmission is typically referred to as Mobile Originated Messaging. Then, the Short Message Service Center delivers the message to the Mobile Station of the receiver, the part that is referred to as Mobile Terminated Messaging. In this section, we will discuss both parts of the transmission scenario.

2.3.1 Mobile Originated Messaging

As stated above, Mobile Originated Messaging refers to the transmission of a message from the Mobile Station to the Short Message Service Center. A successful transmission can be broken down into the following steps, and is shown in general in figure 2.4:

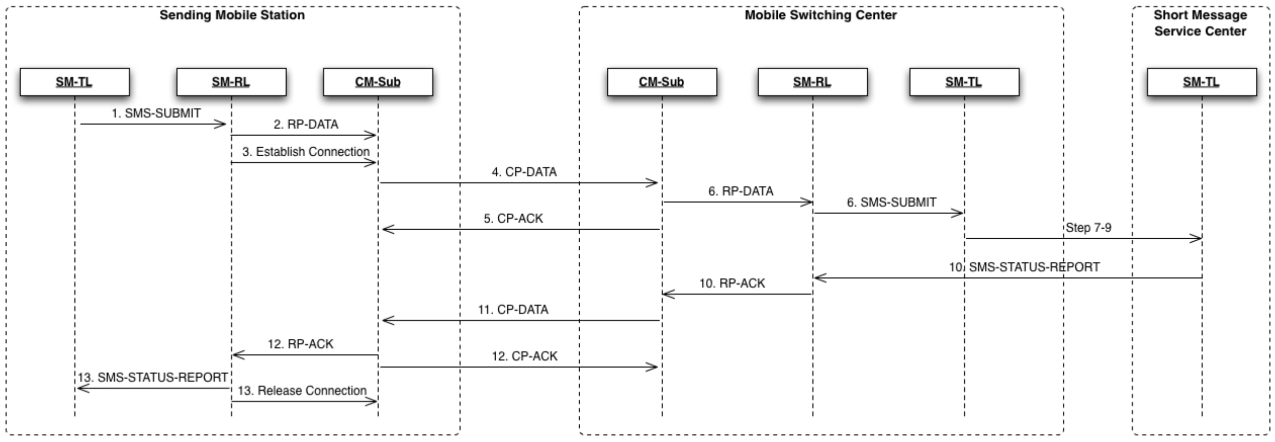

Figure 2.4: Mobile Originated Messaging

1. The Short Message Transfer Layer of the sending Mobile Station issues an SMS-SUBMIT TPDU to the Short Message Relay Layer of the sending Mobile Station.

2. The Short Message Relay Layer of the sending Mobile Station wraps the SMS-SUBMIT TPDU in an RP-DATA RPDU, and asks the Connection Management Sublayer to establish a connection to transfer the RPDU between the sending Mobile Station and the Mobile Switching Center. A timer is set and when an ACK is not received as the timer expires, the connection is aborted and an error is passed back to the Short Message Transfer Layer of the sending Mobile Station.

3. The Short Message Control establishes a Mobility Management connection and the SMS coordinating process is invoked in the Mobile Switching Center. Before processing the message, it is verified whether or not the sending Mobile Station is allowed to send messages at all. This verification can include several parameters, such as the type of subscription, whether or not all invoices were paid etc.

4. As soon as the connection to the Mobile Switching Centre is established and the sending Mobile Station is successfully verified, the RP-DATA RPDU is wrapped in a CP-DATA CPDU, and sent to the Mobile Switching Center. Again, a timer is set. If this timer expires, the CP-DATA is retransmitted at most three times. An error is reported to the SM-RL of the sending Mobile Station and the CM connection is released if the retransmission has failed all three times.

5. Upon receiving the CP-DATA message, The Mobile Switching Center acknowledges the CP-DATA message by returning a CP-ACK.

6. The SMS-SUBMIT TPDU generated in step 2 is extracted from the CPDU by forwarding the CPDU to the Short Message Relay Layer of the Mobile Switching Center and then to the Short Message Transfer Layer of the Mobile Switching Center. The Short Message Relay Layer sets a timer wherein an SMS-STATUS-REPORT is expected. If the timer expires the connection is aborted and a report indication is sent to the Short Message Transfer Layer of the Short Message Service Center.

7. After the Short Message Transfer Layer of the Mobile Switching Center has received the TPDU, the Mobile Originated Short Message Service Process is invoked in the Mobile Switching Center. This process sends a MAP-SEND-INFO-FOR-MO-SMS message to the Visitor Location Register (VLR) to request information on the location of the receiving Mobile Station.

8. The Visitor Locater Register acknowledges the request, or provides an error.
9. If the Visitor Locater Register has acknowledged the request, the Mobile Switching Center sends a MAP_FORWARD_SHORT_MESSAGE to the Interworking Mobile Switching Center, that is incorporated in the SMSC, and the Short Message is delivered to the SMSC. The SMSC returns a delivery report to the Interworking Mobile Switching Center, contained in a MAP_FORWARD_SHORT_MESSAGE_ack message. Steps 7-9 are detailed in figure 2.5.
10. The Short Message Transfer Layer entity at the SMSC sends an SMS-STATUS-REPORT to the Short Message Relay Layer of the Mobile Switching Center. This layer generates an RP-ACK message, and sends it to the Short Message Control situated at the CM-Sub layer of the Mobile Switching Center.
11. The Short Message Control situated at the CM-Sub layer of the Mobile Switching Center generates a CP-DATA CPDU and sends it to the sending Mobile Station.
12. When the Short Message Control situated at the CM-Sub layer of the sending Mobile Station receives the CP-DATA CPDU, it forwards an RP-ACK RPDU to the Short Message Relay Layer at the Mobile Station of the sender, and issues a CP-ACK to the CM-Sub layer of the Mobile Switching Center.
13. The Short Message Relay Layer of the sending Mobile Station stops the timer, and forwards the SMS-STATUS-REPORT to the Short Message Transfer Layer of the Sending Mobile Station. Then, a request is sent to release the CM connection.
14. The Connection Management connection (and thus the Mobility Management connection) between the sending Mobile Station and the Mobile Switching Center are released. When the transfer is not successful, a status report provides the cause of the error to the sending Mobile Station.

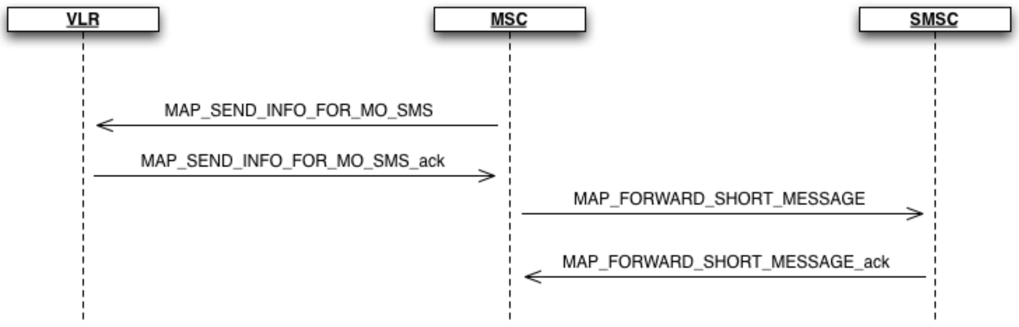
VLR
MSC
SMSC
MAP_SEND_INFO_FOR_MO_SMS
MAP_SEND_INFO_FOR_MO_SMS_ack
MAP_FORWARD_SHORT_MESSAGE
MAP_FORWARD_SHORT_MESSAGE_ack

Figure 2.5: Step 7-9 of Mobile Originated Messaging - First, the VLR is queried after which the message is forwarded to the Short Message Service Center

2.3.2 Mobile Terminated Messaging

Mobile Terminated Messaging refers to the transmission of a message from the sending Short Message Service Center to the receiving Mobile Station. A successful transmission can be broken down into the following steps, an overview is shown in 2.6:

1. The Short Message Service Center requests routing information from the Home Location Register (HLR) by sending the message MAP_SEND_ROUTING_INFO_FOR_SM. This message includes the ISDN Number of the receiving Mobile Station (MSISDN). (Note: actually, the SMSC incorporates a Gateway Mobile Switching Center (GMSC) entity for this, but for simplicity this step is left out.)

2. The HLR uses the MSISDN to look up routing information to the receiving Mobile System, and sends a MAP_SEND_ROUTING_INFO_FOR_SM_ack message to the (GMSC entity of the) SMSC. This message contains the International Mobile Subscriber Identity (IMSI) and serving Mobile Switching Center address of the recipient Mobile Station.
3. The (GMSC entity of the) SMSC delivers the message to the Mobile Switching Center to which the receiver of the message is currently subscribed. This is done using a MAP_FORWARD_SHORT_MESSAGE. The procedure is the same as the procedure described in step 9 of Mobile Originated messaging, except that the More-Messages-To-Send parameter has to be set this time.
4. The Mobile Switching Center to which the receiver of the message is currently subscribed sends a MAP_SEND_INFO_FOR_MT_SMS to the VLR, requesting the Base Station to which the receiving Mobile Station is currently subscribed.
5. The VLR initiates the paging procedure. This can result in two kinds of actions. If the VLR knows the Location Area of the receiving Mobile Station, it sends a MAP_PAGE message to this Mobile Station. Otherwise, a MAP_SEARCH_FOR_SUBSCRIBER message is sent to all Location Areas that are attached to the Mobile Switching Center, to determine the area of the receiving Mobile Station. Once this is known, the MAP_PAGE message is sent as described above.
6. The Mobile Switching Center to which the receiver of the message is currently subscribed, pages the receiving Mobile Station. If this delivery is successful, the MSC sends a MAP_PROCESS_ACCESS_REQUEST message to the VLR.
7. The VLR sends a MAP_SEND_INFO_FOR_MT_SMS_ack to the Mobile Switching Center. Now, the MSC knows it is allowed to forward the message to the receiving Mobile Station.
8. The Short Message Relay Layer of the Mobile Switching Center to which the receiver of the message is currently subscribed, generates an RP-DATA message and relays it to the receiving Mobile Station. Again, this procedure is as described in steps 1 to 6 and 10 to 14 of Mobile Originated messaging. Note that again, the Base Station Subsystem is involved.
9. After the Mobile Switching Center to which the receiver of the message is currently subscribed, receives the SMS-STATUS-REPORT from the Mobile Station, the result is included in a MAP_FORWARD_SHORT_MESSAGE_ack, to acknowledge the message that was sent in step three.
10. Now that the (GMSC entity of the) SMSC has received the acknowledgement, it determines whether or not the Mobile Station received the message or was reported absent. If the Mobile Station was absent, the Home Location Register may be queried to determine further processing actions for the receiving subscriber. No matter the status, the result is forwarded to the Short Message Service Center.

