Application Level Security Enforcement Mechanisms for Advanced Network Services

by

Trond Ivar Johansen
Olav B. Lømsland

Thesis in partial fulfilment of the degree of
Master in Technology in
Information and Communication Technology

Agder University College
Faculty of Engineering and Science

Grimstad
Norway

May 2006

Summary

Today the telecom world and the Internet world are converging. Ericsson has foreseen this convergence and developed a prototype of a service creation and execution environment called ServiceFrame. ServiceFrame is an extension of the ActorFrame framework. ActorFrame features new concepts described in UML 2.0, such as connectors, ports, parts and behaviour inheritance and structured classes. ActorFrame has central components called actors and agents. Actors and agents are modelled and described using the UML 2.0 notation. In ActorFrame and ServiceFrame actors and agents are communicating asynchronously using messages and concurrent state machines. The ServiceFrame developers have always concentrated on making ServiceFrame a framework with distributed components. The developers have not yet focused on the security issues in ServiceFrame. As a result ServiceFrame currently has no security mechanisms for securing actors or agents. This thesis proposes a security protocol and security mechanisms for securing ServiceFrame. The proposed security mechanisms are implemented in a prototype and tested using a test case.

The report first gives an introduction to security concepts, such as authentication, authorisation, integrity and confidentiality. The report also describes relevant frameworks and security protocols. The Java programming language is used for developing and implementing the security mechanisms. UML 2.0 is used as the modelling language. At the end of the report the security mechanisms are analysed and discussed.

Authentication in ServiecFrame is achieved by using a key exchange protocol with certificates. In the thesis a solution for achieving authorisation is proposed. This thesis only proposes a rudimentary solution which uses access control lists. Integrity and confidentiality are achieved using cryptography and signing of messages.

The main conclusion of this thesis is that the security mechanisms proposed can contribute to securing the ServiceFrame framework. The security mechanisms achieve point to point security between two agents. ServiceFrame could be used to secure access to the Parlay gateway and telecom services. Fundamental in the security mechanisms is an extended variant of the Needham-Schroeder-Lowe public key protocol.

The main contribution of this thesis has been to introduce security in ServiceFrame, which previously had not been implemented. The security mechanisms can be used by developers of ServiceFrame to accomplish security in their services. Commercial systems require focus on security to secure both end users and the service providers. The thesis work may contribute to the establishment of ServiceFrame in commercially related products in the future.

The thesis has shown that ServiceFrame does not have security mechanisms and that achieving security is essential for ServiceFrame. The thesis has also shown that some of the proposed security mechanisms can be implemented in the framework. It has also shown how security concepts can be implemented and used by distributed components.

Preface

This thesis is a part of the Master Degree in Information and Communication Technology at Agder University College, Faculty of Engineering and Science in Grimstad. The thesis is written for the Norwegian Advanced Research Center (NorARC), a division of Ericsson in Asker. The thesis work has been carried out between January and May 2006. The work has predominantly been carried out at Ericsson in Asker.

We would like to thank our supervisor Geir Melby at Ericsson, Asker and his colleague Knut Eilif Husa. We would also like to thank Professor Vladimir Oleshchuk at Agder University College, Fritjof Boger Engelhardtsen, Margaret Gulbrandsen and Morten Kråkvik.

Grimstad, May 2006

Trond Ivar Johansen and Olav Bjering Lømsland

Table of Contents

List of Figures

1 Introduction

1.1 Background

The telecommunication world today is experiencing rapid changes. The network operators previously had full control of their system and therefore also full control of the security aspect. Nowadays, the network operators are opening up their network to third party developers, as a result of convergence between the telecom world and the IT world. Standardisation organisations have defined Application Programming Interfaces (API) standards for the third party developers. One example of an API is Parlay/OSA. Through the Parlay API telecom services such as positioning, SMS, MMS and call control are offered for third party developers. Parlay X is another API that can be used for service development. Parlay X is based on Web Services and also provides services for third party developers. This third party interaction raises new security issues since the network operators no longer have full control of their network. One example is a bug in an application using the positioning service which resulted in thousands of lookups during a short period of time. This bug caused a major system overload. Fortunately, this was only a test application but it would have been tragic if the user had been charged.

Ericsson is a major supplier of telecommunication equipment and systems. Ericsson has a Parlay gateway called Network Resource Gateway (NRG) [102,76] with a corresponding API. Ericsson's NorARC is a research department at Ericsson's in Asker which has developed a prototype of service-oriented framework, called ServiceFrame. The framework is modelled in Unified Modeling Language (UML) version 2.0. ServiceFrame makes it possible to create distributed components that can offer services for the telecom networks using the Parlay API. Parlay and Parlay X give ServiceFrame the connection to the telecommunication networks. The ServiceFrame components are loosely coupled and communication is done through asynchronous messages and concurrent state machines.

Security in ServiceFrame has not been addressed earlier. The focus for developers in ServiceFrame has been component development and not security related development. The reason for securing ServiceFrame is that the telecom network operators and third party developers need a service oriented framework, where the security is ensured. This thesis will address the following security concepts: authentication, authorisation, integrity and confidentiality. In the thesis there will be proposed security mechanisms for protecting ServicFrame. The key issues concerning security mechanisms are cryptographic algorithms, certificates and security protocols.

1.2 Security

Today there is an ongoing focus on security. An increasing number of people and computers are connected to the Internet and more businesses are available online. The Encyclopædia Britannica defines Computer Security as "*the protection of computer systems and information from harm, theft, and unauthorized use*" [47].

There are several ways an attacker can compromise security. The main security mechanisms that prevent an attack from succeeding are Authentication, Authorisation, Confidentiality and Integrity. Authentication is the process of attaching an identity to an entity. Authorisation means to give an entity the correct access rights or privileges to a resource. Message integrity implies that the message contents have not been tampered with. Confidentiality denotes that the information is hidden from unauthorised disclosure.

1.3 Ericsson NorARC's ServiceFrame

Ericsson NorARC has developed a prototype of a service creation and execution framework called ServiceFrame [111]. Throughout some of the literature regarding ServiceFrame and ActorFrame these two frameworks have been used interchangeably. ActorFrame defines the core concepts such as actors and agents while in ServiceFrame these concepts are used to create new components or services. Actors are distributed components which are loosely coupled. Each actor has a behaviour which is defined in a state machine. The agents are actors with extended functionality. The agent may have more than one state machine associated with it. These additional state machines are called roles. Actors may also have roles but these are other actors functioning as inner parts. ServiceFrame has service components called edges for accessing the telecommunication network. ServiceFrame has defined edges which are called NRG edges for accessing Ericsson's NRG Parlay gateway. The name ServiceFrame sometimes also refers to the application server where the ServiceFrame framework is running. Actors are distributed components that can communicate independently of lower communication layers and this makes ServiceFrame a middleware.

1.4 Thesis definition

The thesis will address the need for security in Ericsson ServiceFrame service creation framework. Security mechanisms and security issues, as well as relevant frameworks and security protocols will be discussed.

The final thesis definition is formulated as:

The students will propose and evaluate security mechanisms for ServiceFrame. The main security mechanisms under consideration will be mechanisms for authentication, authorisation, integrity and confidentiality. If time allows, the proposed security mechanisms will be implemented and tested in a prototype.

The final thesis title is:

Application Level Security Enforcement Mechanisms for Advanced Network Services

1.5 Work description

The main objective for this thesis is to propose and evaluate security mechanisms for ServiceFrame.

The thesis has been divided into three sub problems that will be addressed:

How to authenticate Agents in ServiceFrame?
Authentication is the key concept for providing security in ServiceFrame, and we will propose mechanisms for authentication in conjugation with the other security mechanisms. Actors and Agents are the main components in ServiceFrame and it makes sense to authenticate these.

How to authorise Agents in ServiceFrame?
Agents are key components in ServiceFrame and authorisation of these is important because there may be situations where access to resources need to be controlled.

How to achieve integrity and confidentiality on messages in ServiceFrame?
Since ServiceFrame is a middleware, all communication is carried out through messages. The messages may be sent through insecure channels, which lead to a need for integrity and confidentiality.

To answer these questions we first describe relevant security concepts and protocols. We will then describe relevant frameworks and the security issues in these. We will then propose security mechanisms and afterwards describe how we implemented these in a prototype in ServiceFrame. A test case is then developed. Finally we will discuss the implementation and the test case.

Throughout the report we will use UML to model and describe components.

1.6 Report outline

In Chapter 2 fundamental security concepts are presented. In this chapter authentication, authorisation, integrity and confidentiality are described. Main security topics are cryptography and signing. This chapter can be skipped by those who are familiar with security concepts.

Chapter 3 describes security protocols and relevant framework for this thesis. A key protocol is the Needham Schroeder Lowe public key protocol and an important framework is the Parlay API. Those who are not familiar with security protocols and Parlay API are encouraged to read this chapter.

In Chapter 4 we describe key concepts in UML 2.0 such as Port, Part, Connector as well as sequence diagrams and structured classes. We describe components of Ericsson's service creation framework such as ServiceFrame, ActorFrame and JavaFrame. Readers who are familiar with UML 2.0 and and Ericsson's ServiceFrame can skip this chapter.

In Chapter 5 we give a threat assessment of ServiceFrame and ActorFrame. We then propose security mechanisms for ServiceFrame. Security mechanisms for authentication, authorisation, integrity and confidentiality and a key exchange protocol are proposed. Limitations of the proposed mechanisms are described.

In Chapter 6 we describe the design and implementation of the proposed security mechanisms in a prototype.

In Chapter 7 we propose a test case scenario where a user sends a message securely from one agent to another. The test case demonstrates the proposed security mechanisms implemented in the prototype.

In Chapter 8 we discuss the proposed security mechanisms in relation to the implementation and the test case. The conclusion of the thesis is drawn based on the discussion.

2 Security concepts

2.1 Introduction

Security is an issue that concerns an increasing amount of people. The increased interest is caused by a boost in Internet use. There is an ongoing convergence between telecom and Internet related services. People want to have the same security on Internet as they have experienced in telecom related services. This demand calls for an increased focus on security issues on the Internet. In this chapter a short description of security related topics are described.

2.2 The history of security

Communication and security has throughout human history been closely connected with one another. Encryption played a key role when the telegraph was introduced in the 1840's. The use of the telegraph played an important part in the American Civil War between 1861 and 1865. The invention of the radio and of the telephone led to new areas where encryption could be used. Encryption in radios and telephones was extensively used during the Second World War. The mythical electrical encryption and decryption machines used by the Axis Alliance (Germany, Italy and Japan) were called "the Enigmas". As a result, the Allies developed the first truly electronic computer due to the considerable task of breaking the Axis Alliance's codes.

In 1973 the need for a standard encryption algorithm was addressed. As the human society has moved into the computer era, encryption has become a public need due to an increasing amount of information being processed. In November 1976 the Data Encryption Standard (DES) [3] was approved. The secret key, also known as a symmetric key, was the only encryption mechanism until 1976. In that year the public key cryptography, named asymmetric encryption, was introduced. The first algorithm was called Diffie-Hellman [11] key exchange. The asymmetric method handles the problems which a symmetric key had. Two parties no longer need to exchange their private key in advance. In 1978 the algorithm RSA [12] was introduced. The RSA introduced a new concept called digital signature, which is used to ensure that the message has authenticity.

Earlier encryption was primarily performed by governments, but this has changed with the introduction of the Internet. When the Internet became publicly known in the late nineties more companies and private individuals gained access to the Internet. This created a new demand for encryption. Companies wanted to secure information passed via the Internet, in order to prevent disclosure of sensitive information. A great deal of private information is sent through the Internet and encryption is often desired. An example of this is bank services. In July 1998 the DES algorithm was cracked within 56 hours. Triple DES [3] was developed as an alternative and introduced on 25 October 1999. A new standard was chosen during an international competition in 2001 [101,21], the objective being to select an algorithm to become the new Advanced Encryption Standard (AES) [14] and the algorithm Rijndael [15] was selected.

2.3 Key security concepts

In this section key security mechanisms will be described. Throughout security literature the names Alice and Bob are used to refer to two subjects that want to communicate securely.

The names Eve and Trudy are usually referring to attackers. A cipher is a cryptographic algorithm used for encrypting or decrypting data.

Authentication

Basically, authentication is the necessity for an entity, user or process to prove its identity. An entity must identify itself and then prove its identification to the system. This process is called authentication, and is the process of mapping an entity to the correct identity. The user Alice must prove to the system that she really is the Alice that the system knows, and not Eve in disguise. The problem with authentication is if another user Eve can impersonate or try to identify herself as Alice to the system. It should not be possible for Eve to claim Alice's identity.

The usual method for authentication of physical users is to employ a username and password. The username holds information about an identity and the password is something only the corresponding entity or user should know. For processes and components in a computer system other mechanisms for authentication are used. The most common way of authenticating processes is to use certificates. A certificate is a way of verifying the entity and will be explained later under chapter 2.9 .

Authorisation

Authorisation means to give an authenticated entity, user or process, access to some part of a system. Authorisation could be achieved by looking up what access rights an entity or subject has in a table or an Access Control Matrix (ACM) [42,43]. The ACM will then include one row for the subject and a column for the object. The objects are the systems resources under access control. The values within the matrix could state what kind of access rights the subject has over an object. Since the ACM would be a huge matrix for every subject and object in the system it is more efficient to divide the ACM into two categories. It is possible to define an Access Control List (ACL) for each object or resource in the system. Each object has an ACL where everyone who has permission is listed and what kind of rights they possess. In these lists explicit deny rights can also be listed, meaning that an entity or subject does not have this right. This is useful especially if the access control lists are inherited from other objects in a hierarchical structure, like a directory structure. This means that a user, for instance, can be allowed to access a directory on the root level and all subsequent directories, but be denied access to a specific directory. The other category is the Capability lists or C-lists, where each subject or person has a defined list of what rights it possesses over an object. Both ACL and C-lists are explained more under Chapter 2.10 .

Other forms of authorisation can be used in other security models. In the Role Based Access Control security model [45], access to resources is controlled by what kind of role an entity is associated with. Instead of mapping access rights from entities (subjects) to resources (objects), access rights are mapped from entities to roles and from roles to resources.

Integrity

Integrity in a security context means that it is possible to detect if data have been tampered with. It could also mean that it is possible to prove that the one who claims to have created the message is the one who actually did so. Data integrity is usually achieved by means of a Cyclic Redundancy Check (CRC) in order to protect against data corruption. CRC is not sufficient to achieve verification of the sender or integrity of the data. Anyone could generate a false CRC after they have changed the contents. Signing of data is therefore necessary to protect against deliberate tampering with data. The signing can be achieved by using a

hashing algorithm and then encrypting the hash. A hashing algorithm is an algorithm that transforms many bytes of data into a few bytes. It is known as a one way function because it should not be possible to reconstruct the original data from the hash value. Another property of the hashing or message digest algorithms is that data with different contents should give different hash values. The likelihood that two different data give the same output should be very small. Another option for signing would be to encrypt the entire message with the signer's private key. This will ensure the recipient that the message originated from the person who signed it, because only the "owner" of the private key can encrypt with it, as long as the key is not exposed or compromised. Private keys are explained under "Asymmetric encryption algorithms" in Chapter 2.5 .

Confidentiality

Confidentiality means to hide information from unauthorised access. This is usually achieved by encrypting the information using a symmetric or an asymmetric algorithm. Encryption algorithms are explained in Chapter 2.4 and Chapter 2.5 .

Availability

Availability means that services or servers are accessible to legitimate users. This could be achieved by using load balancing or redundant servers in case some of the servers are down.

Accountability

Accountability refers to the ability to make users or processes responsible for their actions, such as to achieve non-repudiation. Accountability can be used to prove that someone placed an order or sent a message. This can be done through digital signatures of messages. Auditing and logging can also be used to keep a record of transactions or accesses to resources, although this cannot prove anything.

2.4 Symmetric encryption algorithms

A symmetric encryption algorithm [16,4,9] or symmetric cipher, is used encrypt messages. The key is kept secret. The same key is used for both encryption and decryption. Symmetric encryption supports both key and password encryption. This report will only focus on the key based encryptions. Two types of symmetric algorithms exist, stream and block algorithms. In the stream case the algorithm encrypts a bit or a byte. The bits or bytes are assembled into a stream string which contains the message. The stream encryption is faster and normally the encrypted output is shorter than the encrypted blocks in the block algorithm. In stream encryption there is a continuous flow of bits. The encryption function is simpler and it can for example use bitwise exclusive OR (XOR). With stream encryption padding is not necessary in contrast to block-algorithms. Stream encryption has this advantage in the encryption method compared with block algorithm.

Block algorithm encrypts one block at a time typically 64-bit. The size of the block cipher depends on the size of the input material. The objective for the block cipher is to arrange encryption and decryption. In this section the topic will be algorithms. We will only look at the most used symmetric algorithms and the most used parameters for the symmetric algorithms. For more depth and theory the following books are recommended [16,4,91].

AES

Prior to November 2001, Triple DES was preferred. Since then the Advanced Encryption Standard (AES) has become the favoured algorithm for the National Institute of Standards

and Technology (NIST). AES has a fixed block length and the key size can be between 128-bits and 256-bits.

AES was originally made to protect un-classified data. In June 2003 the US Government published a paper [17], which states that AES of all key lengths withstand cracking of data up to the security level secret. With the key lengths of 192 bits and 256 bits it can be used for the security level top secret. The first attack against AES called "XSL attack" was published in 2002 by Nicolas Courtois and Josef Pieprzyk [17]. "XSL attack" is a specific way of understanding the encryption information without the knowledge of secret key. There is still an open question whether or not this attack form will work against AES. Many cryptography experts are also questioning the underlying mathematics used by the authors in the attack. To date, two "side channel attacks" against AES are also known. "Side channel attacks" do not attack the algorithm but the implementation of the cryptosystem. At present two papers on "side channel attacks" have been published. In April 2005 D. J. Bernstein published the following paper [19]. The attack was a "cache timing attack" on AES on the custom server that used OpenSSL. By "cache timing attack" it means that a malicious local user gained access to parts of the cryptographic key. The second paper was published in October 2005 by Dag Arne Osvik, Adi Shamir and Eran Tromer [20]. This paper also describes "cache timing attacks". One of the attacks did manage to obtain the AES key, but the attack needed to be done on the same system that AES encryption was performed.

AES is esteemed as a simple, fast, secure, universal algorithm demanding little memory. The AES algorithm is compatible with a large range of processors.

The symmetric block algorithms have modes, which mean how blocks are arranged when encrypting. There is one mode mostly used by DES, Triple DES and AES: Cipher Block Chaining (CBC) [4,9,1]. Figure 2.4-1 shows that the first block does not have a block in front. The CBC mode uses an initialisation vector (IV) as a way of compensating for the block in front. This IV needs to be the same when encrypting and decrypting the same content. In a communication system the IV can be distributed without any encryption, and this has been taken into consideration in the manufacturing of the mode. CBC uses an XOR to join ciphertext with plaintext. When decrypting, as shown in Figure 2.4-2, the first ciphertext block is decrypted and XORed with the IV. In the following steps the next ciphertext block is decrypted and XORed with the previous ciphertext block. The use of IV, which is publicly known, makes it easier to crack the ciphertext. When the ciphertext from block one is cracked, then the rest of the blocks can be broken. CBC can also be used for integrity which is achieved by using CBC mode twice with different keys. The last block of the first encryption is used as a Message Authentication Code (MAC). MAC is a part of the message that can be used as message authentication.

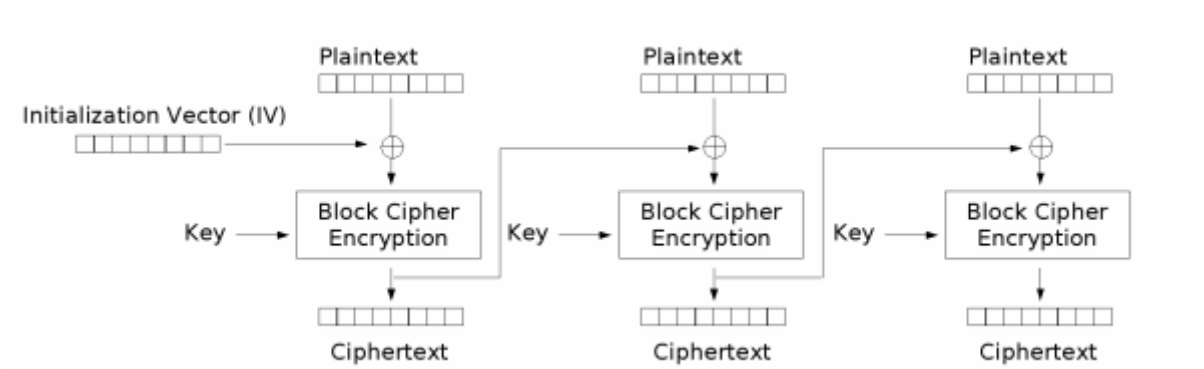

Figure 2.4-1 Cipher Block Chaining (CBC) mode encryption [23]

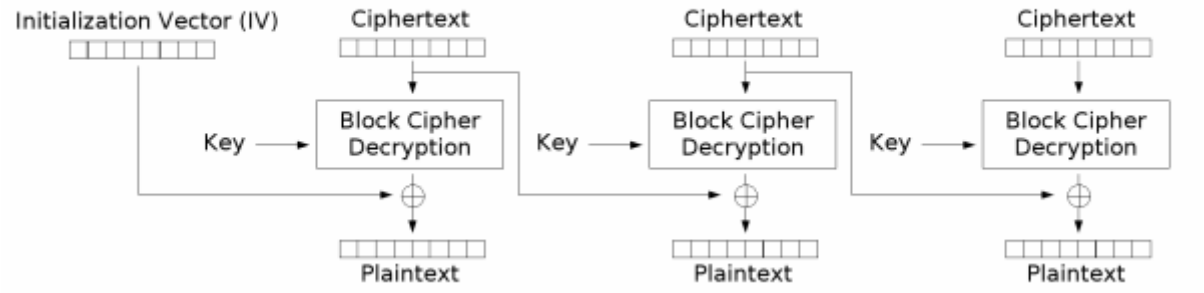

Figure 2.4-2 Cipher Block Chaining (CBC) mode decryption [24]

2.5 Asymmetric encryption algorithms

In an asymmetric encryption algorithm there is a key pair instead of a shared, secret key. The key pair consists of a public and a private key. Both the public and the private key can be used for encryption. A message encrypted with a public key can only be decrypted with the corresponding private key. A message encrypted with a private key can only be decrypted with the corresponding public key.

Public keys are as the name denotes, known to the public, i.e. they are not secret. Nor do they need to be secret because the only person capable of decrypting a message encrypted with a public key is the one person who knows the private key. Private keys are private and should never be exposed to others.

Public and private keys have also another property which is important in computer security; they can be used for signing. Signing means that a message originating from one part, Alice, can be proven to have come only from Alice and no one else. At least the message is created by Alice although you cannot know if it actually came from Alice directly. If a message has been encrypted with Alice's private key anyone can decrypt the message because the public key is known to everyone. The important fact is that Alice is the only one who can encrypt with the private key, and therefore if a message is encrypted with a private key, it must have originated from Alice.

Asymmetric ciphers are usually slower than symmetric ciphers and are therefore only used in a key exchange to authenticate each part and to exchange the secret (symmetric) key. When you need to exchange data larger than 64-bit the symmetric cipher is the preferred option instead of asymmetric. The reason being the efficiency the symmetric key has when it exchanges the message. The symmetric ciphers normally have shorter key sizes than asymmetric ciphers and the symmetric ciphers are usually based on mathematics that needs less computation.

RSA

The most used asymmetric encryption algorithm today is RSA, invented by Ronald Rivest, Adi Shamir, and Leonard Adleman. The RSA algorithm uses two large prime numbers and modulus arithmetic to calculate the private and the public key. It is difficult to break the RSA keys because factoring of large numbers is a difficult task and there are no known algorithms that can do this efficiently. Breaking the RSA keys means to find the private key when only knowing the public key. The RSA algorithm can be used both for encryption and for signing. Signing can be accomplished by encrypting with the private key because only the originator can encrypt with this key. RSA encryption uses a block cipher to encrypt data. RSA Security has a cracking competition where the objective is to factorise large numbers. RSA-640 is a large number in this competition, and has been factorised. It is a 663-bit [46] long number. Key sizes used today are between 1024 and 2048 bits.

RSA Mathematical explanation

The RSA algorithm [27,7,28,12] is highly mathematical and uses modulo arithmetic and large primes. Below follows a brief explanation of the algorithm:

- To generate an RSA key two large primes p and q are chosen, where $p \neq q$ and $n = p * q$.
- A number e is chosen which is bigger than 1 and relative prime to both $p - 1$ and $q - 1$ and with $e < (p - 1)(q - 1)$.
- A number d is calculated so that $d < (p - 1)(q - 1)$ and the remainder of $d*e$ divided by $(p - 1)(q - 1)$ is 1.
- The public key is (e, n) and the private key is d.
- Let m be the plain text message to be encrypted and c is the encrypted message. Then $c = m^e \bmod n$ and $m = c^d \bmod n$.

A secret message m can then be encrypted to c by taking the remainder of m^e divided by n. Decryption of c can be done by taking the remainder of m^e divided by n. There are some known weaknesses in RSA when using small n. This can be avoided using larger n.

2.6 Elliptic Curve Cryptography

Elliptic Curve Cryptography (ECC) is claimed to have equal security as RSA for smaller key size [8,35]. RSA needs large keys and therefore much processing power whereas ECC have smaller keys and is faster. The theory of ECC has existed a long time, but only recently it has been focused on ECC as an alternative to RSA, especially on platforms that require less processor demanding algorithms. ECC is highly mathematical and complex. ECC can be used in conjugation with key exchange protocols such as Diffie Hellman.

2.7 Padding

In block algorithms the data is divided into blocks of fixed size. The result can lead to the last block not being filled. Padding is an option you can make use of when the whole block is not fully utilized. It is possible to specify the kind of padding scheme that is going to be used.

PKCS #5 and PKCS #7

Two examples of padding schemes used by RSA and AES are PKCS #5 [25,16] and PKCS #7 [26,16]. The PKCS #5 and PKCS #7 were originally developed for asymmetric encryption. PKCS #5 is used for block ciphers of 8 byte, and PKCS #7 is used for blocks of up to 255 bytes. The method used for PKCS #5 and PKCS #7 is very simple as shown in Figure 2.7-1. When there is one byte in use for padding it is simply entered as a "1". If there are four empty bytes the number "4" is put into every empty byte. When there are no bytes unused, the next block will be filled with eight "8's". Padding is done before the whole block is encrypted. In relation to the encryption, padding is important because an incomplete block is easier to break than a full block.

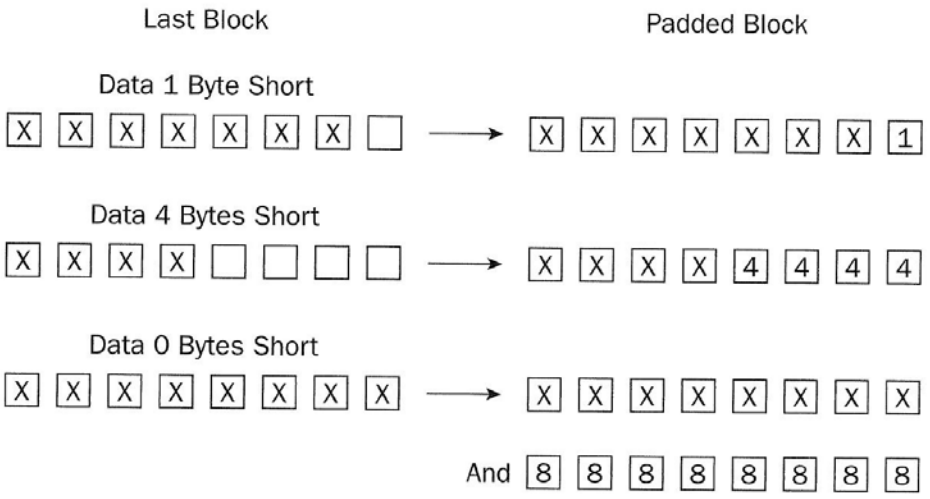

Figure 2.7-1 PKCS #5/#7 Padding with an 8 Byte Block Cipher [4]

Optimal Asymmetric Encryption Padding

Optimal Asymmetric Encryption Padding (OAEP) [30,8] is a padding scheme for RSA. It was introduced in 1994 by Bellare and Rogaway. RSA Security recommends OAEP for newer applications [31].

The Figure 2.7-2 shows how the padding is constructed.

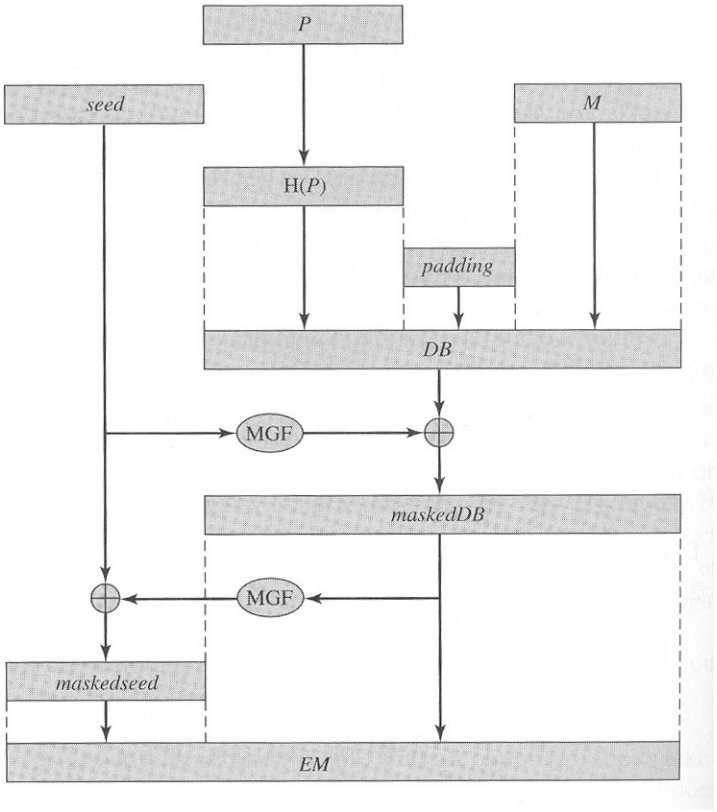

Figure 2.7-2 Optimal Asymmetric Encryption Padding [8]

The symbols in the figure have the following meanings:
M= Message to be encoded
P= Encoding parameters
DB= Data block
MGF= Mask generation function

The encryption is carried out as follows. First the message "m" is padded before it is encrypted. Next the "P" which is an optional parameter is put through hash. The output from the hashed "P" is padded with zeros to fit to the "DB" block. Then the "seed" is put through a hash function "MGF" which is called mask generation function. The result from "MGF" is XOR'ed with the "DB" bit-by-bit. The masked "DB" is put through a "MGF" and the XOR'ed with a "seed". The outcome of the XOR'ed function is a masked "seed". The masked "seed" and the masked "DB" are put together and form a united block.

2.8 Signatures

Signatures can be used to achieve integrity of data, i.e. they can be used to verify whether or not data has been tampered with. Signatures are essential in certificates, which will be explained later in chapter 2.9 .

RSA
An RSA algorithm can also be used to sign [4,56] messages as earlier mentioned. The Figure 2.8-1 shows the process of signing and verifying the signature. The creation part will be explained first. The first step is to use a digest function on the data. Then the private key and the result from the digest function are put into the RSA engine and signed. At the receiver end, data is put into a digest function as in the creation of the signature. The next process in the

verification of the signature is to put the public key and the signature into an RSA engine. The result from the RSA engine is put into a recovery digest. Finally, the result from the digest function and the recovered digest is used to verify that the two results are the same, otherwise the content has been tampered with.

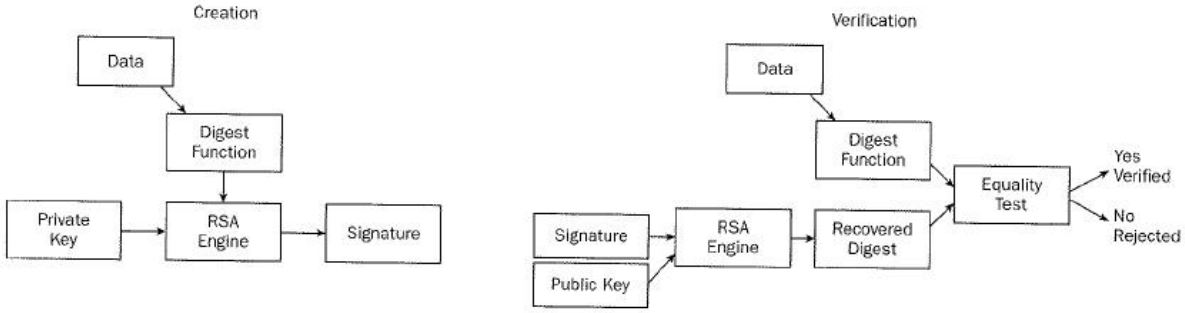

Figure 2.8-1 RSA signing and verification [4]

DSA

In August 1991 the digital signature DSA [52,53] was proposed by the U.S. National Institute of Standards and Technology (NIST). It became the first digital signature to be accepted by the US Government. DSA was published as a standard with the name FIPS PUB 186. An important difference between RSA and DSA is that DSA can only be used to sign data in contrast to RSA, which can be used to encrypt data. The Figure 2.8-2 shows the process of signing data; first the data is put into a digest function. The result of the message digest and the private key is put in a DSA engine and a result appears as a signature. At the receiver end the data is put into a message digest function. The result of the message digest and the signature and the public key is then put into a DSA engine. The engine determines whether the signature has been tampered with or not.

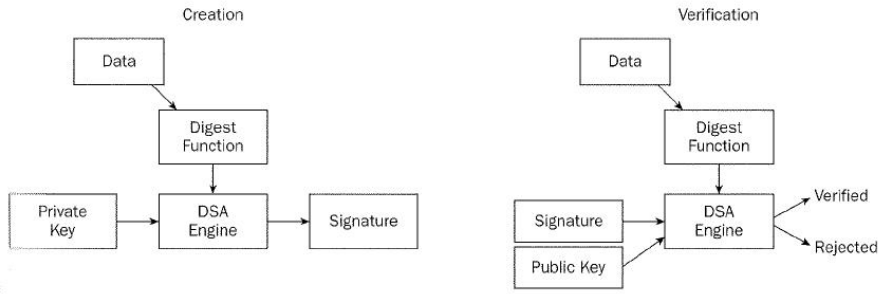

Figure 2.8-2 DSA signing and verification [4]

SHA 1

SHA 1 [33,5] was published as a standard with the name FIPS PUB 180-1 in 1995. NIST has stated that they will phase out SHA 1 by 2010 in favour of SHA 2. It is proved that SHA 1 is broken. SHA 2 is not yet broken, but since SHA 1 and SHA 2 are similar, researchers are worried. The SHA 2 is the name of a group of four new variants. The names of the four are SHA-224, SHA-256, SHA-384, and SHA-512. The names are given according to the digest lengths. This algorithm is supported in the standard Java security provider. The Figure 2.8-3 shows how SHA 1 signs. The "A" to "E" is 32 bit words. The "F" function is a nonlinear function which changes. $\lll_s$ symbol means that this function moves "S" steps left, and "S" can vary with each operation. The symbol $\boxplus$ symbolises modulo 2^{32}. The "K_t" is a constant. "W_t" is a circular array of words that is sixteen in length. If *t* is the operation number (from 0

to 79), W_t represents the t^{th} sub-block of the expanded message, and $\lll_s$ represents a left circular shift of *s* bits, then the main loop looks as follows:

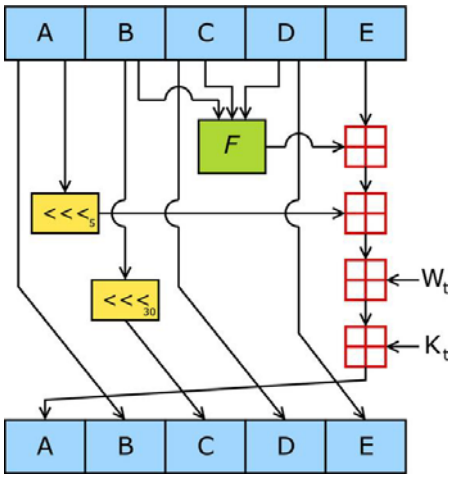

Figure 2.8-3 SHA-1 [22]

HMAC

Hash Message Authentication Code (HMAC) [34,35,36] is used for achieving message integrity. HMAC creates a message digest of a message, along with a key to prevent tampering of the digest. Both the key and the message are hashed, which results in a different hash for the same message if another key is used. This gives messages integrity if the key is secret. An attacker cannot create a valid HMAC of a new message if he does not know the key.