2.4 SMS Storage

As described above, once a message is received by a Mobile Station, it can be stored both in the memory of the Mobile Station or on the SIM-card. Messages that are stored on the SIM-card, cannot be read without knowing the PIN-code or PUK-code that belong to the SIM-card. It must be noted that the messages can be read though, if the SIM card is inside a Mobile Station that is turned on. Messages stored in the memory of the Mobile Station can be accessed by anyone with access to the Mobile Station, as long as the Mobile Station is not protected by some kind of access restriction mechanism. While messages were typically stored on the SIM-card, the large amount of memory available within the Mobile Stations nowadays has resulted in manufacturers storing messages in the memory of the Mobile Station by default. Therefore, an adversary that wants to read messages sent or received by a target, might try to gain possession of the Mobile Station that was used for the communication that the adversary is interested in.

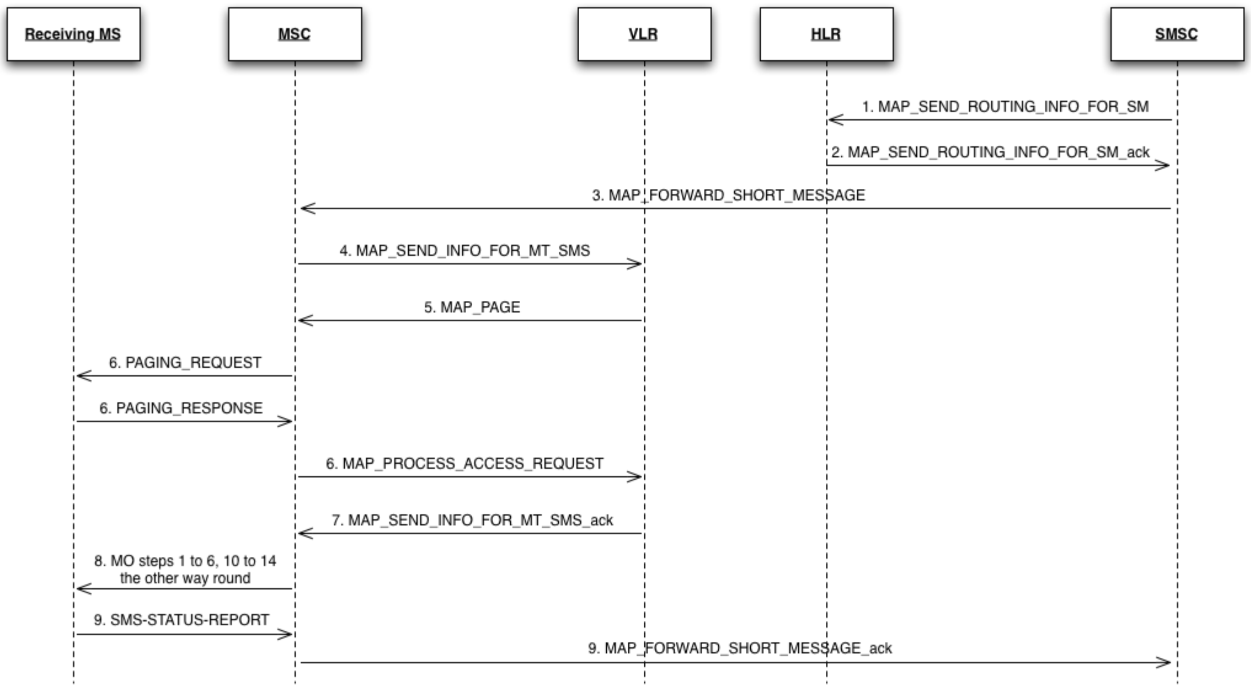
Receiving MS
MSC
VLR
HLR
SMSC
1. MAP_SEND_ROUTING_INFO_FOR_SM
2. MAP_SEND_ROUTING_INFO_FOR_SM_ack
3. MAP_FORWARD_SHORT_MESSAGE
4. MAP_SEND_INFO_FOR_MT_SMS
5. MAP_PAGE
6. PAGING_REQUEST
6. PAGING_RESPONSE
6. MAP_PROCESS_ACCESS_REQUEST
7. MAP_SEND_INFO_FOR_MT_SMS_ack
8. MO steps 1 to 6, 10 to 14 the other way round
9. SMS-STATUS-REPORT
9. MAP_FORWARD_SHORT_MESSAGE_ack

Figure 2.6: Mobile Terminated Messaging - Note that the Mobile Terminated Messaging is the same as Mobile Originated Messaging the other way round, but that additional steps have to be taken. This figure shows the additional steps between the responsible actors.

Chapter 3

Secure Communication

When communicating over a certain channel, there are several security properties one would like to ensure. When one sends a message, for example, one would typically want that this message is not read or altered during the transportation. Furthermore, when receiving confidential documents, one would want to be able to check that they actually came from the claimed sender. Finally, it is important that the sender of a document can be held responsible for it; e.g. that the document holds in court, even when the sender claims not to have sent this document.

3.1 Security properties

In security theory, all assumptions concerning secure communication are described in the four security properties: *confidentiality*, *integrity*, *authentication* and *accountability*.

Confidentiality Confidentiality means that the contents of a message sent over a certain channel can not be read by other people. In general, this property can be broken down into two dimensions: *(i)* one must not be able to read the contents of the message and *(ii)* one must not be able to know the contents of a message without reading it. To ensure the first dimension, mail is usually sent in closed envelopes. Since it is prohibited by law to open these envelopes, the contents of the message inside the envelope remain undisclosed to all persons involved in the transportation process. Furthermore, when one receives a message in an envelope that has been opened, one would know that the confidentiality of the message inside is no longer guaranteed.

While the importance of the first dimension is rather simple to understand, the second dimension is as much important, but more difficult to grasp. Suppose you were sending out an invitation to a lot of people for a wedding. To this invitation, you attach two envelopes, a green one that people can send back when they intend to come, and a red one for people that don't. Now, by simply counting the amount of green envelopes you receive, the mailman can precisely predict the amount of guests on your wedding.[1] Thus, the contents of the message inside the envelope become clear without even opening it.

Integrity While confidentiality ensures that the contents of a message are not exposed to others than the receiver, integrity ensures that the message as received by the customer has not changed during the transportation process. With traditional mail, this property is also ensured by the envelope; when one receives an envelope that has been opened, it is not guaranteed that the contents of the envelope have not been altered.[2]

Authentication When confidentiality and integrity are ensured, it is still possible that one sends a message pretending to be someone else. Therefore, a mechanism to verify the identity of the sender is of importance. Traditionally, this is done through a signature, but one could think of other mechanisms, such as a code table that both parties possess, from which codes are taken to sign messages.

[1]It is left to the fantasy of the reader to think of ways in which the mailman could abuse this information.

[2]Of course, a mechanism to make sure that the (closed) envelope in which the message arrives is the same as the one in which it was sent is also necessary for this to hold.

Accountability Accountability is about tying the responsibility for a message to the claimed sender. Again, the signature is the traditional way of ensuring this property; when faced with a trial over a certain dispute, a document that contains the signature of the alleged, in which the allegations in this dispute are confirmed, will be hard not to take responsibility for by this alleged.

3.2 Cryptography

When designing a secure system, confidentiality, integrity, authentication and accountability can be ensured by using cryptography. While cryptography is often referred to as the art of hiding information or *the art and science of codes and ciphers*[10], the authors of [22] give a broader definition, that explicitly covers all of these properties:

Definition 3.1 ***Cryptography*** *is the study of mathematical techniques, related to aspects of information security, such as confidentiality, data integrity, entity authentication and data origin authentication.*

Cryptography has been around for more than 2000 years; In order to protect his messages from being read as well as understood by messengers or enemies, Julius Caesar transpositioned all letters in the alphabet, resulting in a message with garbled, incomprehensible text. By applying the transposition backwards, receivers of these messages were able to recover the original text, revealing the message Caesar had meant to send them. However, recovering the original message turned out to be rather simple, by using techniques such as frequency and pattern analysis. Nevertheless, the fundamental idea started a long and ongoing tradition of cryptographers designing cryptography systems and cryptanalysists trying to break them.

While cryptography is often applied to secure systems, it has turned out to be hard to apply it correctly. In the following sections, we will discuss the several approaches to cryptography, and some protocols that may be of use for our goal: providing secure communication through SMS.

3.2.1 General definitions

Before we can start an in-depth discussion on cryptography, we have to pin down some general definitions, that will be used throughout the following sections:

Definition 3.2 ***Plaintext*** *is the term we use when we refer to text on which no cryptography is applied yet. Therefore, exposure of the plaintext will reveal all information it contains.*

Definition 3.3 ***Ciphertext*** *is the term we use when we refer to text on which cryptography is applied. When cryptography is applied correctly, exposure of ciphertext will not lead to a revealing of information.*

Definition 3.4 ***Encryption*** *is the process of applying cryptography to plaintext, hiding all information inside this text and thus resulting in ciphertext.*

Definition 3.5 ***Decryption*** *is the process of reverting cryptography applied earlier, transforming ciphertext back into plaintext.*

In general, encryption is done through an encryption algorithm, which is often referred to as a *cipher*, that works under the control of a secret *key*.

$$c = e_{k_e}(m)$$

with

c - the ciphertext

e - the encryption algorithm or cipher

k_e - the key that is used to control the encryption algorithm

m - the message, often referred to as the plaintext

The cipher can be seen as a lock: while the functioning of a lock is common knowledge, a specific lock only opens under the influence of the key corresponding to that lock. When this key gets compromised, e.g. copied by an unauthorised party, the lock will no longer satisfy its goal since it can now be opened by this unauthorised party. Therefore, it is of vital importance to guard the secrecy of the key. Since there is only a small set of encryption algorithms, once confronted with a key an unauthorised party will easily be able to try this key with all of these algorithms to revert the ciphertext into plaintext.

Of course, the receiver of a ciphertext that is encrypted using a specific algorithm must be able to recover the message hidden by this ciphertext. This is done through decryption.

In general, decryption is done through a decryption algorithm d, corresponding to the algorithm e that was used for encryption.

$$m = d_{k_d}(c)$$

with

c - the ciphertext

d - the decryption algorithm or cipher

k_d - the key that is used to control the decryption algorithm

m - the message, often referred to as the plaintext

Again, it is important that the key k_d is not compromised.

3.2.2 Symmetric cryptography

With symmetric cryptography, the same key is used to control either the encryption or decryption algorithm. Thus

$$k_e \equiv k_d$$

When applying symmetric cryptography, it is important that the key is never compromised, since an exposure of the key will eventually lead to exposure of the encrypted information. Therefore, safe key distribution is a hard, if not impossible problem one has to solve when utilising symmetric cryptography.

3.2.3 Asymmetric cryptography

With asymmetric cryptography, different keys are used for encryption and decryption. Of course, there has to be some relation between k_e and k_d in order to revert the ciphertext to the original plaintext. The relation between both keys depends on the algorithm, but is based on a proposal written by Diffie and Hellman, in which they state that a category of mathematical problems that are (too) difficult to solve can be utilised to solve the problem of key distribution. The problems that the current cryptographic algorithms make use of, are all based on discrete logarithms, factoring or elliptic curves.

In an environment where asymmetric cryptography is applied, all users have both a private key and a public key. Each user must publish its public key, so that other parties can look it up. Note that this does not compromise the security of the system. Throughout this thesis, we will denote the public and private key of a user Alice as follows:

k_{E_A} - The public key of Alice

k_{D_A} - The private key of Alice

When Alice wants to send a message m to Bob, she first looks up the public key k_{E_B} of Bob. She then computes

$$c = e_{k_{E_B}}(m)$$

resulting in the ciphertext c. Upon receiving c, Bob can compute

$$m = d_{k_{D_B}}(m)$$

resulting in the plaintext again.