2.9 Certificates

The major problem using asymmetric keys, and symmetric keys for that matter, is how someone can know that the keys actually belong to Alice, for example. Key distribution is one of the most difficult security problems to solve. The most common solution is to use a third party that everybody trusts. One variant is that everybody would then contact this trusted third party every time they need someone's public key. It is difficult to get everyone to trust one common part. Another more accepted solution is to use certificates issued by a third party, a Certificate Authority (CA). Certificates are only valid for a period of time. After that the certificate expires and is invalid. There is also a possibility that a certificate may be invalidated before the expiration date. The certificate is then added to a Certificate Revocation List (CRL).

Problems concerning CAs are specified in [114]. If a CA is irresponsible when signing certificates problems may occur. When a CA has many parties that use the CA, the CA needs to careful when giving Id's. If the CA is not careful when giving Id's, two or more parties may end up with the same Id. This could allow Eve to perform many of the earlier mention attack types on Alice and Bob. A problem will also occur if the CA does not check the background of parties granted entry in the certificate chain.

The thought of public key certificates was first introduced in 1978 by Kohnfelder in his paper [39] published that year. The X.509 [38] was first introduced in 1988 and is a part of the X.500 directory service standard. The most recent version was released in August 2005 [40].

The certificate can contain eleven fields as follows:

1. Version: This field tells which version of the X.509 is used
2. Serial number: This integer number is unique for every certificate
3. Signature algorithm identifier: This field has no relevance since it tells about the algorithm and an associated parameter explained later in the “subject’s public-key info”.
4. Issuer name: This field is dedicated to the CA that creates and signs the certificate.
5. Period of validity: In this field the first shows the date from which the certificate is valid, and the second field shows the expiry date.
6. Subject name: This is the field name of the owner of the private and the public key that is used in the certificate.
7. Subject’s public-key info: This field contains the public key of the subject and the algorithm and the parameters connected to the algorithm.
8. Issuer unique identifier: This is a field which contains the CA and is represented in a bit string. This id can be reused and as a result represents a threat.
9. Subject unique identifier: This is a field which contains the Subject and is represented in a bit string. This id can be reused and as a result represents a threat.
10. Extensions: In this field there is additional information.
11. Signature: This field contains three areas: Here there is the algorithm and the parameters which the certificate makes use of, and lastly the signature.

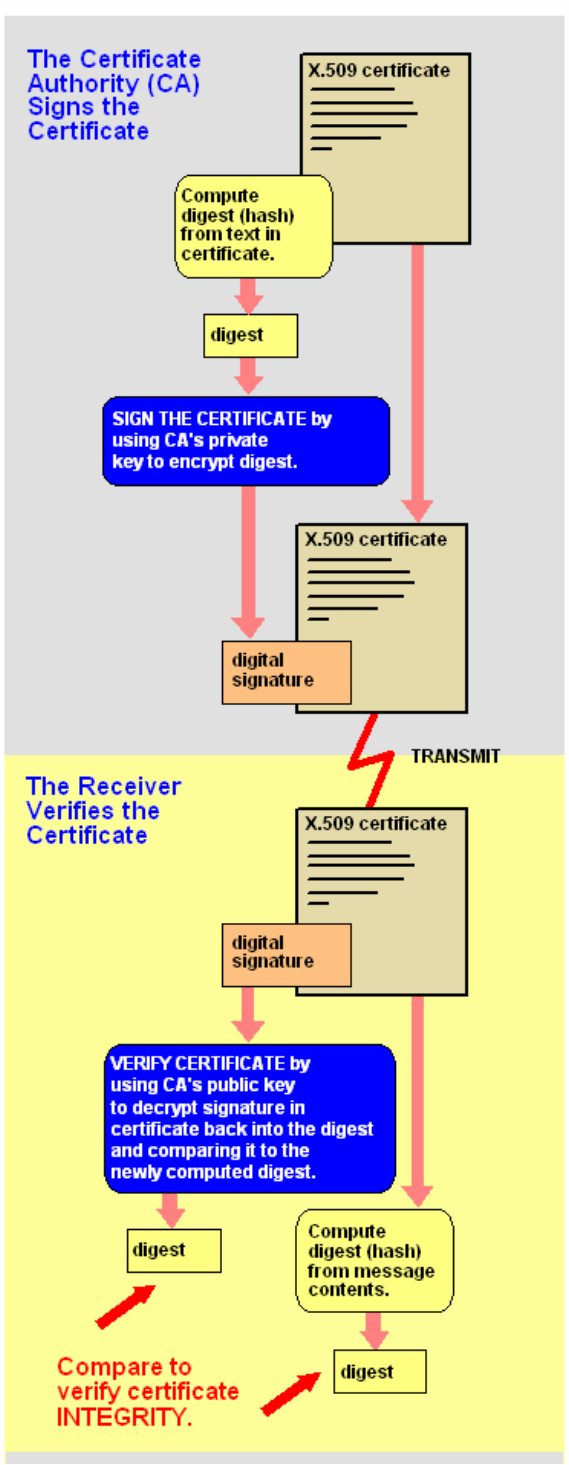

Figure 2.9-1 Certificate signing and verification [41]

The Figure 2.9-1 shows how a CA signs a certificate and how the certificate later can be transmitted and verified by a receiver. The receiver must trust the CA.

1. First the CA computes digest
2. Then the CA signs the computed digest with its private key
3. Afterwards the digital signature is added to the certificate
4. The certificate is then ready to be forwarded
5. The receiver of the certificate starts by checking out the digital signature
6. The receiver begins by using the public key from CA to decrypt the digital signature
7. Then extracts the digest from the signature
8. The receiver then computes the digest from text in the certificate
9. Then the digest in step 7 and 8 are compared
10. If the two digests are the same, then the certificate has not been tampered with

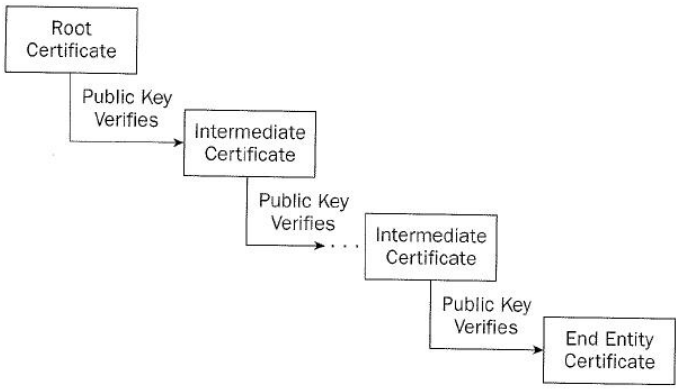

Figure 2.9-2 CA chain [4]

The Figure 2.9-2 shows a CA certificate chain. The example on the previous page shows the process of signing and verifying a certificate. Every certificate signs the next certificate in the chain with its private key. An exception is the root certificate or the first entry in the certificate chain, which is self signed. The process of verification is carried out by checking the certificate at the bottom and all the way up to the root certificate at the top. The advantage of CA signed certificates lies in the fact that the only party that can claim the identity in the certificate is the certificate owner.

2.10 Access Control

Access Control [35,9,7] and authorisation in computer systems can be achieved through the use of an Access Control Matrix (ACM) [42]. An ACM is a matrix where the subjects or entities are defined by the row and the objects or resources defined by the column. Subjects are entities such as the persons or processes given permission over some object or resource. An example of such an ACM is given below. A "W" denotes write permission and an 'R' denotes a read permission, but this could be defined to be anything.

	Printer C	File A	File B
Alice	W	R	
Eve			R
Bob	W	RW	
Trudy			

Such a matrix is not efficient because it would have to have allocate space for all the resources and the users or processes, even if they are not in use. The matrix will be sparsely filled and therefore the matrix is divided into tables for the subject's rights and the object's permission. The list which defines which types of permission the different resources are allowing subjects to have, is called an Access Control List (ACL) [43,7,9,35]. There is one such list for each resource. For "File A" this ACL could look like this:

File A

Alice	Eve	Bob	Trudy
W		RW	

There is also a variant called Capability lists or C-lists [7,9,35]. The Capability lists focus on the subjects and their permission to different objects or resources. The C-list for Alice could look like this:

Alice:

Printer C	W
File A	R
File B	

Access Control Lists are used in operating systems like Windows [44]. Some prefer C-list over ACLs [35,105], especially when delegating permission [104].

Authorisation can be granted by giving the user the permission associated with him in the C-list or the rights on objects if he is included in the object's ACL. In order to make sure the right user or process is asking for the permission they must first be authenticated.

More information on Access Control can be found in computer security books such as [7,9,35].

Role Based Access Control

Role Based Access Control (RBAC) [37] is an access control model where users are attached to roles. Roles are then given permission to resources. In this model it is easier to control which user has what kind of rights even though the users might change jobs. The users who change jobs are only linked to a new role and therefore there will be no need to update the user rights. In traditional capability lists the user would have to have all his old rights removed and the new rights applied. This is not necessary if the user can only be linked to a new role. As shown in Figure 2.10-1, users are connected to roles which are given permission of an object.

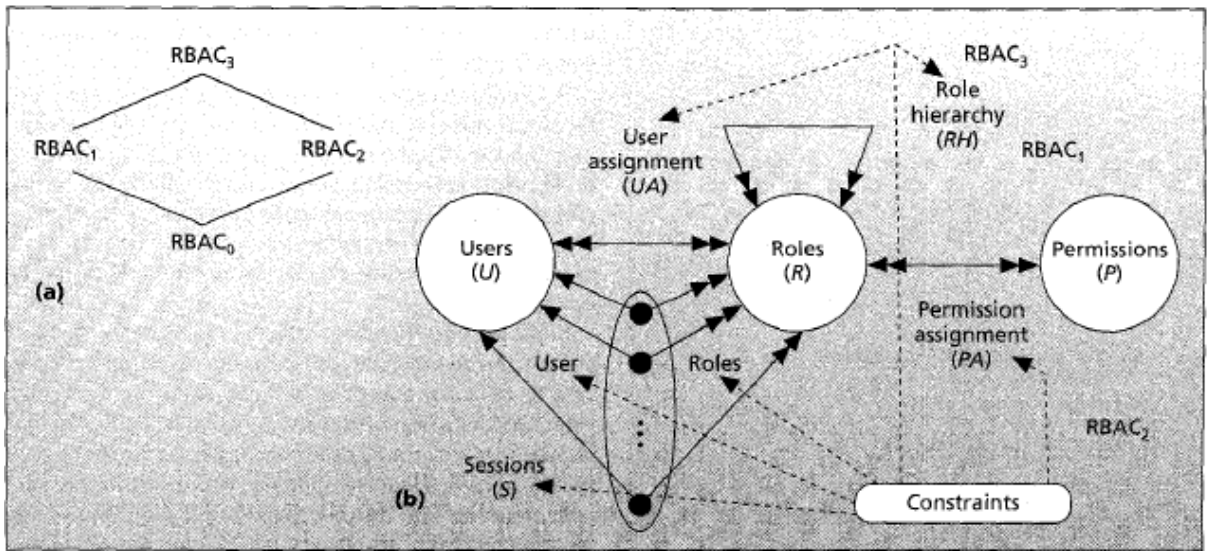

Figure 2.10-1 RBAC model [37]

More information on RBAC can be found in [37] and [45].

Discretionary Access Control

Discretionary Access Control (DAC) is a security policy model for access control where the users and processes themselves specify which access rights and permission they give to objects.

Mandatory Access Control
Mandatory Access Control (MAC) is a security policy model for access control where the system handles the access rights and permission.

2.11 Computer and network attack types

This section will explain different types of computer and network attacks. The attack types mentioned in this chapter will be discussed later in this thesis. More information about the attack forms can be found in the following books [7,5,35].

Man-in-the-middle attack

A "man-in-the-middle" attack is when an attacker, Trudy, intercepts messages passing from Alice to Bob. She would then have to be the link between Alice and Bob. Somewhere in the middle Trudy controls the data flow between them. In this way messages sent from Alice to Bob would have to go through Trudy. This means that Trudy could alter the messages before they reach Bob and Bob would have no idea of this. Trudy could also eavesdrop on the data flow trying to find out what the message contents are. If she can intercept the messages on way between Alice and Bob she can fool Bob to believe she is Alice. This attack can be prevented by combining authentication with confidentiality and integrity.

Replay attack

Replay attack is an attack where Trudy intercepts messages from Alice to Bob, but she might not be the link between them, i.e. messages are not sent through Trudy because she only eavesdrops on the data flow. Trudy can, when she has received a message from Alice to Bob, use the same message and replay it to Bob. Bob would then be unable to tell that this message was not sent twice from Alice, but came from Trudy instead. Trudy need not know the exact contents of the message in order to attack, but this would make the attack easier. She could try to guess the context of the message if the message is short and recurring, i.e. if it is stock trades with messages such as "buy" and "sell". This attack can be avoided by combining authentication, confidentiality, integrity and time stamps of the messages.

Tampering with data (attack on integrity)

Tampering with data means that the integrity of data is modified intentionally by an attacker. Normal data integrity can be achieved through Cyclic Redundancy Checks (CRC), but this check only verifies that the data has not lost integrity through an error in a communication channel. It does not protect against attackers because they can always create a new CRC for the tampered data so that the CRC is valid. To prevent this it is necessary to achieve cryptographic integrity of data. No one other than the original sender can change the data without violating integrity. "Cryptographic" integrity of the data can be achieved with signing of the data. Another way to achieve this is to create a secure hash of the message (data). The secure hash will then be signed using a private key, for instance. This is used in digital signatures. Both the message (data) and the signed hash are then sent to the receiver. The receiving part creates a secure hash of the message and compares this to the decrypted signed hash value. If these match then the message has not been tampered with.

Sometimes it is desired that the message is both encrypted and signed. It is important in most cases to first sign the message and then encrypt it and not the other way around. This is because if the message is first encrypted and then signed, the receiver might not be sure that the sender actually knows what is encrypted and that the sender only has signed the message [5].

Information disclosure (attack on confidentiality)

Information disclosure is an attack on confidentiality and relates to the fact that sensitive information can be revealed or accessed by an unintended part. To prevent this, information can be encrypted to provide confidentiality. The information or data can be encrypted using an asymmetric or symmetric cipher. A cipher is a cryptographic algorithm used for encrypting or decrypting.

Unauthorised access (attack on authorisation)

Unauthorised access is an attack where a user or process gains access to systems or recourses to which he or she does not have permission. To prevent such attacks the system should require everybody that accesses the system to authenticate themselves in front. Thereafter check Access Control Lists (ACL) or capability lists to see if the authenticated user has the right permission to access the resource.

DoS/DDoS (attack on availability)

Denial of Service (DoS) and Distributed Denial of Service (DDoS) are attack types that are launched against a server or service in order to render it inaccessible or unavailable for legitimate users. Distributed Denials of Service usually refer to attacks on availability where a number of distributed machines are co-operating in the attacks, possibly distributed across the world. Handling attacks such as these are usually separated into two categories: Detection and Prevention. Detection could be achieved by having statefull firewalls or sensors listening to the network traffic to spot anomalies. Prevention is also a difficult task because it should not prevent normal users from accessing the services. One solution to this is to use multiple servers running the same services so that one creates load balancing and redundant services. An attack will be less successful if the goal is to make the services unavailable to users, should there be a lot of redundant and replicated services. If one server goes down there will be others that can handle the traffic. The prevention could also try to filter out the normal traffic from the attacking party, although this is very difficult.

There could also be another type of attack on availability which may not be intended as an attack. One such attack is where a legitimate user, or user process or service consumes much of the processing power or network bandwidth due to a bug or software error. These kinds of attack could be very hard to prevent as the attacker or erroneous software are positioned on the inside and are usually trusted and normally have access to these resources.

Spoofing identities (attack on authentication)

Spoofing of identities are attacks where an attacker claims to be someone else and use their identity. These are attacks on authentication where the attacker tries to be authenticated as someone else. The attacker should not be able to spoof identities if the authentication mechanisms work. Spoofing identities in a computer network could sometimes be done by spoofing the sender or receiver address. This is possible because the sender address is not authenticated. Therefore one should not solely depend upon the sender address to authenticate a user or process. This may not only be the case the first time a message is sent, but the address of an attacker could be faked or spoofed at any time. Therefore messages originating from one person or process should always include a Message Authentication Code (MAC) or be signed in such a way that the receiver knows that it came from the right person regardless of the sender address. Message integrity or message origin integrity is very important to prevent spoofing attacks in a replay and “man-in-the-middle” attacks.

Elevation of privilege (attack authorisation)

The attack type called elevation of privilege is an attack on authorisation. This could be an attack similar to attacks to gain unauthorised access except that this could be a legitimate user with some permission who gains more rights than he should. Role Based Access Control (RBAC) has mechanisms to prevent separation of duties where one user cannot be attached to two roles which have different interests at the same time.