Chapter 4

Unsecured SMS

In the previous chapters, we have discussed SMS and Secure Communication in detail. In this chapter we will first discuss the vulnerabilities that we have identified in the current implementation of SMS, and then the threats that the system is exposed to as a result of these vulnerabilities.

4.1 Vulnerabilities in the current SMS implementation

4.1.1 Lack of security services within the network

Due to its design, the GSM Network cannot not provide several important security services [18]. The first of these security services that are not provided by GSM is *mutual authentication.* Mutual authentication forces two communicating parties to authenticate each other, so that both parties have verified the identity of the other peer. SMS provides no form of mutual authentication; a message is simply sent to an address (i.e. a telephone number). There is no way to force that this message can only be read by the person that the sender believes to belong to this number, and thus, the receiver of the message is never authenticated.

The second security service that is not provided by SMS is *end-to-end security.* With end-to-end security applied, data that is sent between two communicating parties is encrypted and/or signed from the moment it leaves the sending party, up until the message is received by the other party. While a message might be encrypted during over-the-air transfer, it will be decrypted as soon as it enters the network after arriving at the BSS and encrypted again when it leaves the network through the BSS that delivers the message to the sender, thus temporarily exposing the cleartext.

The third security service that is not provided by SMS is *non-repudiation.* Non-repudiation encompasses the accountability of the sender for the messages he has sent and is enforced using several techniques including hashing and encrypting of the data that is sent. SMS does not implement such a mechanism.

Furthermore, *the anonymity of the user* is not guaranteed within the SMS infrastructure. While a Mobile Station is identified within the network by a random number called the Temporary Mobile Subscriber Identity (TMSI), the establishment of this number involves communication of the International Mobile Subscriber Identity (IMSI) between the Mobile Station and the network [15]. The TMSI is only valid for a specific area, and must be regenerated when a Mobile Station enters such an area, allowing an eavesdropper to link an IMSI to the TMSI that identifies the Mobile Station within that region.

Another weakness of the network lies in the authentication of Mobile Stations and the Base Station Subsystem. While a Mobile Station must authenticate itself to the Base Tranceiver Station, which acts as a gateway to the Base Station Subsystem, the BTS does not authenticate itself to the Mobile Station. Thus, apart from a lack of mutual authentication between the sender and the receiver, there is also a lack of mutual authentication between parts of the system that are responsible for the delivery of the message. As we will see in section 4.2.3, the lack of mutual authentication between the BTS and the MS allows for BTS spoofing, exposing the data that is sent by a user to whoever owns a Base Transceiver Station.

4.1.2 Transfer of messages from the Mobile Station to the BSS

As described in chapter 2, messages between the Mobile Station and the Base Transceiver Station of the Base Station Subsystem are transferred over the air. During this transfer, these messages can be intercepted by anyone with sufficient equipment, that is freely available online for approximately $ 850,- [1]. While these messages are not necessarily encrypted, most implementations in Europe use A5/1 for encryption, while other networks use A5/2, a weaker version of the algorithm. Mobile stations typically support several or even all of the algorithms in the A5-family and the algorithm that is used depends on the provider.

Encryption with the A5-family of algorithms works as follows: as soon as the Mobile Station is first seen by the network (i.e. the phone is turned on) it must authenticate itself to the network. To achieve this authentication, the network sends a 128-bit random number $RAND$ to the Mobile Station. The Mobile Station calculates the response $SRES_{MS} = A3(K_i, RAND)$, with A3 an algorithm that is implemented both in the SIM card of the user and at the network and K_i a symmetric key that is stored on the SIM card and known at the HLR, and sends it back to the network. The network compares $SRES_{MS}$ with the value $SRES_N$ it has calculated itself and if both values match the Mobile Station is authenticated. Otherwise the protocol ends. If the Mobile Station is authenticated, both the network and the Mobile Station compute the session key $K_c = A8(K_i, RAND)$, with A8 an algorithm that is implemented both in the SIM card of the user and at the network. With the session key established, both parties can compute key material, using the A5 algorithm, to encrypt the data they transmit between each other. An overview of the authentication is given in figure 4.1.

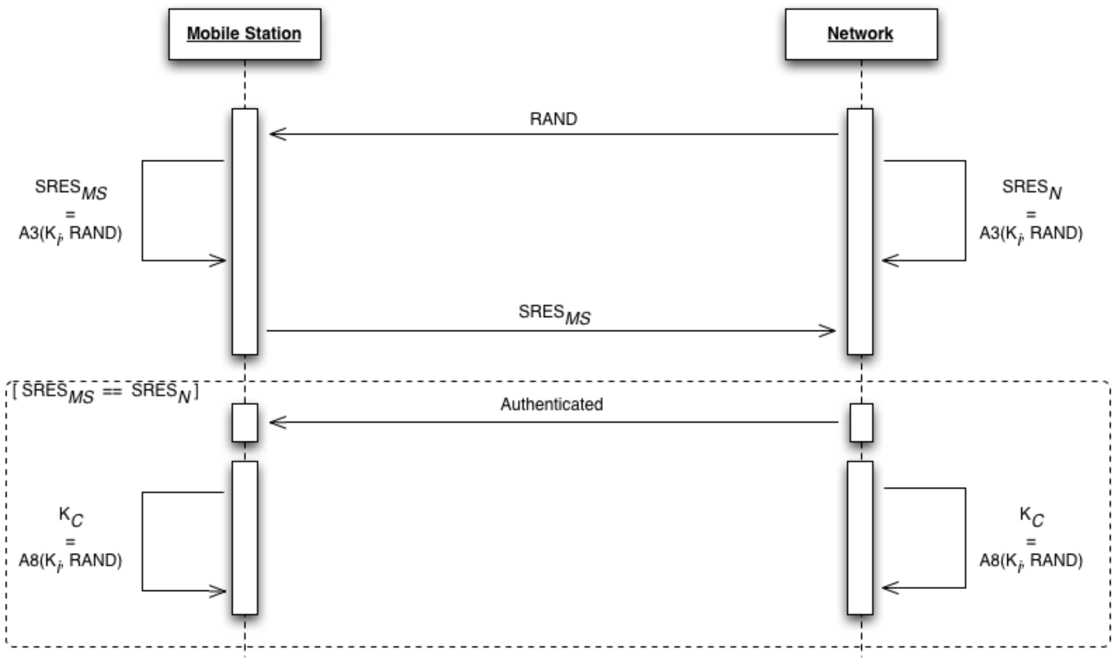

Figure 4.1: Authentication within the A5 protocol

In [12], researchers have shown that both A5/1 and A5/2 do not provide sufficient security. Weaknesses in the algorithms come down to the fact that each transaction between the Mobile Station and the network contains several parts of data that are known in advance. Combining this knowledge with the ciphertext and a precomputed rainbow table, the session key K_c can be reconstructed from several milliseconds of data that is transferred during a conversation, thus exposing further traffic. The rainbow table has been computed by a community donating their computing power and published through torrents at the end of 2009 [2]. While there are still some practical issues such as frequency hopping that make capturing calls difficult, these issues do not hold for SMS [11].

To solve the weaknesses identified in A5/1 and A5/2, a new encryption algorithm A5/3 has been developed. While it is far from being implemented at a large scale due to the high costs of the operation, vulnerabilities have already been found [16].

While the problems described above hold for GSM traffic in general, they do not hold for areas where the new generation of mobile traffic, UMTS, is implemented since the traffic over UMTS is secured using other algorithms. Since this thesis focuses on the vulnerabilities within the GSM workflow, we have not investigated possible security flaws of UMTS. The availability of UMTS on a large scale in countries such as The Netherlands, is another reason not to implement A5/3 in these countries, since communication over UMTS is already more secure.

4.1.3 Unencrypted message passing within the GSM network

During the transmission process of both Mobile Originated and Mobile Terminated messaging, SMS messages are transferred from a Mobile Switching Center (MSC) to the Short Message Service Center (SMSC) and then again to a Mobile Switching Center. For this transfer, either MAP protocol of the SS7 protocol suite is implemented. MAP is an unencrypted protocol, allowing (employees at) the provider to read SMS messages [15]. Since SMS does not implement any signatures over its messages, the integrity of a message is also not guaranteed. Thus, it is also possible for (employees at) the provider to modify the messages without the receiver knowing.

4.1.4 Vulnerabilities of the Mobile Station

Finally, we have also identified some vulnerabilities of the Mobile Station. The most important vulnerability encompasses the unencrypted storage of sent and received SMS messages, on either the SIM card or the memory of the phone. While storage on the SIM card comes with some access restriction in the form of the PIN-code, messages that are stored in the memory of the phone can be accessed easily, using an arbitrary SIM card. Furthermore, modern phones tend to have batteries with a very high capacity. While this is a good development in terms of standby-time and usage time, it also undermines the PIN protection since phones will be typically turned on for extended periods of time.

4.2 Threats

As a result of the vulnerabilities presented in the previous section, we identify the following threats:

- Compromised security due to a stolen or lost mobile station
- SMS Spoofing
- BTS Spoofing
- Interception/modification of messages during transmission over the air
- Replay of messages
- Loss of messages within the network

In this section, we will discuss each of these threats in detail. To facilitate an easy overview of all threats, they will be presented in table 4.1 at the end of this chapter.

4.2.1 Compromised security due to a stolen or lost mobile station

When a mobile station is lost or stolen, messages stored on that mobile station may be compromised in several ways. First, some or even all messages that were sent and received using the mobile station may be stored within the memory of the mobile station. If the mobile station is not protected by a password, these messages can be read by anyone without requiring the PIN code that is used to protect the data on the SIM card, thus compromising confidentiality. While modern phones typically provide password functionality to protect the data stored within memory, older models do not provide such functionality. Furthermore, many people disable password protection in favour of usability.

Another option is that the messages are stored on the SIM card of the subscriber. In this case, the messages are protected by the PIN code, that is required to unlock the SIM card. However, this PIN code is only asked for when the phone is turned on. In many situations in which a phone is lost or stolen, there will remain plenty of time to read or forward these messages before the phone runs out of charge, again compromising confidentiality. This especially holds with the large capacity that batteries nowadays carry. Protection is then again only guaranteed if password protection is implemented by the manufacturer *and* configured by the user.

Finally, messages can also be sent by anyone pretending to be the legitimate sender, once a phone has been lost or stolen and is not protected by a password, again for as long as the phone holds charge, compromising both accountability and authentication.