Repudiation (attack on accountability)

The attack or threat referred to as repudiation is the attack where a user claims not to have ordered or sent a message. To prevent non-repudiation one could ensure that the sender signs the messages, such as a digital signature or digital contract. In this way the only one who could have sent the message must have been the sender, since he is the only one who would be able to sign a message using his signature or private key. Auditing and logging could also be used but these would not prove anything because they cannot authenticate the sender alone.

Other issues

The last attack or threat is the attack type referred to as social engineering. This is a type of attack where the attacker uses personal or social knowledge and skills to acquire access to the computer system. The reason this is so successful is because of the human factor. If passwords are too difficult to remember then people stick a Post It note on the computer monitor showing the password. This would of course render the password useless if someone has physical access to the computer and monitor. Another weakness is the fact that there are often weak routines for resetting passwords for users. If he knew the username, an attacker could many times just ask for a new password by claiming that his account was locked because of too many password attempts. Many departments do not have guidelines that would prevent anyone from giving out a password over the telephone without being absolutely sure that the person on the other end is who he claims to be.

There are also other types of attacks that require physical access to a computer or resource. For example, if someone wants to access something on a computer hard drive he could start and boot the machine with another operating system from a CD-ROM. A hard drive can also be removed from a computer if it is not locked. A common practice for securing computers (especially servers) is therefore to physically lock the computer and the server room to prevent unauthorised access.

2.12 Summary

In this chapter, key security concepts have been described. Symmetric and asymmetric encryption algorithms have also been explained. Other issues that have been covered are signing, certificates, access control, and computer and network attack types. The next chapter will describe relevant frameworks and security protocols for this thesis.

3 Security in relevant frameworks and protocols

3.1 Introduction

This chapter will give a short description of distributed systems. It will also describe security protocols and relevant frameworks for this thesis such as Parlay [54,55], Parlay X [54] and NTT DoCoMo [57]. All protocols and applications mentioned in this chapter are of interest for the remainder of the Master Thesis report. This chapter will also have a short description of the security APIs in Java.

3.2 Distributed systems

The way of organising data in a distributed system has gradually become a relevant way of managing data between many computers. There are numerous definitions of what a distributed system is. Information on a distributed system can be gathered from [58]. This book defines a distributed system as follows "*a distributed system is a collection of independent computers that appears to its users as a single coherent system*". A truly distributed system is when the system looks and acts as a single processor system. This means that the system should hide the fact that the resources are distributed across multiple computers. The system is said to be transparent when the system appears as one application for the user. The system must also be scalable. This system should also make it easier for users to access remote resources.

Security in a distributed system

Distributed systems constitute new challenges for security mechanisms. Much of the information flow goes through insecure telephone- and computer networks. In theory, anyone can eavesdrop on messages passing by, alter or corrupt messages, replay messages or launch a denial of service (DoS) attack. Within an organisation some of the security aspects are easier to control. Several machines working together could perhaps have been given a dedicated network or subnet, physically separated from normal networks or by a firewall. Given this configuration less effort could be used in securing the network traffic if no one has access to the network. The traffic need not necessarily be encrypted, but when all information is flowing through an insecure public network this may become another matter.

Security in a distributed system shares much of the same challenges as security in a traditional network environment. Information transmitted over open air or through Internet and other networks is vulnerable to attacks. These attacks can target confidentiality, authentication, authorisation or integrity. The attacks can be replay attacks, ”man-in-the-middle” attacks and DDoS attacks.

Security challenges in distributed systems differ from traditional systems in that the complexity of the system increases. In turn this makes the security considerations more complex as well. In a distributed system many components co-operate and share information; these components must be able to exchange data without disclosure of sensitive information and without loss of data integrity.

3.3 Security protocols

Achieving secure protocols is an ongoing challenge. Errors are often detected after a while when a new security protocol is proposed. The security of the protocol depends on in what context it is used. It is hard to prove that a security protocol is bullet proof. Today there exist

many methods for verifying a protocol. The use of these methods is an entire discipline and will not be covered in this thesis. The main challenge for security protocols is to achieve authentication of the involved parties and to securely exchange keys. In this chapter some of the well-known security protocols are described.

Needham Schroeder Lowe

Needham-Schroeder-Lowe [61] is a key-exchange protocol and is an improvement on Needham-Schroeder [60] . In this protocol, as shown in Figure 3.3-1, “a” is Alice and “b” is Bob. This protocol assumes that the public keys are known in advance.

The steps in the protocol are given below:

1. Alice uses Bob’s public key to send an encrypted message containing the nonce “na” and her Id “a”. The nonce is a randomly generated value.
2. Bob decrypts the message and generates his own nonce “nb”. Bob then creates a message containing the nonce he got from Alice, “na”, his own nonce, “nb”, and the Id. Bob then encrypts the message using Alice’s public key and sends the message.
3. Alice decrypts the message and verifies that the nonce “na” returned from Bob is the same she sent earlier. She has then authenticated Bob. She then encrypts Bob’s nonce “nb” and returns the message to Bob. When Bob receives this message he has authenticated Alice.

In the Needham-Schroeder-Lowe protocol both parties authenticate each other. This is called mutual authentication. The nonces used in the protocol should be truly random, which is hard to achieve in practice. Instead the nonces are pseudo-random numbers which could lead to an attack [106].

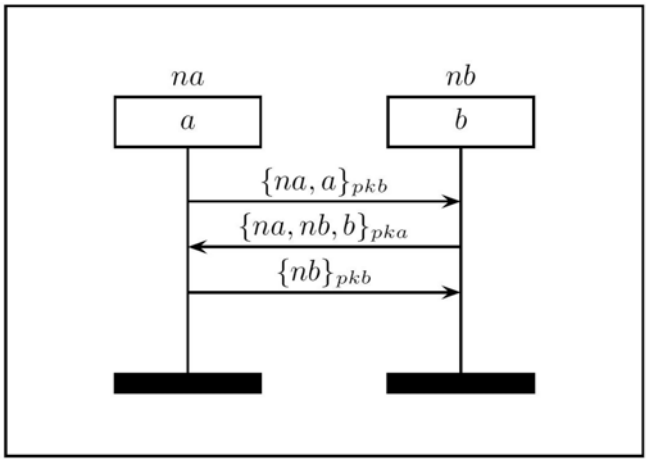

Figure 3.3-1 The Needham Schroeder Lowe protocol [62]

The Needham-Schroeder-Lowe protocol is generally considered secure [48,49]. However, in the paper [50], the authors claim there are weaknesses in the protocol if the nonce is put after the id, as it will then be possible to crack the messages. Their conclusion is that if the nonce is put in front it would solve the problem. In [51] they claim that under certain conditions an attack may be performed on the Needham-Schroeder-Lowe. But these attacks are highly theoretical.

Diffie Hellman

The key exchange protocol Diffie Hellman was developed in 1976 Diffie and Hellman [64]. The protocol is developed for two parties with no earlier knowledge of each other's secret key. A variant of Diffie Hellman called Station-to-Station (STS) [68] was introduced in 1992 by Diffie, van Orschot, and Wiener. This variant of the protocol was introduced to repair the "man-in-the-middle" attack experienced with the first version. STS has also some weaknesses concerning the knowledge of a key. A party cannot use encryption to verify that they are aware of the key. If they do verify, the system has a known weakness. This problem was solved with STS-MAC. But some new vulnerability was also found on this variant. The flaws were related to "Unknown Key-share Attack" [66] which is an attack where the entity "A" is finished with the key exchange protocol. "A" believes that it shares a secret key with "B". "B" actually believes that its key is shared with another entity "E" (this is not "A"). An improvement of the Diffie Hellmann protocol can include certificates and timestamps.

Secure Socket Layer and Transport Layer Security

Secure Socket Layer (SSL) [68] was introduced by Netscape as an encryption protocol for securing on-line transactions between a client browser and a web server. SSL is layered on top of TCP/IP and is often used in conjunction with HTTP (HTTPS). Transport Layer Security (TLS) [69] protocol is the successor to SSL. The term SSL is used to indicate both SSL and TLS, as these terms are often discussed together in literature. SSL can authenticate both the server and the client although the latter is often left as an option. The authentication uses standard public key cryptography algorithms, such as RSA, which are negotiated in the initial (establishment/start up) process. SSL can also provide data integrity on messages during a session. In the start up process the two parties involved agree on a session key which is a symmetric key. This session key is valid for the entire session and is used to encrypt data interchanged, providing confidentiality. Since SSL (and TLS) can be used in conjunction with HTTP, normal SSL can be used to secure normal HTTP traffic.

The SSL protocol is described in [68] and a short description is available from [57]:

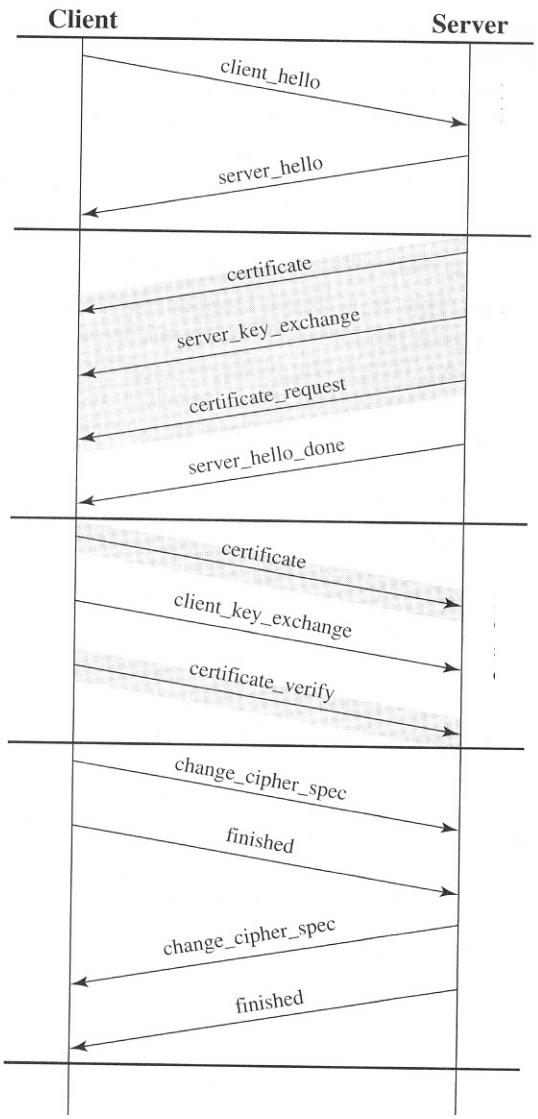

Figure 3.3-2 The Secure Socket Layer (SSL) protocol [8]

- The client first sends a CLIENT-HELLO message to the server (could be a web server) with its name "C", which cipher (cryptographic algorithm) it supports (cipher specification) "SPc", and a random challenge "Rc".
- The server then responds with a SERVER-HELLO message containing its name "S", which cipher it supports (cipher specification) "SPs", a random connection ID "Rs", its public key Ks and its public key certificate "Cs".
- The Client then checks the server's certificate "Cs" and verifies it against its own root certificates, to see if it is signed by known CAs, meaning that Ks belongs to " S". The client will then generate a Pre-Master-Secret-Key (PMSK) which is used to derive three other symmetric keys. The client then sends a ClientKeyExchangeMessage which includes the PMSK with RSA if this algorithm is chosen
- The client sends a ChangeCipherSpecMessage and sends a finish message.
- The server sends a ChangeCipherSpecMessage and then sends a finish message.

One of the drawbacks of SSL is that it does not handle Certificate Revocation Lists. That is, if a certificate is revoked a client has no way of knowing. Some weaknesses have also been

found in the early implementation of the SSL protocol, especially when it comes to the key length. This was because of the US export regulations on strong cryptographic algorithms. The newest TLS version 1.2 was released 26 April 2006. In version 1.2 many flaws e.g. the flaws concerning CBC were fixed. There has yet to be made an analysis of protocol version 1.2, which means we are not aware of any flaws or weaknesses.

Kerberos
Kerberos [8,70] is a protocol used for computer networks to authenticate the involved parties and secure the communication. Kerberos is based on a server-client model which requires a trusted third party. The protocol is based on symmetric cryptography. Kerberos was originally developed by Massachusetts Institute of Technology (MIT). The protocol based on the Needham-Schroeder Symmetric Key protocol [60] system uses a termed Key Distribution Center (KDC). The KDC consists of two logically separate parts which are an Authentication Server (AS) and a Ticket Granting Server (TGS). Kerberos has a key database with a secret key. The secret keys are used by a client and a server both sharing the same secret key called session key.

Microsoft stated in a bulletin dated 9 August 2005 that Windows domain controller using Kerberos, could have a flaw when a specific message is sent to it. The result could be that the authentication service that authenticates users in an Active Directory domain could stop responding. This bulletin was posted at the following link [72]. The new edition of Kerberos is version 5, Release 1.4.3., it was released on 16 November 2005. We have not found any other paper indicating other flaws in the new version of Kerberos.

3.4 Relevant frameworks

In this chapter the relevant frameworks are described. The Parlay/OSA and Parlay X frameworks are essential for this thesis because the thesis concerns a service creation framework which uses Parlay for offering telecommunication services. The security issues of Parlay and Parlay X will therefore be explained. The Parlay X uses Web Services and therefore Web Services Security (WSS) will be described. NTT DoCoMo i-mode is another framework which offers telecommunication services. This chapter provides an overview of the services in these three frameworks.

Parlay/OSA and Parlay X

Parlay/OSA [73] is a specification of an API that developers can use when creating applications and services for the telecom networks. The Parlay/OSA emerged as a result of the convergence between the telecom and computer domains. Parlay/OSA is developed for existing and future communication networks. Parlay/OSA specifications are maintained by ETSI [74], the Parlay Group [73] and the 3GPP [75]. The telecommunication operators have been sceptical to allowing third party developers to access their networks. The services offered in the telecommunication networks are critical and the network operators are worried about the security concerning third party developers. Therefore the security in frameworks such as Parlay must be sufficient.

There is also a Parlay/X specification [77], which describes interfaces to Parlay through the use of Web Services [78]. Web Services is an XML based middleware technology, as described in [78]: *"A Web Service is a software application identified by a URI [IETF RFC 2396], whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML*

based messages via Internet-based protocols" [79]. The Parlay X specification does not offer as wide a range of services as the Parlay/OSA specification.

Parlay/Parlay X allows the applications to interact to the already existing telecommunication services such as SMS, MMS, Location Information, User Status, Call Control and User Interactions. With the Parlay/Parlay X framework the underlying telecommunication technologies are irrelevant, as shown in Figure 3.4-1. There has been a rapid increase in the number of service providers and they use the architecture of telecom networks.

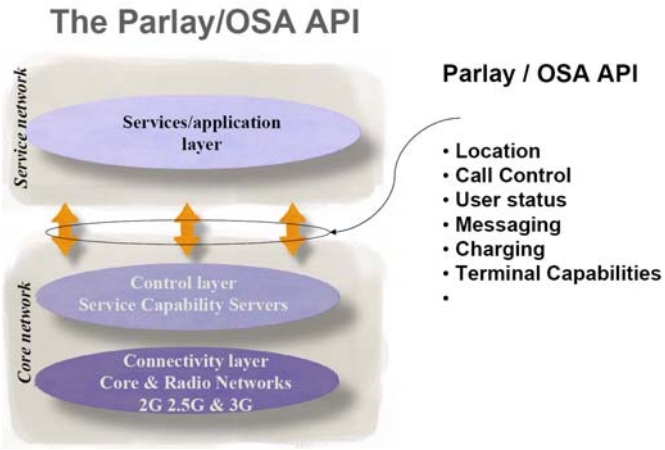

Figure 3.4-1 Parlay/OSA [80]

Security in Parlay/OSA

Distributed applications can make use of Parlay/OSA, which means that security issues are important. One of the key security objectives in Parlay/OSA is accountability. Confidentiality and integrity are also important in Parlay/OSA security [55]. Protection against unauthorised use is also a part of Parlay/OSA.

Authenticating and securing data is achieved by using the Trust and Security Management (TSM) Protocol [115]. The Parlay gateway offers Service Capability Features (SCF), which are the functionality accessible through the standardised OSA interface. The framework functions and the Service Capability Services (SCS) are included in the Parlay Gateway. A critical step in the security setup is the authentication between SCF and the Gateway. The main steps of the TSM protocol are shown in Figure 3.4-2:

- Initiating authentication
- Selecting authentication mechanism
- The client and the framework interface authentication with each other
- Requests an access session
- The access interface is used to negotiate the signing algorithm

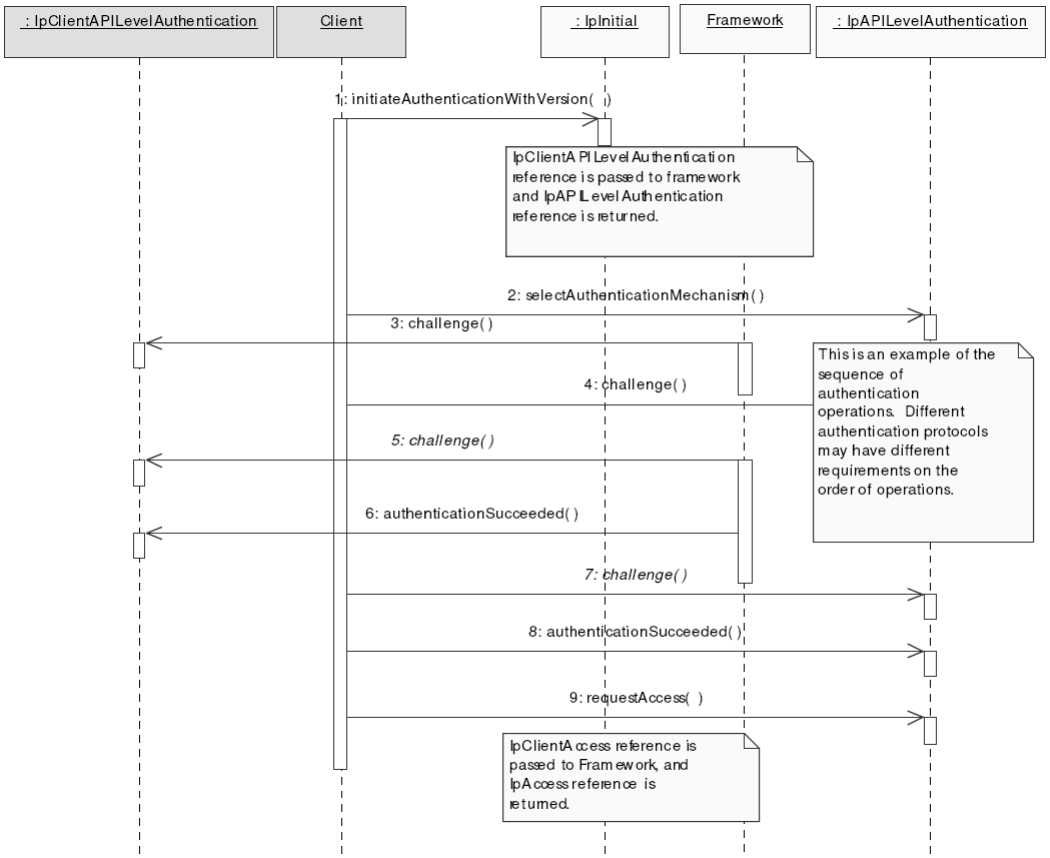

Figure 3.4-2 TSM protocol [85]

Applications must perform several interactions with Parlay/OSA interfaces in order to get references to the required SCF. Applications must first be authenticated before they get a reference to an SCF. Parlay will then verify that the application is authorised to be used according to a subscription profile. As a final stage the agreement is digitally signed and Parlay returns a reference to the SCF.