4.2.2 SMS Spoofing

By applying SMS Spoofing, one can use an arbitrary Mobile Station or a computer to inject SMS Messages into the system with the originator set to a specific Mobile Station [21]. Injecting the messages is possible if the Short Message Service Center (SMSC) does not require that the originating address is equal to the actual sender of the message, and is done by sending a CP-DATA message from a Mobile Station to a MSC as seen in step 4 in figure 2.4, or by submitting a message to the SMSC using a computer connected to that SMSC. While mechanisms to detect spoofed messages are defined in the specification [9], an implementation of these mechanisms is not mandatory and cannot be relied on. The technique can be applied by an attacker Eve to send messages to Bob, that Bob will believe to be from Alice, thus compromising accountability. On the internet, several implementations that allow users to send messages with a spoofed originator can be found. Among these implementations are websites, but also open source programs in the form of Java MIDlets or native applications that are to be run on a cellphone [23]. In combination with BTS Spoofing, this technique can also be used to execute a man-in-the-middle attack on a transmission of SMS messages between two peers, at very little or no costs.

4.2.3 BTS Spoofing

Since a Base Transceiver Station does not authenticate itself to the Mobile Station, as described in the previous section, anyone possessing the necessary equipment can operate its own BTS, for approximately \$ 10.000 [7], to which a target mobile station will automatically subscribe as soon as the BTS is the one closest to that target. Furthermore, since the BTS decides whether or not a Mobile Station should encrypt messages sent to that BTS, the BTS can simply ask for unencrypted messages and thus eavesdrop all SMS sent by an arbitrary subscriber, compromising confidentiality. Mobile Stations might implement software that triggers an alarm when a BTS requests data to be transferred unencrypted[1] but since A5/1 and A5/2 are considered broken as discussed in the previous section, the BTS might as well ask for encrypted messages to avoid this alarm to be triggered. As mentioned above, combined with SMS Spoofing intercepted messages may even be forwarded to the recipient, also compromising confidentiality, authentication and accountability. However, the attack attempt will be known sooner or later since the target will not be able to make or receive calls, and sent messages will not be billed by the provider. To avoid this, the spoofed BTS can be combined with a Mobile Station that forwards all intercepted (and modified) traffic to the GSM network, at the costs of the attacker.

4.2.4 Interception/modification of messages during transmission over the air

In the previous section, we have already discussed the weaknesses in the A5-cryptography family as discovered by [12, 16]. In [11], Barkan et al. describe several practical attacks on the GSM network, eventually leading to a man-in-the-middle attack. They claim that *since the attacker is in the middle, he can eavesdrop, change the conversation, perform call theft, etc. The attack applies to all the traffic including short message service (SMS).* Their attacks comprise of a pre-computation phase of just a few hours after which the encryption key that is used for communication between the Mobile Station and the network can be found in less than a second, requiring only ciphertext to be known to the attacker, provided that the network uses the A5/2 algorithm for encryption. If the network uses the stronger A5/1 or A5/3 algorithm, however, an attacker could use a fake Base Transceiver Station coupled to a Mobile Station to communicate with a target Mobile Station using the A5/2 algorithm and forward the (modified) data to the network using the stronger A5/1 algorithm through the Mobile Station that is

[1]The Sony Ericsson P1i for example, has implemented such a warning.

coupled to the base transceiver station. Even without this equipment, passive eavesdropping on A5/1 encrypted conversations is considered feasible.

The attacks described by Barkan et al. [11] allow an attacker to:

1. Eavesdrop on communication between the Mobile Station and the network, compromising confidentiality.
2. Modify messages that are sent between the Mobile Station and the network, compromising integrity, authentication and accountability.

	Confidentiality	**Integrity**	**Authentication**	**Accountability**	**Costs**
Stolen or lost mobile station	x		x	x	$ 0,-
SMS Spoofing				x	$ 0,-
BTS Spoofing	x	x	x	x	$ 10.000,-
Interception during OTA transmission	x				$ 850,-
Modification during OTA transmission[2]	x	x	x	x	$ 1.300,-
Replay of messages[3]					$ 1.300,-
Loss of messages within the network[4]					$ 0,-

Table 4.1: Threats within the SMS workflow, the costs for an attack and the security properties that they compromise

4.2.5 Replay of messages

Since an arbitrary person can intercept SMS transmission from a Mobile Station to the Base Transceiver Station, messages can also be replayed. While this may not be of interest in the typical SMS scenario, the technology is also used for Machine to Machine communication (M2M). M2M allows two machines to communicate over SMS. For example, two traffic lights on either side of a single road might communicate over SMS about their current state (red or green light on). If the manufacturer of these devices has not implemented measures to detect replayed messages, which is likely since he might think this is implemented in the SMS protocol itself, a replay of a single message might result in a disaster, for example a situation where both of the traffic lights mentioned above change to green at the same time. While this threat is clearly identifiable, it does not relate to the security properties described in 3, since it encompasses the implementation of SMS in a system, rather than single messages itself.

[1]Confidentiality is again compromised in this case since modification implies exposure, whether the attacker is interested in the contents of the message or not.

[2]While this threat is clearly identifiable, it does not relate to any of the security properties since it encompasses the implementation of SMS in a system rather than single messages itself.

[3]This is an issue that may be of importance in case of a dispute between two parties, but does not relate to any of the security properties.

4.2.6 Loss of messages within the network

Due to the best-effort characteristic of SMS, messages that are sent may never arrive at the receiver. While the system allows for an acknowledgement to be sent to the sender of the message upon successful reception of the message by the receiver, this is only done on request and is not implemented on all phones. The iPhone for example, does not support requesting delivery reports. Therefore, messages sent with a phone that does not support the request of delivery reports or is not set up to request these reports might never arrive at the receiver without the sender knowing. While this is an issue that may be of importance in case of a dispute between two parties, it does not relate to any of the security properties defined in 3.

Chapter 5

Secured SMS

Now that we have described how SMS works, discussed the security properties that we want to guarantee and defined the threats to these security properties as a result of the design and implementation of SMS, we can start our effort to enable secure communication over the channel. To achieve secure communication, we want to guarantee confidentiality, integrity, authentication and non-repudiation. In this chapter, we will first discuss general solutions to guarantee the several security properties, independent of specific choices. Then, we will look at some existing implementations we have found, and discuss the flaws and missing features in these implementations.

As a solution for the flaws and missing features in the existing implementations, we will propose two designs for secure SMS, in which we make specific choices for algorithms that we think to be suitable to guarantee the security properties sufficiently, and discuss which of the threats defined in chapter 4 they address. After this, we will have a look at some existing implementations we have found, to investigate whether or not they address the threats sufficiently. An overview of the threats addressed by both the existing implementations and our proposals can be found in table 5.5 on page 31.

It should be noted that, while the security properties can be guaranteed by implementing changes at several levels, we have chosen to propose a design that can be used given the current implementation of SMS. Thus, our design requires no (expensive) changes to the hardware that is in place momentarily, but will focus on a solution that can be implemented with minimal expenses using software.

5.1 Guaranteeing the security properties

In the following section, we will give some general solutions to achieve confidentiality, integrity, authentication and accountability. In the discussion of the solutions, we will not make specific choices for algorithms, key sizes etc., since we will discuss these aspects of the solutions in our proposals.

5.1.1 Confidentiality

As discussed in chapter 3, confidentiality means that a message sent over a certain channel cannot be read by people other than the receiver and has two dimensions: *(i)* one must not be able to read the contents of the message and *(ii)* one must not be able to know the contents of a message without reading it. The first of these dimensions can be achieved by encryption. However, encrypting two messages that have the exact same content, using the same algorithm and key for both of the messages results in equal ciphertexts. Thus, while an adversary may not be able to read the message he may well be able to (i) gain knowledge from the way the messages look and (ii) use contextual knowledge to launch an attack on the ciphertext in order to retrieve the key that was used to encrypt the message. To solve this problem, we can add data to the message that is not likely to occur twice, such as a (large) random number or a timestamp. When using a suitable algorithm, doing so will result in completely different ciphertext each time a message is encrypted, even if the cleartext remains the same.

5.1.2 Integrity

Integrity ensures that the message as received by the receiving party has not changed during the transportation process. When we send a message, we can include a fingerprint of that message to achieve integrity. Such a fingerprint can be made by applying a hash function to the message. A hash function is a one-way function f that, given a message m, creates a string of fixed length h of that message, for which the following properties hold:

1. It is computationally easy to compute $f(m) = h$
2. It is computationally infeasible to compute $f^{-1}(h) = m$
3. Given a message m with $f(m) = h$, it is infeasible to find a message m' such that $f(m') = h$
4. A change of one bit in m leads to a completely different h

Thus, applying a hash function to a message m gives us a string h that uniquely identifies that message. Now, if we send h alongside the message, the receiving party can apply the same hash function to the decrypted message m' and check that $f(m') = h$. From this and property 3 above follows that $m' = m$ and thus that the message has not been tampered with. Of course, the string h should also be sent encrypted to avoid replacing or tampering with it by an adversary.

5.1.3 Authentication

When authentication is guaranteed, both parties involved in a communication know that the other party really is who he/she claims to be. Thus, if Alice and Bob tend to communicate with each other over a channel that provides authentication, it is impossible for Trudy to:

1. Send a message to Bob pretending to be Alice, or a message to Alice pretending to be Bob
2. Receive a message from Bob while Bob thinks it is received by Alice, or a message from Alice while Alice thinks it is received by Bob

When symmetric key encryption is used, authentication can easily be established as long as the key remains secret. Likewise, several asymmetric encryption schemes bring about authentication possibilities [24].

5.1.4 Accountability

When accountability is guaranteed, the sender of a message cannot later dispute that he has sent that message. Accountability can be achieved by using the fingerprint of the message, combined with encryption. If for example Alice sends message m to Bob, alongside a fingerprint of the message h, both encrypted with their shared key k, it is easy to prove that Alice has sent message m to Bob: only Alice is able to encrypt the fingerprint h in such a way that Bob can decrypt it again to reveal h again. Furthermore, if the fingerprint h matches with the fingerprint h' Bob can calculate for himself, the message m as received by Bob is the same as the message sent by Alice due to the integrity property provided by fingerprints as discussed earlier.

5.2 Existing implementations

5.2.1 Message in a Bottle

Message in a Bottle is a commercial application, that *"allows to protect your SMS communications from intrusions and unauthorized access"* [5]. The application is implemented in Java, runs on a large variety of mobile phones and can be downloaded free of charge by a user. After downloading the application, a user can purchase credits, which will then be used to send messages securely. While the implementation of Message in a Bottle is closed source, a user manual is available, from which we were able to derive some information about the implementation.

Message in a Bottle allows users to encrypt messages that are sent through SMS, using asymmetric elliptic curve cryptography. The public/private key pair of a user is established during installation of the software. Furthermore, messages can be signed in order to achieve integrity and accountability. Both the sending and receiving parties must install the software to enable secure messaging. Messages that were sent or received using Message in a Bottle, as well as the keys of all contacts, are stored securely within the phone, protected by a PIN-code. Thus, the following threats, as defined in chapter 4 are addressed by Message in a Bottle:

- Compromised security due to a stolen or lost Mobile Station
- Interceptions/modification of messages during transmission over the air

Due to the closed nature of the implementation, we were not able to verify the correctness of the solution Message in a Bottle delivers, and therefore have to trust the developer of the application.

5.2.2 CryptoSMS

CryptoSMS [4] is an open source application implemented in Java, that runs on a large variety of mobile phones. The application can be downloaded and used free of charge by any user that has a phone that supports the Java APIs that are used to implement the software[1].

CryptoSMS allows users to encrypt their messages using asymmetric elliptic curve cryptography. Again, the public/private key pair of a user is established during installation, and both the sending and receiving parties must install the software to enable secure messaging. Messages that were sent and received using CryptoSMS, as well as the keys of all contacts, are stored within the memory of the phone, encrypted using a passphrase as asymmetric key. Thus, the following threats, as defined in chapter 4 are addressed by CryptoSMS:

- Compromised security due to a stolen or lost Mobile Station
- Interception of messages during transmission over the air

Note that CryptoSMS does not address modification of messages during transmission over the air, since the implementation does not contain any form of signatures over messages. Therefore, the integrity of a message is never guaranteed. In the latest version of CryptoSMS, signatures over keys are implemented, providing some form of authentication.

Another weak point we have found in the implementation of CryptoSMS is related to the first threat it addresses. To enhance the user experience, the passphrase of a user is stored within the phone for a certain time period after the user has entered it. However, the private key of the user, as well as all messages the user has sent/received are stored in the memory of the phone, encrypted using a symmetric algorithm with the passphrase as a key. This exposes both the private key as well as all messages during the time period in which the passphrase is stored within the phone. Therefore, the compromised security due to a stolen or lost Mobile Station is only addressed if the Mobile Station is exposed to an adversary outside this time period. Furthermore, using a passphrase that is typically relatively short if it depends on the user provides only weak symmetric encryption.

5.2.3 Other implementations

Besides the two implementations discussed above, we have found three other solutions that claim to protect SMS Messaging:

- SpiderSMS
- Kryptext
- MultiTasker

However, all of these solutions were commercial, and none of them provided any insight in their functioning. Therefore, we were unable to verify which of the threats identified in chapter 4 they address, and whether or not they do this correctly.

[1]Being the MIDP 2.0 architecture

5.2.4 Conclusions on existing implementations

While several solutions are available to protect SMS, we were able to verify only two of these solutions using information that is freely available. Both Message in a Bottle and CryptoSMS address only some of the threats we have identified in chapter 4leaving us no other option than conclude that both solutions are insufficient. We think that this shows once again that securing SMS is an important subject; while several applications claim to protect communication over this channel, in reality only some of the threats are addressed. Therefore, in the following section we will propose two protocols for secure SMS, that address all of the threats except for one. We think these proposals can serve as a good starting point for an implementation of truly secure SMS Messaging.

5.3 Our proposals

In this section, we will propose two protocols to secure SMS communication between two parties. When designing these proposals, we were faced with two challenges, the first being the computational power available on mobile phones. While recent smartphones certainly have enough computational power available to perform the computations necessary for cryptography fast enough, these computations must remain within reasonable bounds on older phones. Furthermore, securing the communication comes with overhead in terms of the number of bytes, and thus the number of messages, that are transferred between the two parties. Given the fact that SMS messages are paid for per message, this overhead must also remain within reasonable bounds, especially for small amounts of concatenated messages[2]. We want to gain a sufficient level of security, while keeping the overhead for each message that is sent at only one message. An in-depth discussion on what level of security is considered sufficient can be found in [19]. For both proposals, we will say something about the security level provided by that protocol.

The protocols that we discuss comprise their transfer from the moment they leave the telephone up until they reach their destination. No matter which protocol is used, messages that are sent and received must be stored within the mobile station in their encrypted form. The messages will only be decrypted after the user has provided a password, that is stored within the phone securely.

5.3.1 Securing SMS using a Public Key infrastructure

The first protocol to secure SMS that we propose is the following: all users that wish to communicate securely generate a key pair consisting of a private key k_D and a public key k_E. The private key k_D is stored securely within the phone, eventually protected by a password, while the public key can be published in several ways. Furthermore, both users agree on an encryption algorithm a and a hash function f that both parties use.

Sending messages between Alice and Bob

Now say Alice wants to send a message to Bob. To do this, she sends the following over the unsecured channel:

- A → B: $e_{k_{E_B}}(m,\, t,\, d_{k_{D_A}}(h(m,t)))$
- with:

$e_{k_{E_B}}$	The encryption algorithm, parameterised with the public key E of Bob
m	The message
t	A timestamp
$d_{k_{D_A}}$	The decryption algorithm, parameterised with the private key k_{D_A} of Alice
$h(m,\, t)$	A hash h of m and t calculated with $f(m,\, t)$

Upon receiving this message, Bob can decrypt it to reveal both m and t, as well as $d_{k_{D_A}}(h(m,t))$. Using the same hashing algorithm f as Alice, he computes $h' = f(m,t)$. He then computes $h =$

[2]Remember from chapter 2 that we refer to multiple messages that are sent by one user to another user at the same time as concatenated messages.

$e_{k_{E_A}}(d_{k_{D_A}}(h(m,t)))$, using the property that the algorithm d that Alice used combined with the algorithm e that he uses satisfies both $d_{k_{D_A}}(e_{k_{E_A}}(x)) = x$ and $e_{k_{E_A}}(d_{k_{D_A}}(x)) = x$. Bob now knows that

1. The message m has not been tampered with since $h = h'$
2. The message has been sent by Alice, since only Alice is able to compute $d_{k_{D_A}}(h(m,t))$ so that $e_{k_{E_A}}(d_{k_{D_A}}(h(m,t))) = h$

Note that the last property only holds as long as the private key of Alice is not compromised. Thus, confidentiality, integrity and authentication are guaranteed. Furthermore, if Bob stores $d_{k_{D_A}}(h)$ accountability is also guaranteed *as long as Alice does not change her key pair* since, given a dispute, Bob can easily apply $e_{k_{E_A}}(d_{k_{D_A}}(h))$ to reveal h, proving that Alice herself created the fingerprint of the message she sent and use $h' = f(m,t)$ again to prove that this was the same as the message that he received, and that it was sent at time t.

Correctness of the protocol

Given the goals of a security protocol, it is important to say something about its correctness and robustness. The protocol as described above has many similarities with the protocol used with PGP, the only differences being that PGP does not use timestamps, and only uses the algorithm a in combination with the public key of the receiver to encrypt a message key, that can be used by the receiver to decrypt the message. With PGP, this is done due to performance reasons. However, since the short nature of SMS messages, whether concatenated or not, these performance issues are not of importance for our proposal. For an extensive discussion of PGP, see Tanenbaum [24]. PGP has proven to be robust over the years, and no feasible attacks on PGP are known as of today. Therefore, we found that a proof of the correctness of this protocol using a proof assistant was not necessary.

Implementing the protocol

We need the following to implement this protocol:

k_{D_u} A private key for each user u

k_{E_u} A public key for each user u

e An encryption algorithm

d A decryption algorithm

f A hash function

Commutativity The combination of algorithms e and d must satisfy both $d_{k_{D_A}}(e_{k_{E_A}}(x)) = x$ and $e_{k_{E_A}}(d_{k_{D_A}}(x)) = x$

Secure storage To store the private key k_{D_u} of a user u on the Mobile Station

Central archive Of public keys k_{E_u} used by a user u, to facilitate accountability

On the generation of the public and private key, a lot can be found in literature. An implementing party could use one of the algorithms described in [17] for example. After generation, the public keys can be communicated over several channels[3]. As an algorithm combination of e and d, one can opt for RSA or the Diffie Hellman protocol. We propose choosing one algorithm, that is used by all users, in order to keep key management feasible. RSA is a safe choice, since it is the industry standard for asymmetric encryption. Furthermore, on most mobile platforms, secure storage is available nowadays, for example through the Java MIDP architecture. The central archive can be arranged in the form of a key server. An important problem with this is that all users of the system have to trust the party that implements this archive. Such a party may be difficult, if not impossible to find.

[3]For example on the website of the user, or automatically using bluetooth during a 'real' meeting

Since an SMS message always consists of one or more messages of 140 bytes, it is important that we look at the message overhead that an implementation of this protocol leads to. As a hash algorithm, we have chosen SHA-256, which provides us with a security level of 128 bits, which is sufficient until at least 2050 according to [19]. With respect to the algorithm a, nowadays a key length of 2048 bits is seen as the minimum length to provide sufficient security when using such algorithms. Due to the design of these algorithms, the length of the ciphertext is always a multiple of the key length.[4]

The total length of data in the message that is sent between the communicating parties has a length of 1504 bits. The calculation of this length is detailed in table 5.1. Since the $1504 \leq 2048$, the ciphertext will be 2048 bits in size at least, resulting in the transfer of two messages worth of data when a single message is sent using this protocol. The overhead in terms of the number of messages can be calculated accurately for different key sizes using the following formula:

$$\frac{ceil(\frac{(n*1120)+384}{|k|})* \mid k \mid}{1120} \tag{5.1}$$

with n the number of concatenated messages and $\mid k \mid$ the keysize. In table 5.2, we have listed the overhead for several amounts of messages and keysizes.

Entity	**Bit length**	**Remark**
Message	1120 bits	$140 * 8 = 1120$
Time stamp	128 bits	Of the form YYYYMMDDHHMMSSMsMs
Hash	256 bits	Using SHA-256
Total	1504 bits	

Table 5.1: Calculation of the message overhead for one message when using the Public Key protocol to secure SMS

Plaintext	**Ciphertext** 2048 bits key	**Ciphertext** 4096 bits key	**Ciphertext** 6800 bits key
1	1,83	3,66	6,07
2	3,66	3,66	6,07
3	3,66	3,66	6,07
4	5,49	7,31	6,07
5	5,49	7,31	6,07
10	10,97	10,97	12,14
15	16,46	18,29	18,21
20	21,94	21,94	24,29
50	51,20	51,20	54,64
100	100,57	102,40	103,21

Table 5.2: An overview of message overhead using several keysizes. The table shows the number of messages that are required for the ciphertext, given a number of plaintext messages and the keysize.

In the protocol, we have used timestamps instead of nonces to give each message a different appearance. The reason we have used timestamps is that they have a context that can be used to keep the list of timestamps a user has already seen at minimal length; a receiving user should only remember timestamps that fall within the time window the provider uses to keep a message at the Short Message Service Center.

Threats addressed by the protocol

Implementing this protocol addresses the following threats defined in chapter 4:

[4]It must be noted that the security level provided by a 2048 bits key is well below the 128 bits security level we have chosen for our hashing algorithm. From a practical viewpoint, a 2048 bits key protects the data of the user up until 2022 [19]. According to the formula defined in [19], a key size of 6800 bits is required to get a security level of 128 bits. Given the values in table 5.2, we consider a 4096 bits key as the current maximum when one looks at the costs in terms of the message overhead. However, a key of such length will likely lead to bad performance on older cellphones.

- **Compromised security due to a stolen or lost mobile station**
  By storing the messages that were sent and received in their encrypted form, a lost or mobile station will not compromise security, since the messages cannot be read by the adversary that gained access to the telephone without knowing the password.
- **SMS Spoofing**
  SMS Spoofing by a third party is no longer possible in this protocol, since the party will not know the private key of the sender, and thus cannot compute $D_{sender}(h(m, t))$.
- **BTS Spoofing**
  While BTS Spoofing is of course still possible, an adversary will gain nothing by doing so, since it is impossible to deduce the contents of the message or modify the message to mislead the receiver.
- **Interception/modification of messages during transmission over the air**
  Again, it is still possible to intercept messages during transmission over the air. An adversary will gain nothing by doing so, however, since it is impossible to deduce the contents of the message or modify the message to mislead the receiver. It must be noted, however, that a communication between two parties can still be disturbed by intercepting and not forwarding messages.
- **Replay of messages**
  The use of timestamps within the protocol enables the receiver to easily maintain a list of messages he has already seen, taking out the possibility to replay a message.