When authenticating in Parlay/OSA, the system needs to identify an entity. Servers and gateways are authenticated in the context of application-defined Access Sessions. To be able to access Parlay/OSA interfaces, a temporary Access Session must be established. Parlay/OSA framework interfaces have a number of objects which are associated with Access Session. An authenticated peer entity could have a number of interface objects associated with each Access Session. These objects are created and are only valid during the lifetime of an Access Session. The interfaces are non-sharable objects which are only used by a selected authenticated peer entity. Parlay/OSA clients make a challenge/response with the Parlay/OSA gateway. The challenge/response is exchanged at the CORBA layer. CORBA is the middleware used when connecting to the Parlay/OSA gateway. The IETF RFC 1994 [81] “PPP Authentication Protocols - Challenge Handshake Authentication Protocol” (CHAP) deal with the challenges and responses. This is a protocol originally used for dialup lines. The protocol is also unilateral. Since the protocol uses two unilateral authentication processes it is possible that a “man-in-the-middle attack” can occur. The authentication procedure has a flaw when a remote OSA/Parlay application Server is invoked in an Access Session [55,82]. Immediately after this process the system is vulnerable and not protected against malicious attacks. The vulnerability can be reduced by frequent re-authentication, but not totally removed. In this case sufficient intelligent attacks will still be able to use the security

weakness. An Access Session may be broken into by an intruder either by guessing or stealing the secret object reference of a Framework interface attached to an Access Session. In this case the intruder masquerades as the authenticated and legitimated entity.

Authorisation in Parlay/OSA is based on Access Sessions. Privileges are assigned to a given authenticated entity when authorisation is granted. Any unauthorised use is prevented by Access Control. When talking about unauthorised use it means prohibited use of a resource and service. To enforce the right granted by authentication, Access Control is utilised. Unintended disclosure of information can be protected with confidentiality.

Communication between a Parlay client and the Parlay gateway goes through insecure networks, which requires confidentiality and integrity protection. The communication is made using the CORBA Internet Inter-ORB Protocol (IIOP). The use of IIOP raises some problems when implementing a firewall. The firewall will not be able to let IIOP pass securely through the firewall. The solution to this problem is a proxy, which will let IIOP pass securely. Parlay has one way of securing the connection using either SSL/TLS, IPsec (security protocol) or both.

In the security paper by Corin et al. [85], they analyse the TSM protocol used in Parlay/OSA. In this paper they analyse the TSM protocol using formal methods. They used the program CoProVe [86] to verify the protocol. This tool is especially designed for model checking security protocols. They have also found a few flaws and weaknesses in the security protocol. The flaws found regarded the authentication and that an intruder could impersonate a client and in this way get access to the framework. In this, an intruder could also use replay attack to make the framework perform a specific authentication mechanism. This shows that the TSM has some flaws which can be exploited.

Security in Parlay X

In Parlay X security [87] is handled differently because it uses web services, [78] as shown in Figure 3.4-3 below. Security can be implemented using HTTPS (encrypted web traffic) or Virtual Private Network (VPN). But those methods only protect the data between the client and the web server or VPN gateway. They only protect the confidentiality of the data and not the integrity. The data integrity could be provided by XML, which has support for digital signatures. Authentication assertions on the XML message level are given by the Security Assertion Markup Language (SAML). For security see under WSS (also earlier called WS-security).

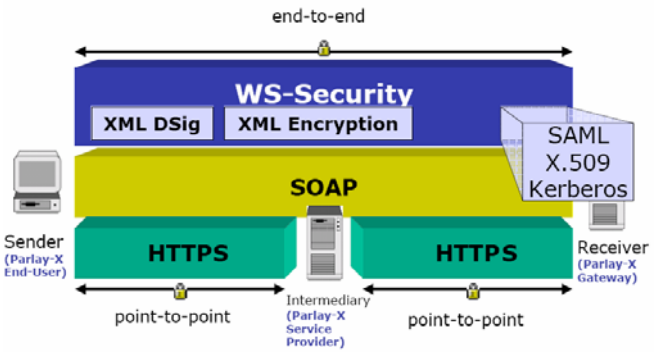

Figure 3.4-3 Security of Web Services in Parlay X [87]

Web Services

Web services are defined by the World Wide Web Consortium (W3C) [92] as the programming interfaces used on the World Wide Web for application to application communication. The interfaces to Web Services are described in a Web Services Description Language (WSDL) [93] specification. Web Services are described using WSDL in a XML format that is a Machine processable format. Access to Web Services can also be made by using Simple Object Access Protocol (SOAP) [93]. SOAP is a protocol for exchanging messages in XML format. In the WSDL description the Web Services are divided into an abstract and a concrete part.

Web Services Security

Web Services has a protocol that provides security which is called Web Services Security (WSS). WSS version 1.0 standard was introduced on 19 April 2004. Today the official name for WS-Security is WSS. Below we will list the basic building blocks for WSS and their relation to the security concepts [93]:

- Integrity is solved by using Web Services Description Language
- Extensible Access Control Markup Language fulfils the authorization concept.
- Authentication and authorization is fulfilled through Security Assertion Markup Language
- Channel confidentiality is ensured through Secure Socket Layer (SSL)
- To achieve granular confidentiality XML-Encryption can be used
- The granular non-repudiation concept is achieved through XML-Digital Signature

A known flaw is the possibility of executing an XML rewriting attack [94] on a web service which can lead to a replay attack. An XML rewriting attack is an attack where the attacker modifies the XML file. There is also a vulnerability related to the password, which means that the password can be faked. The SAML may have some vulnerability [96] when using real Web Services; the reason being the possibility of interference from different protocol layers. Another problem related to SAML 1.0 is when SAML artefacts leak over an insecure channel. SAML V1.1 has vulnerability because the security constraints are not specified for the response from the identity consumer to the browser. SAML 2.0 has also some weaknesses mentioned in this paper [97].

NTT DoCoMo i-mode

NTT DoCoMo [88,57] is one of Japan's largest telecommunication companies and offers mobile and wireless networks, mobile equipment/handsets and services for mobile customers. In 1999 NTT DoCoMo introduced i-mode [88] as an Internet access service for Japanese mobile customers. I-mode enables mobile phone users to browse the web, as well as send and receive e-mail. I-mode offers various services such as banking, booking of restaurants and train tickets, checking weather conditions, maps and location based services. The i-mode uses a packet switched mobile access network. I-mode also offers voice transfers as well as data transfers and users can search the i-mode directory and place calls.

The security aspect of NTT DoCoMo i-mode is not easily analysed because it is a closed system and uses many proprietary protocols. The Unofficial Independent i-mode Frequently Asked Questions (FAQ) web site [90] has listed a number of security issues with i-mode. This is one of very few texts discussing security in i-mode and it has listed a few security issues, such as the radio link between an i-mode phone and the cellular base use proprietary protocols and encoding only known to NTT DoCoMo. In the FAQ site it also lists a few other potential security issues but does not give any solutions or more detail. There is no way of knowing if

these security issues are genuine threats, since it is not explained in the FAQ and the fact that NTT DoCoMo keeps the security aspect of i-mode hidden from public knowledge. This is usually referred to as security by obscurity, and is not highly regarded by the security community professionals. Security by obscurity means that the secret and security lies in the algorithms and internal structure of a system. The followers of security by obscurity believe the system to be secure as long as no one knows how it works, but this has failed many times through history and security protocols and algorithms open to the public are shown to be more secure than when hidden. In the paper [90] the authors discuss some security issues of i-mode. They conclude that there is no way of knowing if the i-mode is secure since there is no public information on the subject and algorithms and protocols have not been formally tested and analysed by others. NTT DoCoMo i-mode uses SSL but how the implementation is done is not publicly known.

3.5 Security APIs in Java

Java has support for various security mechanisms as shown in Figure 3.5-1 The Java extensible security architecture and its core APIs [2]. These are divided into different APIs.

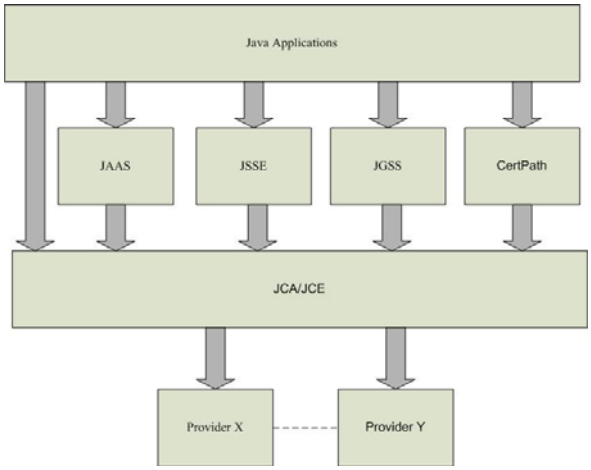

Figure 3.5-1 The Java extensible security architecture and its core APIs [2]

These APIs give a wide area of security support such as cryptography, certificate management, authentication and authorization, secure communication, and other security mechanisms. This section is based on the information provided in the book [2]. The elements in Figure 3.5-1 will explained in the following sections.

Java Cryptography Architecture (JCA) [99,2]: JCA gives cryptography services and algorithms which support digital signatures and message digests. The engines in JCA are:

- MessageDigest: Produces a hash value for a given message
- Signature: Develops a digital signature of a document
- KeyPairGenerator: Generates a key pair
- KeyFactory: Imports and exports keys and translates keys
- KeyStore: Controls and saves different keys and pairs
- SecureRandom: Exists for random numbers to be used in cryptography
- AlgorithmParameters: Controls encoding and decoding parameters for specific cryptography algorithm

- AlgorithmParameterGenerator: Develops a set of parameters demanded by a cryptography algorithm
- CertificateFactory: Produces a public key and a revocation list for certificates
- CertPathBuilder: Controls the certificate chain between certificates
- CertStore: Controls and save certificates and its' revocation list

Java Cryptography Extension (JCE) [101,2,98,98] JCE 1.2 was originally made as an extension to the JCA available in the Java 2 platform, and according to U.S. export regulation the JCE 1.2 was not to be exported outside the U.S. and Canada. This version contains a support for encryption and decryption operations, secret key generation and agreement, and message authentication code (MAC) algorithms. The 1.2.1 and the maintenance version 1.2.2 are allowed to be exported outside the U.S. and Canada and these versions have restrictions as opposed to 1.2 which is a full version. Today the USA no longer has restrictions on the strength of the exported cryptographic algorithms. Some limitations are still maintained in Java 2 SDK, v 1.4 and later. A stronger and restricted cryptography has been permitted because the unlimited strength cryptography is not allowed in every country. These restrictions are maintained in some countries because they have import control restrictions. For those countries where the restriction does not apply, a security policy file is available that gives the unlimited strength cryptographic algorithms.

JCE has the following engines:

1. Cipher has operations that are used when encrypting and decrypting
2. KeyGenerator is used to generate secret keys
3. SecretKeyFactory is equal to the JCA KeyFactory except that it only works on secretKey instances.
4. MAC delivers message authentication code functionality.
5. KeyAgreement includes a key agreement protocol which is used to dynamically produce a shared secret for multiparties.

Java security APIs:
CerPath (Java Certification Path API): This API provides support for checking, verifying and validating the authenticity of certificate chains.

Java Authentication and Authorization Service (JAAS): A user or a device can be verified by the mechanisms provided by this API. The API will determine the trustworthiness of the user or the device, and, depending on its identity, access rights and privileges will be allocated. The security provided by JAAS makes it easier to adopt pluggable authentication mechanisms and user-based authentication.

Java Secure Socket Extension (JSSE): Integrity and confidentiality in JSSE are maintained by using SSL/TSL protocols to secure the communication. JSSE is socket-based which means that it typically uses TCP connections.

Java Generic Secure Services (JGSS): JGSS API gives support for authentication mechanisms that a third party can use. In JGSS there is support for Kerberos. When implemented this support leads to a system which needs to ask for a password. JGSS is token-based, meaning that communication can be made through TCP sockets, UDP datagram or other types that tolerate JGSS-API built tokens. The advantage JGSS has, because it is a token-based API, is that it can explicitly encrypt just one or some or all messages.

Provider: Both JCE and JCA have the ability to implement providers as shown in Figure 3.5-2 Java Service Provider Interface [2]. This is done in the following way: First the application calls the JCE/JCA API classes. Then the JCE/JCA SPI is invoked by the API. Finally the SPI invokes the provider.

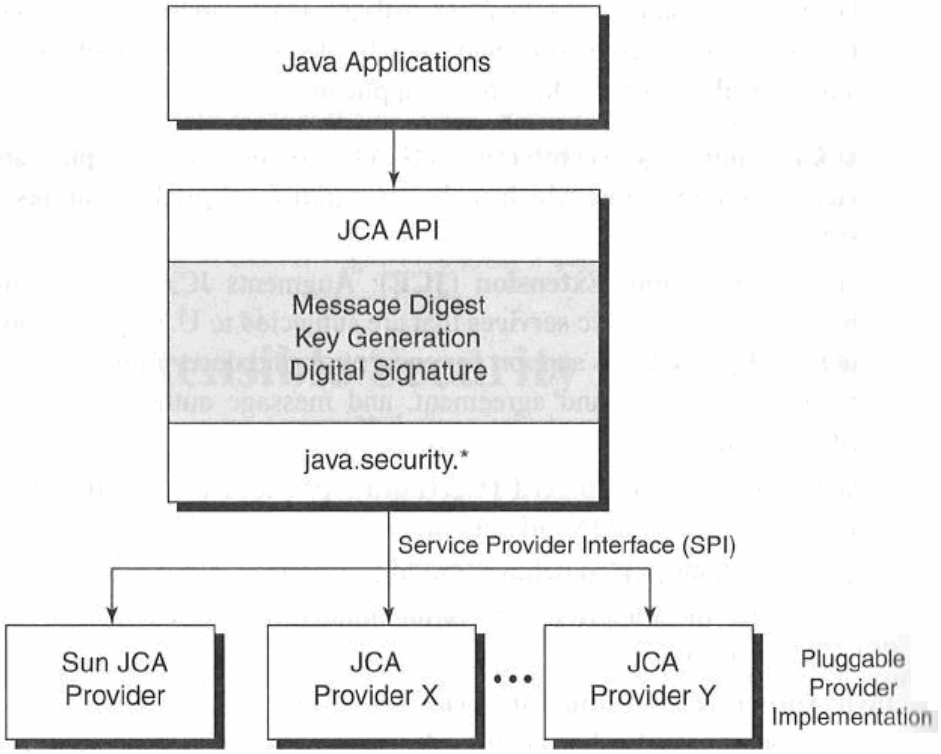

Figure 3.5-2 Java Service Provider Interface [2]

3.6 Summary

In this chapter security aspects related to a distributed system have been explained. All protocols and applications mentioned in this chapter are of interest for the remainder of this Master Thesis report. It has been especially focused on Parlay because it is an important part of Ericsson's service creation framework, called ServiceFrame. The next chapter will explain the new concepts of UML 2.0 which is used to describe ServiceFrame. The Serviceframe's three layers will be explained.

4 Ericsson's service creation framework

4.1 Introduction

This chapter describes Ericsson's service creation framework, which contains ServiceFrame, ActorFrame and JavaFrame. UML 2.0 [112,13] is the modelling language used to describe the three layers. Ericsson's service creation framework is primarily a framework for developing telecommunication services. JavaFrame is a Model Developing Kit (MDK) for Java development of state machines which communicates asynchronously. ActorFrame defines concepts such as actors and agents, which use the state machine concept from JavaFrame. ServiceFrame can be used by third party developers to develop components that use the concepts from ActorFrame.

4.2 UML 2.0

UML 2.0 plays an important part in understanding of ServiceFrame. UML 2.0 is a modelling language used to model object oriented systems and standardised by the Object Management Group (OMG) [83]. New in UML 2.0 is the merging of SDL & MSC 2000 and UML 1.4. The connector, port and part are three new aspects of UML 2.0. More information on UML can be found in [106,108]. Figure 4.2-1 shows a typical example of new concepts introduced. The connector may be connected to two or more elements each holding a set of instances. A connector does not necessarily need a port to become connected to parts. The port is only used to encapsulate the part so as to hide its features. The decision whether or not to use a port is handed over to the designer of the system. The part concept is related to instances which are owned by a containing classifier instance. The state machine may have inner states which it enters depending on the signals received.

UML 2.0 ports, parts and connectors

The Figure 4.2-1 shows an example of the port, part and connector concepts in UML 2.0. This simple component of type "Car" has two inner part types called "Wheel" and "Engine". The connector between the "Engine" and two "Wheel"s is called "axle". The "Engine" called "e" has a port "p" that connects the "axle" to the "rear" components of type "Wheel". The number "2" after "Wheel" denotes that there are two instances of type "Wheel".