5.3.2 Securing SMS using a symmetric key

Another protocol we propose uses symmetric keys instead of a public key infrastructure. In this protocol, two users Alice and Bob that wish to communicate securely generate a symmetric key k_{AB} for their communication. The key is stored securely within the phone of both parties. Furthermore, they agree on an encryption algorithm e and decryption algorithm d, both parameterised with the key k_{AB} and a hash function h.

Sending Messages between Alice and Bob

Now, when Alice wants to send a message to Bob, she sends the following message over the unsecured channel:

- A $\rightarrow$ B: $(A,\ e_{k_{AB}}(m,\ t,\ h(m,\ t,\ A,\ B)))$
- with:

A	The identity of Alice
B	The identity of Bob
k_{AB}	The symmetric key Alice and Bob have established
m	The message
t	A timestamp
$h(m,\ t,\ A,\ B)$	A hash h of m, t, A and B calculated with $f(m,\ t,\ A,\ B)$

Upon receiving the message, Bob can now deduce m, t and $h(m,t,A,B)$ from this message using a decryption function d parameterised with the key k_{AB}, and compute h' by concatenating A and B to m and t. Now, if $h' = h$ Bob knows that

1. Alice created the message, to be sent to him
2. The message has not been tampered with, since a third party can never create $e_{k_{AB}}(m,t,h(m,t,A,B))$

Correctness of the protocol

To prove the correctness of the protocol, we have used the cryptographic protocol verifier ProVerif, which will be introduced in more detail in appendix A. In ProVerif, we have modelled the symmetric protocol, using the pi-calculus. This model has subsequently been checked by ProVerif for several security aspects. The following aspects have been checked:

1. Confidentiality of m
   By asking ProVerif whether or not the attacker can get knowledge of m, using

   ```
   query attacker : m.
   ```

2. Integrity of m
   This has been done in two ways. First, we have asked ProVerif whether or not the attacker can gain access to $h(m, t, A, B)$ using

   ```
   query attacker : hash(FourTuple(m,t,A,B)).
   ```

   If the attacker could gain access to the hash, he could also change it, tricking the receiver into believing that the integrity of the message, of which he could get no knowledge as a result of 1, is compromised. Furthermore, we have modelled the check on integrity done by the receiver to result in distinct events on success (`receivedMessage`) and failure (`sentMessage`). Then, we have asked ProVerif whether or not each report of success is a result of a send operation, using

   ```
   query evinj : receivedMessage(w,x,y) ==> evinj : sentMessage(w,x,y).
   ```

   Using this query, ProVerif determines that for each receivedMessage event that occurs, exactly one sentMessage event occurred on beforehand. By doing so, we prove that a message can only be received by B if that message has been sent by A. From this, the fact that an attacker has modifying the message as the only option left follows, which is in turn not possible since he cannot gain access to m and h.

3. Authentication of A and B
   In 2, we have already seen that a message can only be received by B if it has been sent by A. This, combined with the check on the hash, makes that A and B are authenticated.

After running ProVerif with the protocol as modelled in appendix B, the tool confirmed the confidentiality of m, but said a replay attack was still possible, since it did not understand that for each message, the timestamp t is checked against earlier received messages, and accepted only once. It turned out to be difficult, if not impossible, to implement the state of the protocol into ProVerif, so we had to come up with a workaround for this problem. As a workaround, we have changed the protocol, as can be seen in appendix C. The change encompasses the definition of the timestamp. Where at first, the timestamp was introduced by the sending party, we have now modelled the protocol using a timestamp that is shared and synchronised between the sender and the receiver. Upon receiving a message, the receiver checks whether or not the timestamp in that message equals the timestamp he shares with the sender (i.e. if the message has arrived at the receiver in 0 time). Now, if the attacker wants to replay a message, he does so at another time (i.e. he shares no timestamp with the receiver). Therefore, messages that are replayed by an attacker will be rejected. An output trace of the models can be found in appendix A

Thus, we have proved the correctness of the symmetric protocol using ProVerif. The only property that does not hold for the symmetric protocol is accountability, since Bob can also generate $h(m, t, A, B)$ himself when in a dispute with Alice, claiming that Alice has send m at time t.

Implementing the protocol

We need the following to implement this proposal:

$k_{u_1 u_2}$ A private key for each pair of users (u_1, u_2)

e A cryptographic algorithm parameterised with $k_{u_1 u_2}$ used for encryption

d A cryptographic algorithm parameterised with $k_{u_1 u_2}$ used for decryption

f A hash function

u_1 A way to express the identity of u_1

Entity	**Bit length**	**Remark**
Message	1120 bits	$140 * 8 = 1120$
Time stamp	128 bits	Of the form YYYYMMDDHHMMSSMsMs
Hash	256 bits	Using SHA-256
Subtotal	1536 bits	$(1120 + 256 + 128)$
Identity sender	88 bits	Represented as the phone number
Total	1624 bits	

Table 5.3: Calculation of the message overhead for one message when using the symmetric key protocol to secure SMS

u_2 A way to express the identity of u_2

For the establishment of a symmetric key, we refer again to [17] since it contains several algorithms that facilitate secure symmetric key establishment between two parties. For now, note that it is important that both parties meet each other in real life to do this completely secure and that in chapter 6, we will shortly discuss a solution for this problem. From then on, secure communication is possible as long as the key remains secret. Therefore, it is important that this key is stored securely within the phones. We repeat here that such storage is available on most mobile platforms nowadays. As cryptographic algorithms e and d, we propose AES since it is the standard defined by NIST. Up until now, no feasible attack on AES has been found. If such an attack is ever found, implementing parties can not be blamed for using the standard cryptographic algorithm. For AES, we propose a key size of 256 bits to get a security level of 128 bits[5], and operation in cipher-block chaining mode, to ensure that a change in the plaintext propagates throughout the ciphertext. As a hash function, we again propose SHA-256 to match up with the security level of 128 bits provided by AES. The identity of u_1 and u_2 can easily be expressed using their phone numbers including the country code.

Again, it is important that we look at the message overhead that an implementation of this protocol leads to since an SMS message always consists of a multiple of 140 bytes. An example calculation, for one message can be found in table 5.3. The overhead in terms of the number of messages can be calculated accurately using the following formula:

$$\frac{88 + (ceil(\frac{1120n+384}{128}) * 128)}{1120} \tag{5.2}$$

with n the number of messages and a block size of 128 bits. In table 5.4, we have listed the overhead for several amounts of messages.

We have again used timestamps instead of nonces, for reasons that are explained in our discussion of the public key protocol.

Plaintext messages	**Ciphertext messages**
1	1,45
2	2,48
3	3,51
4	4,42
5	5,45
10	10,48
15	15,51
20	20,42
50	50,48
100	100,42

Table 5.4: An overview of message overhead when using the symmetric protocol.

[5]Which is again enough to keep our data secure until at least 2050.

Threats addressed by the protocol

Implementing this protocol addresses the following threats defined in chapter 4:

- **Compromised security due to a stolen or lost mobile station**
  By storing the messages that were sent and received in their encrypted form, a lost or mobile station will not compromise security, since the messages cannot be read by the adversary that gained access to the telephone without knowing the password.

- **SMS Spoofing**
  SMS Spoofing by a third party is no longer possible in this protocol, since the party will not be able to compute $K_{u_1 u_2}(m, t, h(m, t, A, B))$.

- **BTS Spoofing**
  While BTS Spoofing is of course still possible, an adversary will gain nothing by doing so, since it is impossible to deduce the contents of the message or modify the message to mislead the receiver.

- **Interception/modification of messages during transmission over the air**
  Again, it is still possible to intercept messages during transmission over the air. An adversary will gain nothing by doing so, however, since it is impossible to deduce the contents of the message or modify the message to misled the receiver. It must be noted however, that a communication between two parties can still be disturbed by intercepting and not forwarding messages.

- **Replay of messages**
  The use of timestamps within the protocol enables the receiver to easily maintain a list of messages he has already seen, taking out the possibility to replay a message.

	Message in a Bottle	**CryptoSMS**	**Proposal 1 - Asymmetric**	**Proposal 2 - Symmetric**
Stolen or lost mobile station	x	x	x	x
SMS Spoofing			x	x
BTS Spoofing			x	x
Interception during OTA transmission	x	x	x	x
Modification during OTA transmission	x		x	x
Replay of messages			x	x
Loss of messages within the network				

Table 5.5: An overview of implementations to secure SMS, and the threats that they address marked with an x.

5.4 Feasibility of the proposals on a mobile platform

Due to limited resources on a mobile platform, the question remains whether or not the computation of the cryptographic algorithms is feasible on all platforms. While we do have some ideas to investigate this, a precise answer to this question is something we cannot give within the scope of this thesis due to its complex nature. To be able to give an indication, table 5.7 gives an overview of several processors that have been implemented in a wide variety of mobile phones over the course of the years. For each processor, we have listed its computational power, and the number of instructions the processor executes per second in millions (MIPS). Furthermore, we have found benchmarks of a C-library of cryptographic algorithms performed on an Intel Pentium 4 at a clock speed of 2.93 GHz [3]. This processor executes 7926 million instructions per second [6]. Due to the nature of the library, it might also be used to implement our proposals on platforms that support programming in C, such as the iPhone and mobile phones that run the Symbian operating system, and thus allows us to estimate the execution time. Platforms that do not support C, such as Android which only supports Java development, are likely to perform slower.

From the benchmark data, we have derived an estimation of the execution time of RSA, AES and SHA-256 on one message on the several processors listed in table 5.7. In the benchmark, encryption and decryption times for RSA are given in seconds per operation on a block as big as the keysize. For SHA-256 and AES-256, the execution time is expressed in mebibytes[6] per second. Thus, our estimations are derived using the following calculations:

$$RSA_{Encryption}(Processor) = \frac{MIPS(Pentium4)}{MIPS(Processor)} * RSA_{Encryption}(Pentium4) \tag{5.3}$$

$$RSA_{Decryption}(Processor) = \frac{MIPS(Pentium4)}{MIPS(Processor)} * RSA_{Decryption}(Pentium4) \tag{5.4}$$

$$SHA_{256}(Processor) = \frac{MIPS(Pentium4)}{MIPS(Processor)} * \frac{1424}{(SHA_{256}(Pentium4) * 8.388.608)} \tag{5.5}$$

$$AES_{256}(Processor) = \frac{MIPS(Pentium4)}{MIPS(Processor)} * \frac{1536}{(AES_{256}(Pentium4) * 8.388.608)} \tag{5.6}$$

Note that for RSA, we have only calculated the time it takes to generate the hash in the symmetric protocol[7], since this is the worst case. Thus, hashing in asymmetric protocol will be faster. For AES, the total length of what must be encrypted is 1536 bits, as can be seen in table 5.3. The estimation of the execution time can be found in table 5.6. From this table, we conclude that an implementation of all algorithms can be considered feasible on processors released after 2004, as a result of the short time it takes to perform the necessary operations on a message, even when using the ARM-926 processor. Of course, processors that are state of art nowadays perform much better, but the performance of the ARM-926 is good enough for a practical implementation.

Processor	RSA Encryption	RSA Decryption	SHA-256	AES-256
ARM-926	7,93 ms	379,37 ms	0,0497 ms	0,0879 ms
ARM-PXA27X	2,17 ms	104,33 ms	0,0137 ms	0,0242 ms
ARM-1136J(F)-S	2,36 ms	112,78 ms	0,0148 ms	0,0261 ms
ARM-PXA3XX	1,74 ms	83,46 ms	0,0109 ms	0,0193 ms
ARM-Cortex A8	0,87 ms	41,73 ms	0,0055 ms	0,0097 ms

Table 5.6: An estimation of the execution time of the cryptographic operations for a single message, on the processors found in several Mobile Phones as listed in table 5.7

[6]1 mebibyte equals 1.048.576 bytes
[7]Being a hash over m, t, A and B, together having a total length of 1424 bits.

Processor	**Mhz**	**MIPS**	**Year**	**Phones**
ARM-926	220	220	2004	Sony Ericsson K and W Series, Siemens/Benq X65, LG Arena
ARM-PXA27X	624	800	2004	HTC Universal, Motorola Q, Motorola A728, Motorola A780, Motorola A910, Motorola A1200, Motorola E680, Motorola E680i, Motorola E680g, Motorola E690, Motorola E895, Motorola Rokr E2, Motorola Rokr E6
ARM-1136J(F)-S	532-665	740	2006	Zune, Nokia E90, Nokia N93, Nokia N95, Nokia N82, Nokia N800, Nokia N810, Nokia E63, Nokia E71, Nokia 5800, Nokia E51, Nokia 6700 Classic, Nokia 6120 Classic, Nokia 6210 Navigator, Nokia 6220 Classic, Nokia 6290, Nokia 6710 Navigator, Nokia 6720 Classic, Nokia E75, Nokia N97, Nokia N81, HTC TyTN II, HTC Dream, HTC Magic, HTC Hero, HTC Nike
ARM-PXA3XX	1250	1000	2008	Samsung Omnia
ARM-Cortex A8	600	2000	2009	iPhone 3GS, Motorola Droid, Palm Pre, Nokia N900, Sony Ericsson Satio

Table 5.7: An overview of processors, their speed in Mhz, the number of instructions they can execute per second in millions (MIPS), some of the phones in which they are implemented, and the year the processor was first implemented in a phone.

Chapter 6

Conclusions

In this thesis, we have first shown the vulnerabilities within the current SMS implementation. We have seen that a lack of security services within the network, weakly encrypted transfer of messages from the Mobile Station to the Base Station Subsystem, unencrypted message passing within the network and vulnerabilities within the Mobile Station result in several threats that make the implementation insecure. These threats include interception or modification during transmission over the air, the possibility to spoof a message or even a Base Transceiver Station, the replay of messages and compromised security due to a stolen or lost Mobile Station. Furthermore, we have seen that messages may get lost as a result of the best-effort character of SMS.

From this, it follows that to make sure that the four security properties - Confidentiality, Integrity, Authentication and Accountability - hold while communicating over SMS, several measures need to be taken. Therefore, we have taken a look at several existing implementations that claim to secure SMS communication. Only two of these implementations - Message in a Bottle and CryptoSMS - turned out to be verifiable, both containing some flaws and missing key features, while on others no information could be found at all. Therefore, we set out to propose our own protocol to secure SMS communication, with two design goals in mind; (i) The protocol had to be feasible on a mobile platform and (ii) The overhead in terms of number of messages per sent message as a result of an implementation of the protocol had to be as small as possible. From this design goals, two proposals were derived; one using asymmetric cryptography, the other using symmetric cryptography.

After proposing the protocols, we have shown that they are both feasible, correct and address all threats defined in chapter 4 apart from the loss of messages within the network. While the symmetric protocol comes with less overhead in terms of the amount of messages, a higher security level, especially when we look at the small amount of messages that is typical for SMS usage, it has the disadvantage that key distribution is much harder; two users that wish to communicate preferably meet in person to exchange the symmetric key necessary for their communication.

The asymmetric key protocol, on the other hand, has a lower security level when we keep the overhead in terms of the number of messages within reasonable bounds. Also, asymmetric key encryption is likely to be slower than asymmetric encryption [17]. However, key distribution is much easier since a party can simply publish its public key to facilitate secure messaging by any other party.

With the advantages and disadvantages of both protocols in mind, it is hard to prefer one of them over the other. For most people, the security level provided by the asymmetric protocol will be enough, making this protocol the preferred implementation due to its ease of key distribution. One could also argue however, that the symmetric protocol will be preferred due to its lower overhead costs. For situations in which the data that is sent must remain confidential after 2022, the security level of 2048 bits provided by the asymmetric protocol is certainly not high enough. Thus, the symmetric protocol may be the preferred solution, while one might also argue that the costs of overhead are not of much importance in these situations, which is again in favour of the asymmetric protocol.

Therefore, we propose an implementation that supports both protocols, and leaves the user the choice; if the user has exchanged a symmetric key with a certain person, this person can be communicated with using the (cheaper and more secure) symmetric protocol, while other persons can be contacted securely using the asymmetric protocol. Preferably, the key size used in the asymmetric protocol is also left for the user to decide[1]. Implementing both protocols comes with an additional benefit: the asymmetric protocol can be used to establish a symmetric key between two users, that can then be used for further communication. This enables two users to communicate using the symmetric protocol without ever having met the other in real life.

This leaves us with the problem of maintaining a central archive, containing all public keys a party within the system has ever used, in order to provide accountability. While this problem is hard to solve, it does not render the asymmetric protocol useless since all other security properties are guaranteed. Apart from this consideration, we can think of several solutions for the problem. Just as with PGP, several key servers might be maintained by independent organisations. A party could then query at least two of these servers, to see if the keys on both servers match. Such servers should of course be implemented on a worldwide scale, to facilitate international messaging. Likewise, the archive could also be maintained by organisations people tend to trust, such as the government. An archive maintained by such an organisation would certainly be satisfactory in a trial over a certain dispute. Note that we have not included any costs for querying the server in our calculations of overhead for the asymmetric protocol, since (i) it does not involve the transfer of the message and (ii) the server may be queried in several ways that come with no additional costs.

With the problems around accountability in the asymmetric protocol out of the way, implementing both protocols leaves users the choice whether or not they want to enforce the accountability property as well. In that case, they should use the asymmetric protocol to communicate. Another interesting result of implementing both protocols is that the symmetric protocol may be enhanced, using the public/private keypair of a user to include a signature over a timestamp in the message that is encrypted with the symmetric key to provide accountability.

By proposing the two protocols, we have shown that it is possible to instantiate security measures upon the current implementation to facilitate secure messaging. No action by the providers is necessary for the protocols to work, allowing users to take matters into their own hands. To improve our proposals, more research should be done into the implementation of both symmetric and asymmetric encryption algorithms on a mobile platform, as to not exclude Mobile Stations that were produced before 2004. Furthermore, we welcome anyone to implement our proposals on a variety of platforms, and perform some benchmarks of the protocols on each platform. Another way to accurately say something about the performance of our proposals on a specific processor would be to calculate the exact amount of operations that each cryptographic algorithm requires, and investigate the number of instructions for each operation on the processor. Then, an accurate estimation of the execution time of the algorithm can easily be derived. Both an implementation of the proposals and an accurate estimation of the execution time of the several algorithms were beyond the scope of this thesis, however.

[1]Note that this implies that each user should have keys of several lengths, since the receiver determines the length of the key that is used, while the sender typically pays for the costs. Managing keys of several length may be a big concern in the practical context.

Appendix A

A short introduction to ProVerif

ProVerif is a protocol verifier, that is used to prove that certain security properties hold for a cryptographic protocol. To do so, the protocol is modelled in an extension of the pi-calculus. In this appendix, we will give a short overview of the structure of such a model. For an extensive discussion of ProVerif, we refer to [13].

A model starts with a list of declarations, after which a process definition follows. The declarations define the protocol, the entities that play a role within the protocol and their security aspects. The process describes what happens between the entities within the protocol, and is used by ProVerif to simulate message passing within the protocol.

A listing of our original model is given in figure B. Everything between `(*` and `*)` is not interpreted by ProVerif, as this is the syntax for comments. Our model starts with a description of the symmetric protocol in comments. Furthermore, we have used the setting `param traceDisplay = long` to get verbose output in the cases where ProVerif found a trace that exposed some vulnerability in the protocol.

At the beginning of the protocol, we first declare the free variables within the protocol. Free variables within ProVerif are variables to which anyone has access, and of which anyone can have knowledge. Thus, we have declared the channel over which the SMS messages pass as well as the identities of the sender and receiver as free variables.

In the following part of the model, we define functions for operations within the protocol, using the keyword `fun`. The names for these functions can be chosen freely, words such as `encrypt` and `hash` are no keywords within ProVerif, and thus will be attached no meaning to by the tool. For the two-way functions, we also define functions that describe how to unpack the result of the application of an earlier defined function again, using the keyword `reduc`. Note that for `hash`, we have not defined such a function since `hash` is a one-way function.

Now that we have declared the free variables and functions for operations within the protocol, we have to tell ProVerif what we want to know about the protocol. This is done using the keyword `query`. After this, we model the behaviour of both the sender and the receiver. Finally, we indicate what processes ProVerif can initiate, using the keyword `process`. Here, the use of `!` is important: it indicates what can be executed in parallel by ProVerif.

As mentioned in section 5.3.2, executing this model led to the following results:

```
RESULT evinj:receivedMessage(w_30,x_31,y_32) ==> evinj:sentMessage(w_30,x_31,y_32)
is false.
RESULT (but ev:receivedMessage(w_170,x_171,y_172) ==> ev:sentMessage(w_170,x_171,y_172)
is true.)
RESULT not attacker:hash(FourTuple(m_20[!2 = v_373,!1 = v_374],
t_21[!2 = v_375,!1 = v_376],A[],B[])) is true.
RESULT not attacker:m_20[!2 = v_486,!1 = v_487] is true.
```

Studying the traces given by ProVerif learned us, that ProVerif thought a replay attack would be possible, since we haven't modelled the fact that the receiver check for each message it receives, whether or not a message with this timestamp was received earlier. It turned out that modelling such a mechanism in ProVerif would be too difficult, if not impossible. Therefore, we decided to change the model, as can be seen in figure C. The difference between the protocols is that, instead of running the sending and receiving process independently from each other, in parallel, now only a combination of the sending and receiving process can be run in parallel, with a shared timestamp that holds only for this instance of the process. Thus, a replayed message can easily be distinguished from a real message, by comparing the timestamp of the message to the timestamp in the process between the sender and the receiver. If these do not match, the receiver can discard the message as being a replayed message.