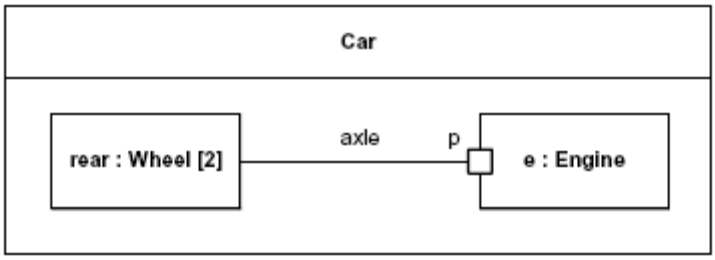

Figure 4.2-1 Example of ports, parts and connectors in UML 2.0 [112]

Composite State in UML 2.0

Figure 4.2-2 below shows the State machine concepts in UML 2.0. The state machine has an initial and an end state denoted by the two circles. Transitions from one state to another triggered when the state receives a signal or message. The transition can have a guard condition which could be a logical expression that must be true. In the state diagram three

main concepts are given. A simple state is a state without inner structure, i.e. it does not contain inner states. A composite state is a state that can have inner simple states or inner composite states. A submachine state is a state that references to another state machine, i.e. how the state machine is constructed is given in another state machine diagram.

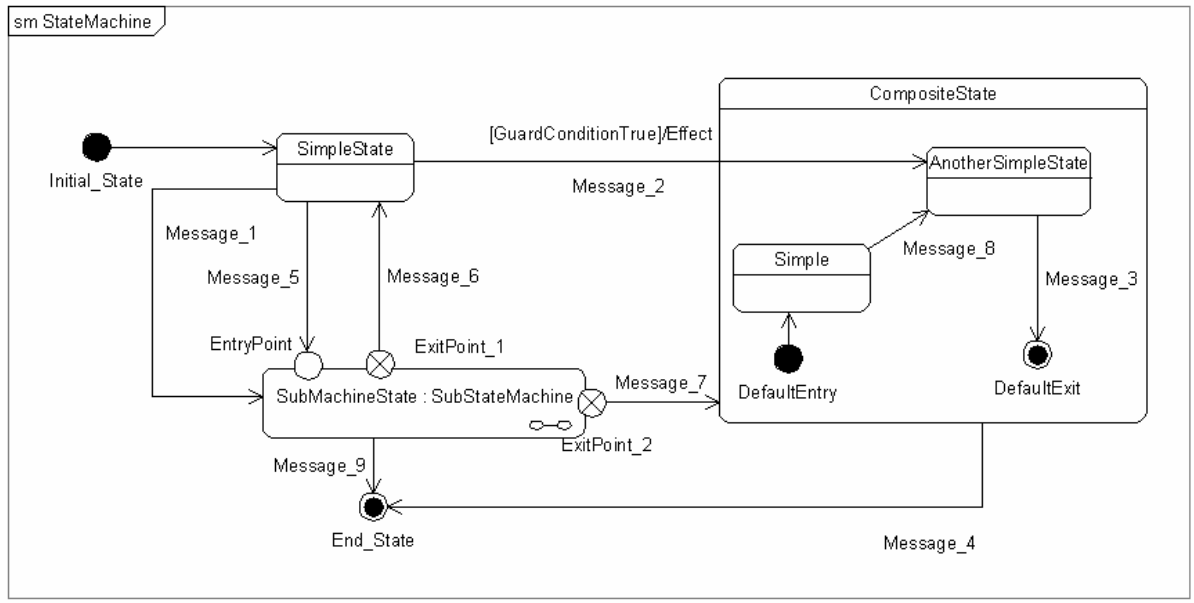

Figure 4.2-2 Example of a state machine

In Figure 4.2-2 the submachine state called "SubMachineState" references the sub state machine called "SubStateMachine". A "submachine state" is the state within a state machine that references another state machine called a "sub state machine". The submachine has one entry point called "Entry point" and one exit point called "Exit point". In Figure 4.2-2 the composite state "CompositeState" has an inner state called "AnotherSimpleState". "CompositeState" has an entry point called "EntryPoint" and two exit points called "ExitPoint_1" and "ExitPoint_2". The names are just for illustrating the point, i.e. the entry and exit points can be given other names. Figure 4.2-2 shows that after receiving the signal "Message_7" in the "ExitPoint_2" the state machine enters the "CompositeState" through the default entry point. An arrow that goes directly to a sub machine state or a composite state means that the state machine enters through the default entry point. An arrow from the composite state or a submachine state denotes an exit through the default exit point. If the default entry or exit points are not to be used, an arrow must go to one of the specified entry points or from one of the specified exit points. This means that there are two ways of entering and two ways of exiting a composite state, either through a default entry or exit point or through a specified entry or exit point. "Message_9" and "Message_4" are coming from a default exit point, whereas "Message_6" is from the defined exit point "ExitPoint_1".

Figure 4.2-3 below shows the sub state machine "SubStateMachine", which is another state machine, referenced by the state machine "StateMachine" in Figure 4.2-2 above. This means the SubStateMachines are reusable components that can be referenced by many state machines. Figure 4.2-3 below shows that "SubStateMachine" is just like any other state machines and can have simple states. It can also reference other sub state machines or have composite states, although this is not shown in Figure 4.2-3. The "InitialState" and the "FinalState" are the default entry point and default exit respectively. The figure also shows

that the state machine has one specified entry point called “EntryPoint” and two exit points called “ExitPoint_1” and “ExitPoint_2”.

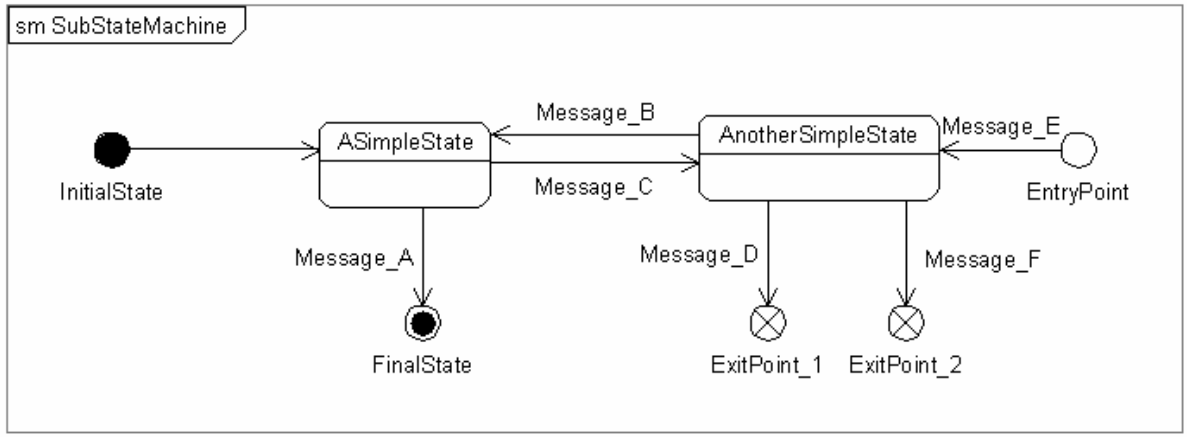

Figure 4.2-3 Example of a sub state machine

4.3 Overview of the ServiceFrame framework

ServiceFrame is a framework with three layers. The bottom layer is the JavaFrame layer where the concepts of state machines and composite states are defined. This layer allows asynchronous communication. The layer on top of JavaFrame is the ActorFrame layer. In ActorFrame, concepts such as actors and agents are defined. The ServiceFrame layer is the topmost layer and uses the concepts from ActorFrame to create service components which can be used by third party developers. The three layers are modelled using UML 2.0.

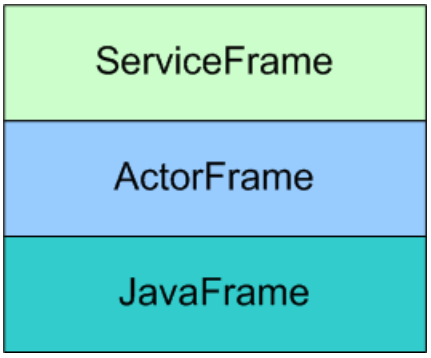

Figure 4.3-1 The ServiceFrame layers

4.4 ServiceFrame

Ericsson has foreseen the increased interest in developing services for the telecom network, and has created a prototype for an application framework for services called ServiceFrame. Services like creation, deployment and execution are a part of the Service architecture. The ServiceFrame is explained in more detail in the ServiceFrame Whitepaper [110]. “*ServiceFrame is an application server in the service network. It provides functionality for communication with users connected through different types of terminals, such as phones, PCs or PDAs. It also provides access to the network resource through the OSA API which enables services to set up phone calls between users* [111]”. The new alternative which

ServiceFrame introduces is the combination of traditional telephone services, multimedia services, internet services, information services, message services and location and context aware services. The system also offers hybrid services, and establishes services through heterogeneous communication such as PSTN, IP, GSM and UMTS. The hybrid way of thinking introduces the aspect of communication between different networks. In ServiceFrame different networks do not address any problems since the ServiceFrame architecture is network independent.

The components in ServiceFrame communicating with Parlay/OSA are known as NRG edges. NRG is Ericsson's Parlay gateway [76]. Ericsson has a software developer's kit (SDK) [113] for the NRG Parlay gateway. This SDK provides an API that resides on top of the Parlay/OSA API, offering a higher level of abstraction. The API enables programmers to use standard Java interfaces rather than connecting directly to Parlay/OSA using CORBA [83]. An NRG simulator is also included which simulates the telecom networks. ServiceFrame is the highest abstraction level and can be used by application developers creating services. ServiceFrame is an application of ActorFrame, and ActorFrame itself will be discussed in the next section. The NRG Edges are defined in the ServiceFrame level. The following figure describes how ServiceFrame itself can be considered an actor with the other actor as inner parts. The inner actors have ports that are linked to each other through connectors. There are also actors connected to the outside world through ports. These actors are called edges at the ServiceFrame layer.

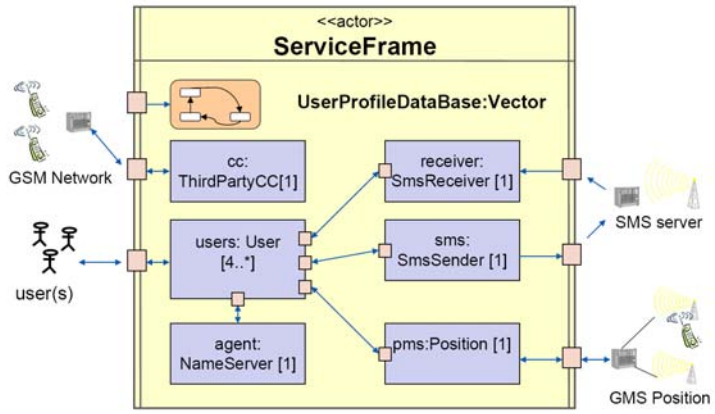

Figure 4.4-1 ServiceFrame example [109]

4.5 ActorFrame

An important layer in the ServiceFrame is ActorFrame. ActorFrame is an application of JavaFrame. JavaFrame will be discussed in the next section. In ActorFrame we have actors which communicate with each other through actor messages.

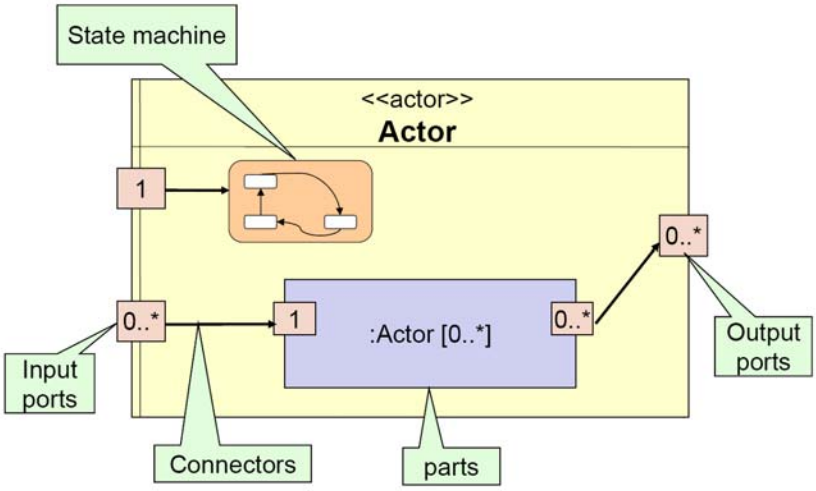

Figure 4.5-1 Actor with port, parts, connectors and a state machine [109]

Figure 4.5-1 shows an Actor with port, parts, connectors and state machine explained in the UML 2.0 section. Out and in ports are used to communicate with other actors. A unique name and type is included in the address. An actor has a state machine and could possibly have an inner structure. Actors are components that are loosely coupled. The actors in ActorFrame are equal peers and possibly distributed across network domains. This means that the components themselves negotiate and agrees upon a common protocol. The components should be able to set up a connection between themselves them without require a central server. The components should be able to be deployd anywere and without connecting to server. Actors can be dynamically deployed and removed during the lifetime of the application server. An actor may encapsulate many actor-processes that execute in parallel. The most essential messages used in ActorFrame Protocol between actors are shown in Figure 4.5-2.

RoleRequest	Request for a role to be played by an actor
RoleRelease	Release of an association between two actors
RoleRemove	Delete of non persistent actors including inner actors
RoleUpdate	Update of the configuration of an actor including inner actors
ReportRequest	Request for an actor report including inner actors (parts)

Figure 4.5-2 The most essential Role messages [109]

A RoleRequest message is sent to an actor to ask if it can play a role. RoleRequest is a variant of Actor message. If it cannot play the requested role, it will answer with the roles it is capable of. ActorFrame has a generic behaviour that keeps track of what associtations the Actor take part in and then track it. The behaviour also releases all the Roles in a play when the play ends. Role commands are used as dynamic communication between actors.

The ActorFrame environment is described in the figure below. This figure shows the class hierarchy of ActorFrame. It shows the relations between the components in the UML notation. ActorSM is the base class of the actor and contains the state data. It can contain composite states, parts and ports. ActorCS implements the state machine and behaviour of the actor. PartSpec contains the specification of the inner actors or parts. Port implements a simple port state machine. More detail on ActorFrame is explained in ActorFrame Developers Guide [6].

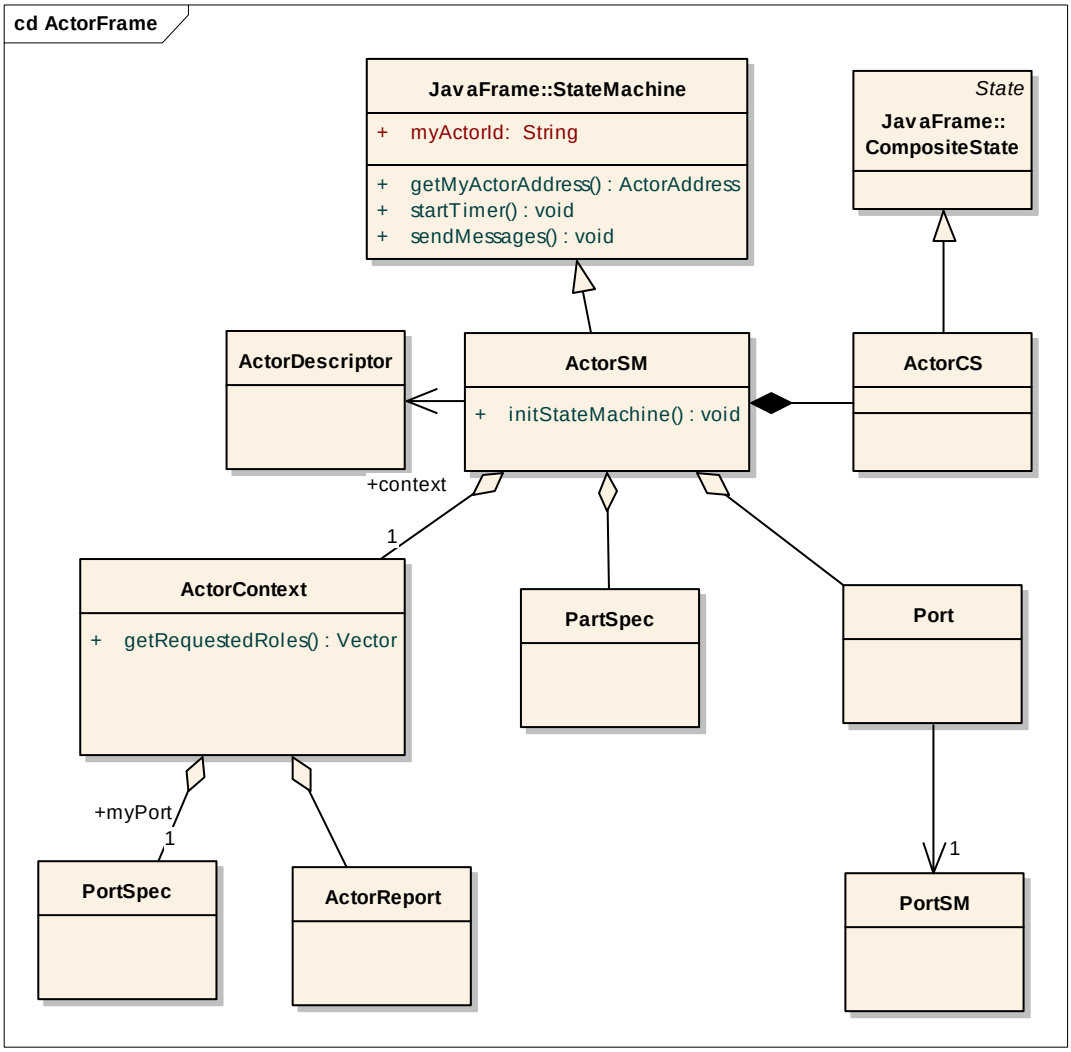

Figure 4.5-3 Class hierarchy of ActorFrame [110]

ActorDomain

Figure 4.5-4 shows an example of an ActorDomain. ActorDomain is a special kind of Actor that is running on the ServiceFrame application server. It is a sort of root "root" actor for the ActorFrame domain where all actor components are inner parts of this ActorDomain actor. Each actor domain will have an ActorDomain actor running. In Figure 4.5-4 below, the ActorDomain is called "arts", i.e. "arts" is an instance of the component "ActorDomain". This actor domain has three inner parts called "a", "b and "c". The act or "c" has also an inner part called "d". As shown in Figure 4.5-4 "b" is actually a component of type Agent, which is an Actor with additional functionality. The figure also shows that the actor "a" has two ports "r" and "p". The port "r" is connected to agent "b" while the port "p" is connected to port "q" of actor "c".

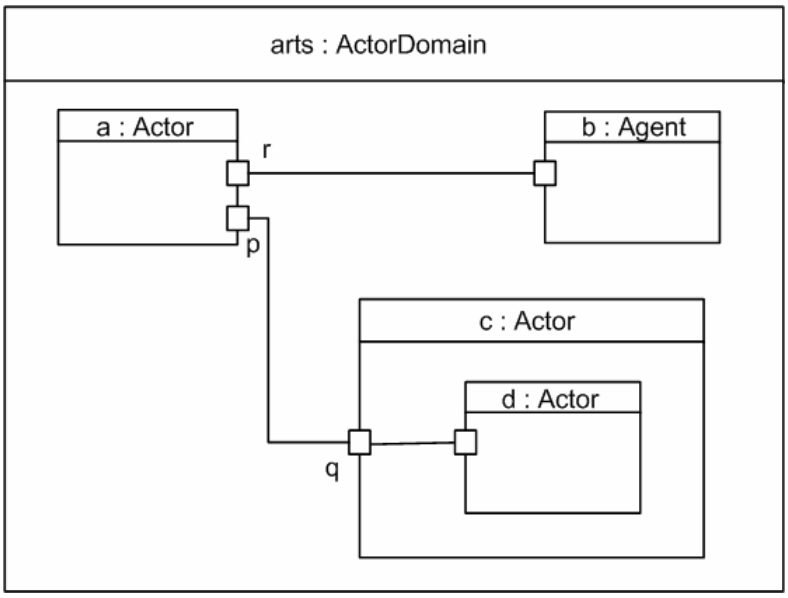

Figure 4.5-4 ActorDomain

ActorAddress

Each of the actors or agents in ServiceFrame has a unique address called Actor Address. The address consists of the instance name of the actor and the instance name of the surrounding actors which this actor is a part of, separated by a slash "/". This is much the same as the path names in a hierarchical directory structure. For example the actor "d" has the actor address "/c/d", whereas the actor "c" has the address "/c". If the actor "d" had an inner part called "e" this could be addressed as "/c/d/e". This is not shown in Figure 4.5-4. Sometimes the actors are also addressed by their class type in addition to their actor address, for example "c" could be addressed "/c@Actor" and "b" could be addressed as "/b@Agent". The actor domain name is not a part of the actor address because the name is supposed to be unique in the whole name space, across all actor domains. The reason that the names should be unique is that a component should easily be transferred from one actor domain to another and still be accessible without modifying the actor address.

ActorMsg

Actors and agents in ServiceFrame communicate asynchronously through message passing. The messages sent are called actor messages and are of the type "ActorMsg" or specialised class inheriting "ActorMsg". The actor messages contain fields for the actor address of the sender and the actor address of the receiver as well as the message contents. Messages in ServiceFrame can be sent to a local port if this is connected to a port of another actor. In Figure 4.5-4 above, the actor "a" can send a message to its local port "p". The message is then automatically sent to port "q" of actor "c". The connectors and ports can therefore be used when sending messages instead of using actor addresses. Although sending to the actor address is always possible.

ActorRouter

ActorRouter is a special component that connects actor domains. The ActorRouter is responsible for forwarding messages over a network connection, using TCP or UDP packets. It routes messages in a similar way to traditional network routers, except that it sends messages between actor domains. An ActorMsg message is encapsulated in a RouterMsg

before it is sent over the Internet. The actor router in the other actor domain then unpacks the ActorMsg message from the RouterMsg before it forwards the message to the right recipient actor. The ActorRouter is marshalling and un-marshalling actor messages. This means that it transfers the actor message into a byte stream, called “serialising the message”, that can be sent over the Internet and then back again to an actor message, called “de-serialising the message”.

4.6 JavaFrame

“JavaFrame is a Modelling Development Kit for development and execution of state machines in Java. It provides a layer between the Java language and concepts used in state machine modelling.”[111]. The JavaFrame has a Composite State and a State Machine. The Composite State in JavaFrame is a sub state machine in UML concept. It is possible to use the submachine concept of UML in JavaFrame. In JavaFrame this is called Composite State. State Machine may consist of one or many Composite States. UML allows entering and leaving of composite states without going through entry and exit points. This is not possible in JavaFrame, because the entry and exit points are required. JavaFrame is the class library which defines concurrent state machines in Java. The communication between state machines is done asynchronously. The figure below shows how an asynchronous communication is performed. As explained in the Figure 4.6-1the communication between A and B are independent of the work done at A and B.

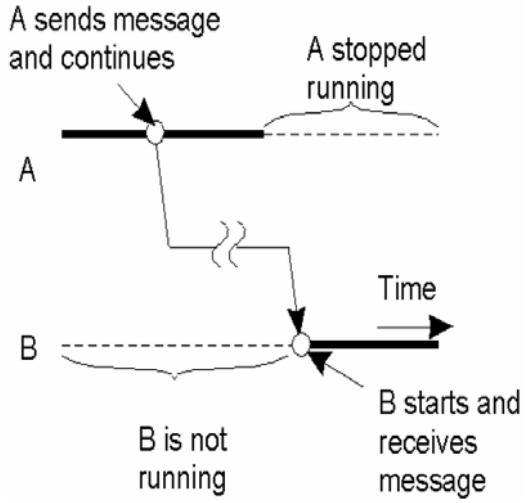

Figure 4.6-1 Asynchronous message [58]

4.7 Summary

This chapter has concentrated on presenting ActorFrame, ServiceFrame and JavaFrame and their relation to each other. The role of UML 2.0 in presenting the construction of these three layers has been well documented. The language used to develop these three layers has been Java. In the next chapter we will give a brief security assessment of possible ActorFrame and ServiceFrame threats. In the next chapter we will also propose some security mechanisms for authentication, authorisation, integrity and confidentiality.

5 Proposed security mechanisms

5.1 Introduction

In this chapter we will give a threat assessment of ActorFrame and ServiceFrame. We will then propose some security mechanisms to handle the ActorFrame threats. We will first explain the security protocol with which we suggest two of the three research questions of our master thesis can be solved. This protocol proposes to offer mutual authentication before establishing a secure communication using both confidentiality and integrity. We will then present the security mechanisms which we propose using to solve the third research question concerning authorisation of agents. Some of these proposed security mechanisms are implemented in a prototype.

5.2 Threat Assessment for ActorFrame and ServiceFrame

There is practically no security provided in ActorFrame or ServiceFrame at present. The overall design of ActorFrame has been to make it a highly distributed and a loosely coupled system. Actors in ActorFrame are loosely coupled as they do not require a central server. Interactions and communication between actors are handled by the actors themselves. ActorFrame has been designed to be flexible regarding the accessing of actors. Actors can be accessed through the use of connectors when linking two ports, or through specifically sending messages to their actor address.

The way the actor structure is defined makes it possible to access all actors, even if they are defined as inner actor parts. All actors can be accessed through their unique actor's address, registered in the name server. When a message is sent from an inner actor to another external actor, all the addresses of the actors forwarding the message are recorded in a reply stack. This means that anyone that is able to listen to the message flow, may see from where the message originated by examining the reply stack. Should the inner actor be hidden and not accessible directly from the outside world, the address must not be known to the public. Some mechanisms are already defined which may reduce the exposure of actor addresses to attackers. It is possible to hide the inner actors' addresses by not placing them in the reply stack. This is done by specifying that the port of the surrounding actor has the "hideInner" property set to true. The problem is that messages do not need to pass through the port. Removing the actor addresses of the inner part may only provide a false sense of security. It is still possible to access the inner actors if one should somehow acquire their actor address.

The actors' addresses are meant to be unique in the whole ServiceFrame system, even across domains. The domain name is not part of the actor's address because the actors could be relocated from one domain to another and still be accessible using the same address. This may result in a possible spoofing attack. Two actors with the same actor's address can be created on different actor domains. When a message is sent to an actor's address the name server is first called to see if the actor exists in the same domain. If the actor is not found, the message is sent to the ActorRouter which then forwards it to another actor domain. The result is that the actor's address cannot alone be used to identify the actor. The actor's address has the same weaknesses as the IP address in traditional networks. The authentication cannot be performed based solely on these addresses. In order to authenticate actors since the actor's address can be spoofed, the actor has to somehow prove its identity. Currently it is not possible to authenticate the actors or agents in ActorFrame.

ServiceFrame provides actors known as “edges” for accessing third party APIs such as Parlay/OSA and Parlay X. These edges take care of the connection to the Parlay gateway using a lower layer of protocols such as CORBA and Web Services. The Web Services are considered lower layer protocols in our context since ActorFrame communication is at the highest level. The same weaknesses found in both Parlay and CORBA are still relevant security issues for the NRG edges. ServiceFrame is currently an entirely open system and access is based upon trust. There are no restrictions on what the actors or agents can do. This will not be sufficient should the system offer commercial services for the telecommunication networks. There must be some kind of access control which regulates the access to resources such as the NRG edges. There is no control of the access to the Parlay gateway, since the NRG edges are not secured. The result is that any services can access the Parlay gateway and can use resources in the telecommunication networks. This is of course a serious security concern.

If two remote ServiceFrame applications servers are running an ActorDomain, no security mechanisms for controlling access to the ActorRouter is running and therefore there is no control of the network traffic that slips through. The two actor domains could communicate securely through a VPN tunnel but this would rely on security at a lower level. An attacker could possibly create his own ServiceFrame running actor domain. He could establish a connection to the other actor domains if there is no VPN security. He could then launch an attack where he gains access to all actors and agents in the other ActorDomain, since the actors and agents are not secured. Currently there are no authorisation or access control mechanisms to secure the access to the actors or agents in ServiceFrame. This poses a security threat.

ActorFrame and ServiceFrame rely on asynchronous messages passing from actor to actor. If the actors reside on different domains the message is sent to the actor router. The actor router only encapsulates the message in a RouterMsg message and then forwards it to the next actor router, until the message reaches its destination. This means that messages are sent in plaintext and there is no integrity check of the message contents or integrity of the originator. Since there is no integrity, it is possible to tamper with the messages as they pass by. ActorFrame is therefore subject to ”man-in-the-middle” and replay attacks as well as attacks on integrity and confidentiality. An actor or agent receiving a message cannot verify the origin of the sender or verify that the contents have not been altered.

5.3 The key exchange protocol

At an early stage in our thesis work we were forced to make a decision about whether or not Needham Schroeder Lowe should be used in our prototype. We considered both the Diffie Hellman with signatures and the Needham Schroeder Lowe Public Key exchange protocols. Diffie Hellman and Needham Schroeder Lowe are, in our opinions, both suited for ActorFrame. We decided on the latter because Needham Schroeder Lowe is a protocol generally considered as secure as mentioned in Chapter 3. We are not aware of any flaws in the protocol which affect the way we use the protocol.

We propose using a variant and an improved version of the Needham-Schroeder-Lowe public key protocol mentioned in Chapter 3. We have modified the protocol to include certificates and adapted it for use in ActorFrame. Our protocol does not send an Id string of the sender as part of the message because the actor address of the sender is used as an id. We have earlier mentioned that actor addresses can be spoofed, but in our protocol the id does not need to be

secret. The certificates also provide id information of the sender although it cannot alone prove any identity.

The three research questions in our thesis definition were: *"How to authenticate Agents in ServiceFrame?", "How to authorise Agents in ServiceFrame?"* and *"How to achieve integrity and confidentiality on messages in ServiceFrame?"*

Our protocol proposes to solve two of these research questions. The protocol offers mutual authentication thereby solving the first research question. The protocol also offers confidentiality and integrity on messages and solves the third research question. The protocol is shown in the Figure 5.3-1 below and an accompanying description follows.

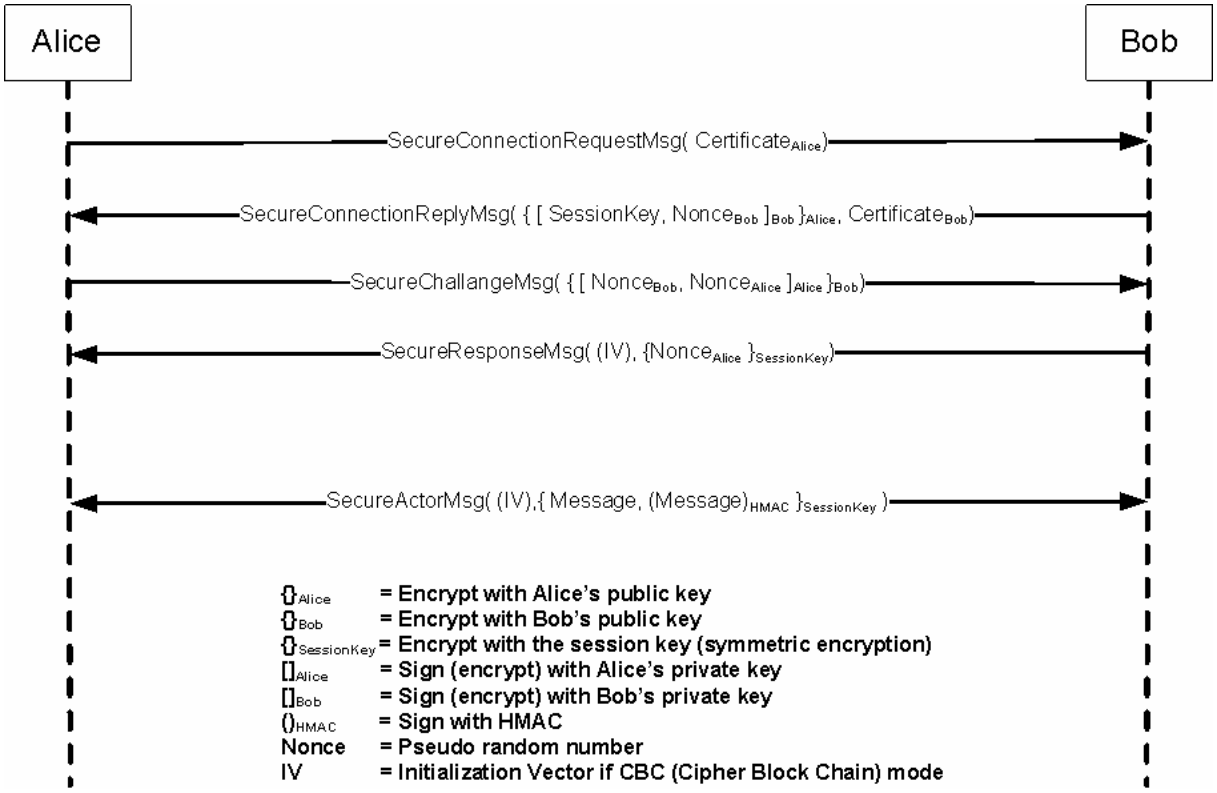

Figure 5.3-1 The proposed security protocol with actor messages

1. Alice sends a SecureConnectionRequestMsg. The message contains the field Certificate$_{Alice}$. The Certificate$_{Alice}$ field contains Alice's certificate. The certificate is used to identify Alice and is not encrypted since the certificate is publicly known.
2. Bob then sends a SecureConnectionReplyMsg. SecureConnectionReplyMsg contains the following fields: SessionKey, Nonce$_{Bob}$ and Cerificate$_{Bob}$. Normally it would have been wise to use a field with the id of the sender, but in our case we do not need to think of the id. The reason for this is that ActorFrame has an id informing the recipient which party is the sender. This id is always publicly known. The SessionKey is a key used for symmetric communication and is consequently used in the protocol after the asymmetric communication is done. In this protocol the asymmetric communication is done by the use of RSA. The symmetric communication algorithm is AES. Nonce$_{Bob}$ is a field with a random number only used once in the lifetime for the connection. The purpose in this field is to avoid replay attacks and a "man-in-the-middle". The SessionKey and Nonce$_{Bob}$ is signed with Bob's private key and encrypted with Alice's

public key. The signing is done to prevent tampering with the signed content. Bob is the only person who has his own private key and everyone has the public key. So Alice will know if the content form Bob has been tampered with. The encryption is made in order to protect the content form from following the wrong path.

3. Alice sends a SecureChallengeMsg contains $Nonce_{Bob}$ and $Nonce_{Alice}$. The $Nonce_{Bob}$ and $Nonce_{Alice}$ are sent for the same reasons as in SecureConnectionReplyMsg and are at this stage a part of a challenge response sequence. $Nonce_{Bob}$ and $Nonce_{Alice}$ are also signed with Alice's private key and encrypted with Bob's public key to withstand tampering for the same reasons as earlier mentioned.
4. The SecureResponseMsg contains IV and $Nonce_{Alice}$. IV is an initialization vector which is used as a start for the information to the CBC mode. The $Nonce_{Alice}$ is encrypted.
5. All following messages are SecureActorMsg. Integrity and confidentiality of the SecureActorMsg is provided by encrypting the message and the HMAC of the message with the session key. The IV is sent in plain text along with message.