While this may not be perfect, we think that this behaviour is still a reasonable representation of the 'real-world' situation: a timestamp may be used only once between a sender and a receiver. Running ProVerif on this model yields the following result:

```
RESULT evinj:receivedMessage(w_30,x_31,y_32) ==> evinj:sentMessage(w_30,x_31,y_32)
is true.
RESULT not attacker:hash(FourTuple(m_21[!1 = v_149],t_20[!1 = v_150],A[],B[]))
is true.
RESULT not attacker:m_21[!1 = v_239] is true.
```

Appendix B

The original protocol model

```
(*
        Verification of Symmetric Protocol

        kab      : The symmetric key established between A and B
        m        : The message A wants to send to B
        t        : A timestamp to be sent with the messages (freshness)
        h        : A hash of (m,t,A,B)

        A begins sentMessage(A,B,m)
        A -> B : A, {| (m,t), h |}kab
        B ends receivedMessage(A,B,m)

*)
(* param traceDisplay = long. *)

(* Declaration of free variables *)
(* The unsecure channel over which the messages are sent *)
free SmsChannel.
(* Identifiers of A and B *)
free A, B.

(* Declaration of constructors/functions *)
(* Encryption of M with K *)
fun encrypt/2.
(* A hash function over M *)
fun hash/1.
(* Construct a two-tuple *)
fun TwoTuple/2.
(* Construct a three-tuple *)
fun ThreeTuple/3.
(* Construct a four-tuple *)
fun FourTuple/4.

(* Declaration of deconstructors *)
(* Decryption of encrypt(M,K) with K leads to M *)
reduc decrypt(encrypt(m,k),k) = m.
(* Unpack two-tuple *)
reduc firstOf2Tuple(TwoTuple(a,b)) = a.
reduc secondOf2Tuple(TwoTuple(a,b)) = b.
(* Unpack three-tuple *)
```

```
reduc firstOf3Tuple(ThreeTuple(a,b,c)) = a.
reduc secondOf3Tuple(ThreeTuple(a,b,c)) = b.
reduc thirdOf3Tuple(ThreeTuple(a,b,c)) = c.

(* Queries / what we want to know about the protocol *)
(* Whether or not the attacker can derive the message m : Confidentiality *)
query attacker : m.
(* Whether or not the attacker can derive the hash h *)
(* Integrity (In combination with above) *)
query attacker : hash(FourTuple(m,t,A,B)).
(* A message can only be received AFTER that message was sent *)
query evinj : receivedMessage(w,x,y) ==> evinj : sentMessage(w,x,y).

(* Sending process *)
let sender =
(* Sender generates message m, timestamp t*)
new m;
new t;
(* Sender fires event that he sends the message *)
event sentMessage(A,B,m);
(* Sender sends tuple over SmsChannel *)
out(SmsChannel,
    TwoTuple(A,encrypt(ThreeTuple(m,t,hash(FourTuple(m,t,A,B))),secretKey))).

(* Receiving process *)
let receiver =
in(SmsChannel,receivedPacket);
(* Receiver finds identity of sender to determine key to use *)
let identitySender = firstOf2Tuple(receivedPacket) in
let encryptedMessage = secondOf2Tuple(receivedPacket) in
(* Receiver decrypts message with the key he shares with sender *)
let decryptedMessage = decrypt(encryptedMessage,secretKey) in
(* Receiver unpacks tuple to reveal m,t,h *)
let receivedMessage = firstOf3Tuple(decryptedMessage) in
let receivedTimestamp = secondOf3Tuple(decryptedMessage) in
let originalHash = thirdOf3Tuple(decryptedMessage) in
(* Receiver generates h' from (m,t,h,A,B) *)
let newHash = hash(FourTuple(receivedMessage,receivedTimestamp,identitySender,B)) in
(* If h' == h => Accept message *)
if originalHash=newHash then event receivedMessage(identitySender,B,receivedMessage)
(* If h' <> h => Reject message, it cannot be trusted *)
else event rejectedMessage(identitySender,B,receivedMessage).

process
new secretKey;
!receiver |
!sender
```

Appendix C

The modified protocol model

```
(*
        Verification of Symmetric Protocol

        kab     : The symmetric key established between A and B
        m       : The message A wants to send to B
        t       : A timestamp to be sent with the messages (freshness)
        h       : A hash of (m,t,A,B)

        A begins sentMessage(A,B,m)
        A -> B : A, {| (m,t), h |}kab
        B ends receivedMessage(A,B,m)

*)
(* param traceDisplay = long. *)

(* Declaration of free variables *)
(* The unsecure channel over which the messages are sent *)
free SmsChannel.
(* Identifiers of A and B *)
free A, B.

(* Declaration of constructors/functions *)
(* Encryption of M with K *)
fun encrypt/2.
(* A hash function over M *)
fun hash/1.
(* Construct a two-tuple *)
fun TwoTuple/2.
(* Construct a three-tuple *)
fun ThreeTuple/3.
(* Construct a four-tuple *)
fun FourTuple/4.

(* Declaration of deconstructors *)
(* Decryption of encrypt(M,K) with K leads to M *)
reduc decrypt(encrypt(m,k),k) = m.
(* Unpack two-tuple *)
reduc firstOf2Tuple(TwoTuple(a,b)) = a.
reduc secondOf2Tuple(TwoTuple(a,b)) = b.
(* Unpack three-tuple *)
```

```
reduc firstOf3Tuple(ThreeTuple(a,b,c)) = a.
reduc secondOf3Tuple(ThreeTuple(a,b,c)) = b.
reduc thirdOf3Tuple(ThreeTuple(a,b,c)) = c.

(* Queries / what we want to know about the protocol *)
(* Whether or not the attacker can derive the message m : Confidentiality *)
query attacker : m.
(* Whether or not the attacker can derive the hash h *)
(* Integrity (In combination with above) *)
query attacker : hash(FourTuple(m,t,A,B)).
(* A message can only be received AFTER that message was sent *)
query evinj : receivedMessage(w,x,y) ==> evinj : sentMessage(w,x,y).

(* Sending process *)
let sender =
(* Sender generates message m *)
new m;
(* Sender fires event that he sends the message *)
event sentMessage(A,B,m);
(* Sender sends tuple over SmsChannel *)
out(SmsChannel,
    TwoTuple(A,encrypt(ThreeTuple(m,t,hash(FourTuple(m,t,A,B))),secretKey))).


(* Receiving process *)
let receiver =
in(SmsChannel,receivedPacket);
(* Receiver finds identity of sender to determine key to use *)
let identitySender = firstOf2Tuple(receivedPacket) in
let encryptedMessage = secondOf2Tuple(receivedPacket) in
(* Receiver decrypts message with the key he shares with sender *)
let decryptedMessage = decrypt(encryptedMessage,secretKey) in
(* Receiver unpacks tuple to reveal m,t,h *)
let receivedMessage = firstOf3Tuple(decryptedMessage) in
let receivedTimestamp = secondOf3Tuple(decryptedMessage) in
let originalHash = thirdOf3Tuple(decryptedMessage) in
(* Receiver generates h' from (m,t,h,A,B) *)
let newHash = hash(FourTuple(receivedMessage,receivedTimestamp,identitySender,B)) in
(* If h' == h => Accept message *)
if originalHash=newHash then
(* If h' == h and timestamps match => Accept message *)
if receivedTimestamp=t then event receivedMessage(identitySender,B,receivedMessage)
(* If timestamps don't match -> this is a replay, deny *)
else event rejectedMessage(identitySender,B,receivedMessage)
(* If h' <> h => Reject message, it cannot be trusted *)
else event rejectedMessage(identitySender,B,receivedMessage).

process
new secretKey;
!(new t; (receiver | sender))
```

Bibliography

[1] http://www.ettus.com/. [Online; accessed 05-May-2010].

[2] http://reflextor.com/torrents/. [Online; accessed 05-May-2010].

[3] Benchmarks of several cryptographic algorithms executed using the Crypto++ library. http://www.cryptopp.com/benchmarks-p4.html. [Online; accessed 18-June-2010].

[4] CryptoSMS. http://www.cryptosms.org/. [Online; accessed 01-June-2010].

[5] Message in a Bottle, user manual. http://www.ugosweb.com/miabo/MIABOUserManual.pdf. [Online; accessed 01-June-2010].

[6] Pentium 4 specification. http://www.cpu-world.com/CPUs/Pentium_4/Intel-Pentium%204%20515-515J%202.93%20GHz%20-%20JM80547PE0771M.html. [Online; accessed 18-June-2010].

[7] The GSM Standard (An overview of its security). http://www.sans.org/reading_room/whitepapers/telephone/gsm-standard-an-overview-security_317/. [Online; accessed 24-May-2010].

[8] ISO/IEC 21989 - Information technology — Telecommunications and information exchange between systems — Private Integrated Services Network — Specification, functional model and information flows — Short message service. pages 1–46, Feb 2002.

[9] ETSI TS 123 040 - Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); Technical realization of Short Message Service (SMS) (3GPP TS 23.040 version 9.1.0 Release 9). pages 1–204, Jan 2010.

[10] R. Anderson. Security engineering: A guide to building dependable distributed systems. *portal.acm.org*, Jan 2008.

[11] E. Barkan, E. Biham, and N. Keller. Instant ciphertext-only cryptanalysis of gsm encrypted communication. *Journal of Cryptology*, Jan 2008.

[12] A. Biryukov, A. Shamir, and D. Wagner. Real time cryptanalysis of a5/1 on a pc. *Lecture Notes in Computer Science*, pages 1–18, 2001.

[13] B. Blanchet. ProVerif Automatic Cryptographic Protocol Verifier User Manual. http://www.proverif.ens.fr/proverif-manual.pdf. [Online; accessed 20-June-2010].

[14] T. Clements. SMS – Short but Sweet. http://developers.sun.com/mobility/midp/articles/sms/, 2003. [Online; accessed 13-May-2009].

[15] N. Croft and M. Olivier. Using an approximated one-time pad to secure short messaging service (sms). *Proceedings of the Southern African Telecommunication Networks and Applications Conference (SATNAC) 2005*, pages 71–76, 2005.

[16] O. Dunkelman, N. Keller, and A. Shamir. A practical-time attack on the a5/3 cryptosystem used in third generation gsm telephony.

[17] N. Ferguson and B. Schneier. *Practical Cryptography.* 2003.

[18] O. Kolsi and T. Virtanen. Midp 2.0 security enhancements. *Proceedings of the 37th Hawaii International Conference on System Sciences*, pages 287–294, 2004.

[19] A. Lenstra and E. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, 2001.

[20] Y. Lin. GSM Point-to-Point Short Message Service. *International Journal of Wireless Information Networks*, 4(4):249–256, 1997.

[21] S. Lord. Trouble at the Telco: When GSM Goes Bad. *Network Security*, 2003(1):10–12, 2003.

[22] A. Menezes, P. V. Oorschot, and S. Vanstone. Handbook of applied cryptography. *books.google.com*, Jan 1996.

[23] T. Sannum. SMS Spoof. `http://www.waste.org/~terje/palm/SMSspoof/`, 2003. [Online; accessed 14-April-2010].

[24] A. Tanenbaum. *Computer Networks*. 2002.