The protocol solves problems one and three in thesis definition which are authentication, integrity and confidentiality. The authentication in the protocol is solved by using the certificates. Cryptography and signing solves the integrity. The symmetric cryptography, asymmetric cryptography and the signing uses the algorithms mentioned earlier. In the protocol the first steps use an asymmetric encryption method to exchange the session key. After the asymmetric encryption is executed, the protocol uses a symmetric encryption method. The reason that we change from asymmetric to symmetric is that symmetric encryption is faster, more efficient and less processor-demanding for large amounts of data, than the asymmetric is. The reason for the RSA being preferred for signing and for asymmetric encryption lies in the fact that it is the most used and has not been broken. RSA is today the most widely used algorithm and is used in the protocol Secure Socket Layer (SSL). In the protocol we chose to use AES as the preferred symmetric alternative. The algorithm used in AES was chosen following a competition in 1999 held by National Institute of Standards and Technology (NIST). The preliminary work consisted of a thorough analysis of the participants in a competition. We consequently chose AES. The main objective for the AES competition was to elect a Federal Information Processing Standard (FIPS) for algorithms in the USA. The result of this competition is described in the report [101].

Possible attacks on our protocol

The "man-in-the-middle attack" can be executed at every stage of the protocol. If the attacker tries to act as Alice or Bob, it should not work, since the attacker does not know the private key to Alice or Bob. The attacker is not aware of the symmetric key. Figure 5.3-2 shows that Eve can be a "man-in-the-middle" attacker without affecting Alice or Bob.

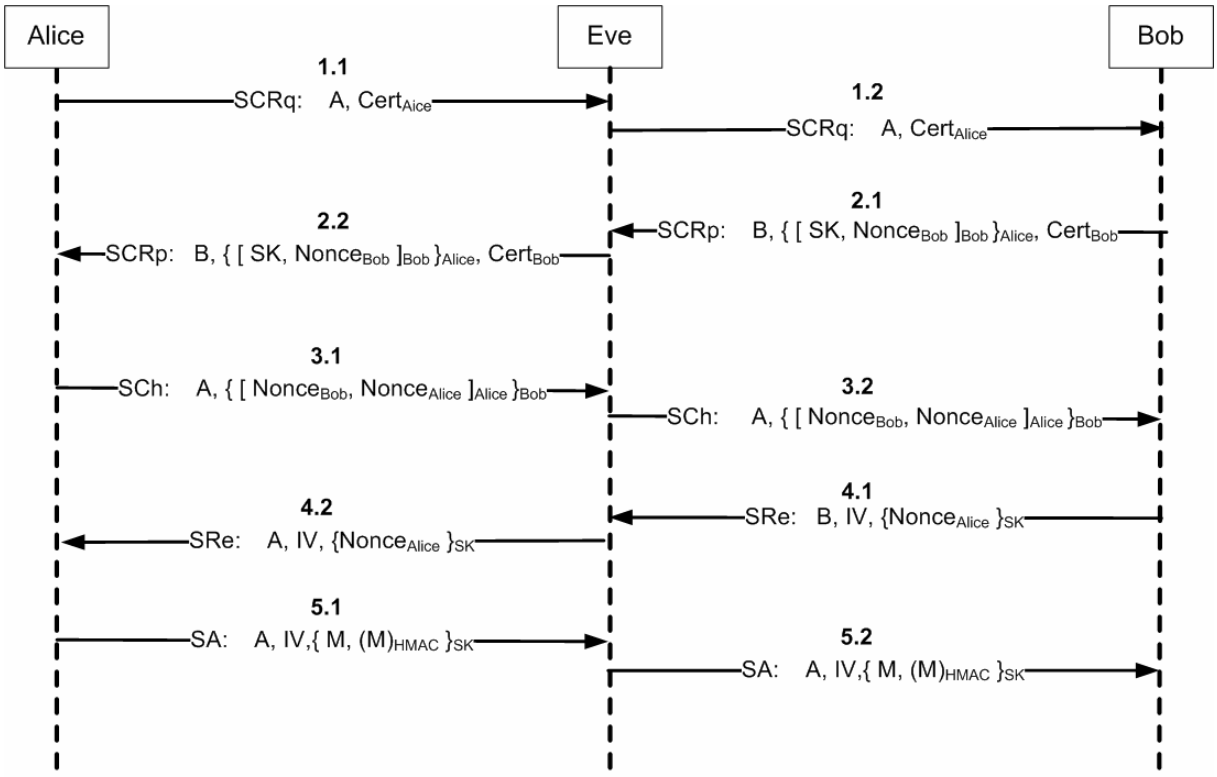

Figure 5.3-2 Example of a "man-in-the-middle" for the protocol

In this section we will conduct a brief analysis of the security of the protocol. We will not use formal methods and cryptanalysis of the protocol as this is beyond the scope of this thesis. We only try to give a short argumentation of why we believe this protocol is secure.

- In step 1.1 Alice sends her actor address and her certificate. Eve receives this and since this information is sent in plaintext she can replace both the address and the certificate.
- Let us assume in this scenario that Eve does not alter any messages in step 1.1 and just forwards the message as it is to Bob in step 1.2.
- Bob responds according to the protocol with a pseudo-random $Nonce_{Bob}$ and generates a pseudo random session key. Bob first signs the message and then encrypts the message with Alice's public key. Bob also sends his certificate and his actor address as shown in step 2.1.
- Eve receives the message but since the message is encrypted with Alice's public key she cannot recover the session key or $Nonce_{Bob}$ from the message. Eve can however send her own certificate, but Alice should notice this since she intentionally wanted to send a message to Bob. Let us assume that Eve only forwards the original message to Alice in step 2.2
- Alice decrypts the message using her own private key and then decrypts once more with Bob's public key. Alice then gets a session key and a $Nonce_{Bob}$ but she cannot yet be sure that this was actually sent from Bob. Alice encrypts the $Nonce_{Bob}$ and a new pseudo-random $Nonce_{Alice}$ and signs the message with her private key. In step 3.1 Alice then sends the message to Bob via Eve, whom Alice does not know is trying to act as a "man-in-the-middle".

- Eve receives the message from step 3.1. Eve does not know Bob's private key and cannot decrypt the message to recover the $Nonce_{Alice}$. Eve would just have to forward the message to Bob in step 3.2.
- In step 3.2 Bob decrypts the message using his own private key and then decrypts it using Alice's public key. Bob originally signed the $Nonce_{Bob}$ before using Alice's public key. It should not be possible for anyone else to retrieve the $Nonce_{Bob}$ except Alice. Since Alice sent the same $Nonce_{Bob}$ back to Bob, he has therefore in this step authenticated Alice. Eve cannot have impersonated Alice. Bob takes the $Nonce_{Alice}$ he received in the message that only he could decrypt and sends this encrypted with the session key in step 4.1.
- Eve cannot retrieve the $Nonce_{Alice}$ because she does not know the session key and therefore only has to forward this message unchanged to Alice in step 4.2
- Alice decrypts the $Nonce_{Alice}$ in step 4.2 and verifies that this is the one she sent to Bob. Eve cannot have forged this message because she does not know the session key. Alice therefore has authenticated Bob.
- In step 5.1 Alice sends an encrypted message with an encrypted HMAC of the message. When Eve receives this she does not know the session key and she cannot alter the message because then the HMAC integrity will fail.

5.4 Proposed Security Mechanisms

This section deals with possible security solutions for the ServiceFrame framework. We will propose solutions to the three problems stated in the thesis definition. Our proposed prototype will also deal with the security threats described in earlier in this chapter.

Limitations

ServiceFrame and ActorFrame framework are developed using the Java programming language. Java has already some security and cryptographic APIs included. The security mechanisms we make use of are of part the Java Cryptographic Extension (JCE). We have developed our prototype using Java 2 Standard Edition (J2SE), but the prototype should also be executable on Java 2 Enterprise Edition (J2EE) and Java 2 Mobile Edition (J2ME). At least it should work without many changes, although the J2ME platform is still very limited. We have not based our solution on making it work with the current limitation of J2ME, but what we soon hope to be capable of support, such as the cryptographic APIs in Java.

We have found that it is more efficient to place the security mechanisms in the Agent class instead of the Actor class. The Agent class inherits behaviour and functionality of the Actor class, but in addition it may have more internal state machines called Roles. Actors can also have roles but these roles are actually just other actors running as inner parts.

We use the X.509 version 1 since Keytool, which we use to generate certificates, only creates certificates of version 1. It can however import certificates of version 3. We use OpenSSL to sign the certificates. OpenSSL can generate certificates of version 3, but Keytool cannot import private keys generated by other tools. We therefore use OpenSSL only for creating the root certificate and use this root certificate for signing other certificates created by Keytool.

There are some limitations on how the security mechanisms should be implemented. Actors have contents or state data in a class called ActorContext. The contents should not include much data because of the persistency provided in J2EE. J2EE uses store-and-forward

operation to provide persistency of data. Persistency means that all data are stored on a temporary disk so that messages can be delivered even if the receiver is temporarily off line, in addition no data is lost if the server goes down. The mobile telephone platform J2ME does not support the object "serialize" functionality of Java. Due to this limitation every class has to have its own "serialize" method and do its own serialisation. Serialisation is similar to marshalling in distributed systems. It transforms the objects into a byte stream that can be transformed over the network and then transforms it back on the receiver side. This functionality is also needed should the objects be stored on disk. In J2SE there is support for "serialize" which leads to an easier implementation. One example is the SignedObject class which handles the whole process of signing a message. SignedObject only works if Java serialisation is supported. J2ME today does not fully support security, which makes our work harder. Speed is important and processor demanding operations should be chosen carefully. One of AES's strengths is that it is well suited for embedded systems.

Authorisation

The second research question, "*How to authorise Agents in ServiceFrame?*" cannot alone be achieved through the use of the protocol. Authorisation and access control are closely related because authorisation is a mechanism that provides access control. Our proposed access control mechanisms should not be dependent on central servers or mechanisms. As a result Mandatory Access Control (MAC) mechanisms cannot be used because there is no system to specify the access rights and permissions. MAC is primarily used in classified and military systems with classification levels between subjects and objects. In Discretionary Access Control (DAC) models the access control mechanisms are user or data centric. It is possible for users or resources to have specified access rights or permissions. DAC is more suited for ActorFrame because ActorFrame is a framework with distributed components and does not have a central server.

It is possible to incorporate access control through the use of Access Control Lists (ACL) or Capability lists (C-lists) in ActorFrame. The authorisations of the actors or agents could be achieved if the Agents themselves had specified an ACL with the permissions given to other agents. Such an ACM for the SmsNrgEdge could look like this:

SmsNrgEdge:

ActorDomain A	ActorDomain B	SmsService	Alice	Bob
Send	Denied	Send, Receive	-	Receive

It is possible to specify that ActorDomain B should be denied although it is common to let this be the default right. Denied usually only makes sense if it is possible to inherit rights through the use of a hierarchy of permissions given to both users and groups. Then Bob could be granted the access because he has this right, but because he might belong to the group ActorDomain B, he should actually be denied access. Denied usually overrides allow access. Through the use of the security notion of always giving access based on the least privilege, Alice will have the default permission. This is to deny Alice access, if she is not part of any other group which allows permission. The ACL and the access rights for the agent can be specified in the xml actor descriptor.

For ActorFrame we will argue that ACLs are easier to maintain than C-list because there is no operating system or built-in access control features in ActorFrame. An agent could have sent a C-list signed from an authority (root certificate perhaps) to prove that it has these

permissions. It is difficult to create an access control mechanism for authorisation when their agents and components are loosely coupled. A central server could have made it easier. ACL and C-list operates on the lowest level of access control. There would have to be some mechanisms on a higher level, such as RBAC. This is out of the scope of this master thesis' problem definition.

5.5 Alternative solutions

One of the possible solutions we considered at an early stage was to place the security mechanisms in the ActorRouter. The ActorRouter is the exit and entry point of an ActorDomain. All messages that are sent and received from the internet are sent through the ActorRouter. It is possible to let the communication between two ActorDomain be encrypted. When an actor sends a local message to the ActorRouter, the ActorRouter would encrypt it. When the message is received at the ActorRouter of the other ActorDomain, it is decrypted.

It is also possible to place the security mechanisms in the connectors and the actor ports. When a connector between two actors' ports is defined in an XML file it is also possible to add XML tags for specifiying secure conncetions. The connection between two ports is established through a protocol and involves a few messages between the two ports before the connection is established. When a message is received at the local port could encrypt it. The recevier port of the other actor could decrypt it. It could therefore be possible to establish a secure connection between two actors through the ports.

Procedure call could have been an alternative to the asynchronous messages in the security protocol. Procedure call has the advantage that the whole security protocol and the secure connection could be established inside a transition between two states. This means the the the procedure call does not affect the state machine and user code because it is treated as a normal method call.

5.6 Summary

In this chapter we have proposed security mechanisms that solve the research problems in the thesis definition. We have modified the Needham Schroeder Lowe public key protocol so that it uses certificates and suits our needs. The protocol features mutual authentication as well as confidentiality and integrity. Implementation of the protocol in ActorFrame as a prototype will then solve both the first and the third research question. The second research question regarding authorisation and access control can be solved using elementary access control mechanisms such as an ACL. The access control is rudimentary but provides some authorisation. The next chapter will describe how we designed and implemented the security mechanisms in ActorFrame.

6 Implementation of security mechanisms in ActorFrame

6.1 Introduction

We have implemented the security mechanisms proposed in a prototype. We have used the ActorFrame template provided by ServiceFrame to generate new actors and agents. The security mechanisms are implemented as a security state machine or inner role of the Agent. Each SecureAgent has then two state machines, the default one and the EstablishSecurityCS role. The default state machine performs the normal operations whilst the EstablishSecurityCS role waits for incoming secure connection requests. When the role state machine receives a request, it acts as a server and establishes a secure connection using our protocol. The other Agent sending the secure connection request enters a composite state that completes the security protocol on the other side. Each secure agent must have a certificate signed by a root certificate from a trusted Certificate Authority (CA). When the protocol has been performed and the two agents have authenticated each other, they can communicate securely with encryption and integrity.

We stated in the thesis definition that: *"If time allows, the proposed security mechanisms will be implemented and tested in a prototype"*. We have not had time to implement all the security mechanism but we have implemented the core concepts such as Authentication, Integrity and Confidentiality. Authorisation is an extensive topic and we have only proposed some solutions for that topic.

6.2 Design

We have designed the prototype without changing the fundamental structure of ActorFrame, only implementing additional functionality. The reason was that we did not want to redesign the whole system. If we had changed some of the core concepts we might have rendered some of the already existing components useless.

We have chosen to let the security mechanisms function with Agents not Actors. The reason for this is that Agent has the ability to have state machines or roles tightly connected to it. Actor also has the ability to have roles, but these roles are inner Actors. The roles of an Agent can be inner Actors, but can also be state machines which are a part of the Agent. These roles have their own state machine, but have access to the state data of the Agent. This means that they may alter the Agent's data directly. This feature makes it possible to let a security role inform the Agent's own state machine that a secure connection is established.

We have created a new generic type of Agent called SecureAgent. Programmers, who want to create services that make use of our security mechanisms, can create a new agent that inherits from our SecureAgent. The SecureAgent is a specialisation of Agent which has the role EstablishSecurityCS. Figure 6.1-1 below displays some of the class structure of our prototype. The classes in grey are the ones we have created. SecureConnectionState is a composite state used in the implementation of the protocol and is explained later. The other classes are part of ActorFrame.

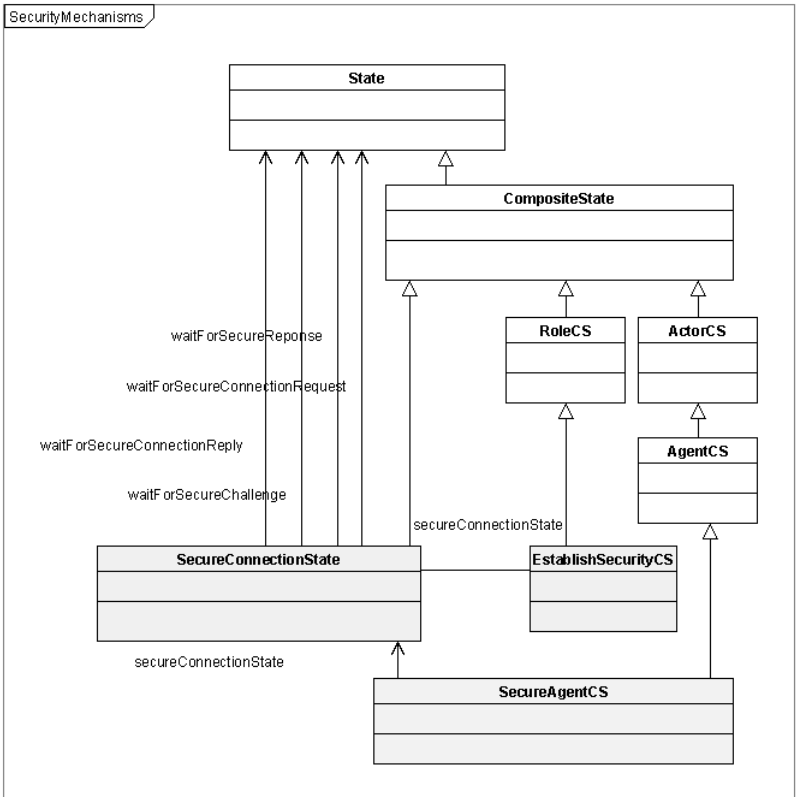

Figure 6.1-1 Shows an excerpt of the class structure with our SecurityAgent

We have implemented the security protocol we proposed in Chapter 5. The protocol had two parties, Alice and Bob. Alice initiated the secure connection with her SecureConnnectionRequestMsg message and thereby acting as a client. Bob acts as a server and waits for incoming request. 'Alice' and 'Bob' are in our implementation instances of SecureAgent. SecureAgent has, as stated earlier, two state machines, the default one and the role EstablishSecurityCS. The two instances Alice and Bob are two equal peers and can both act as a server and a client. The SecureAgent's default state machine is the client whereas the EstablishSecurityCS state machine is the server. Alice and Bob therefore have both a server state machine that can accept requests and a client state machine that can send a request. In the protocol in Chapter 5 we assume that Alice initiates the protocol. The message flow is shown in the sequence diagram Figure 6.1-2 .

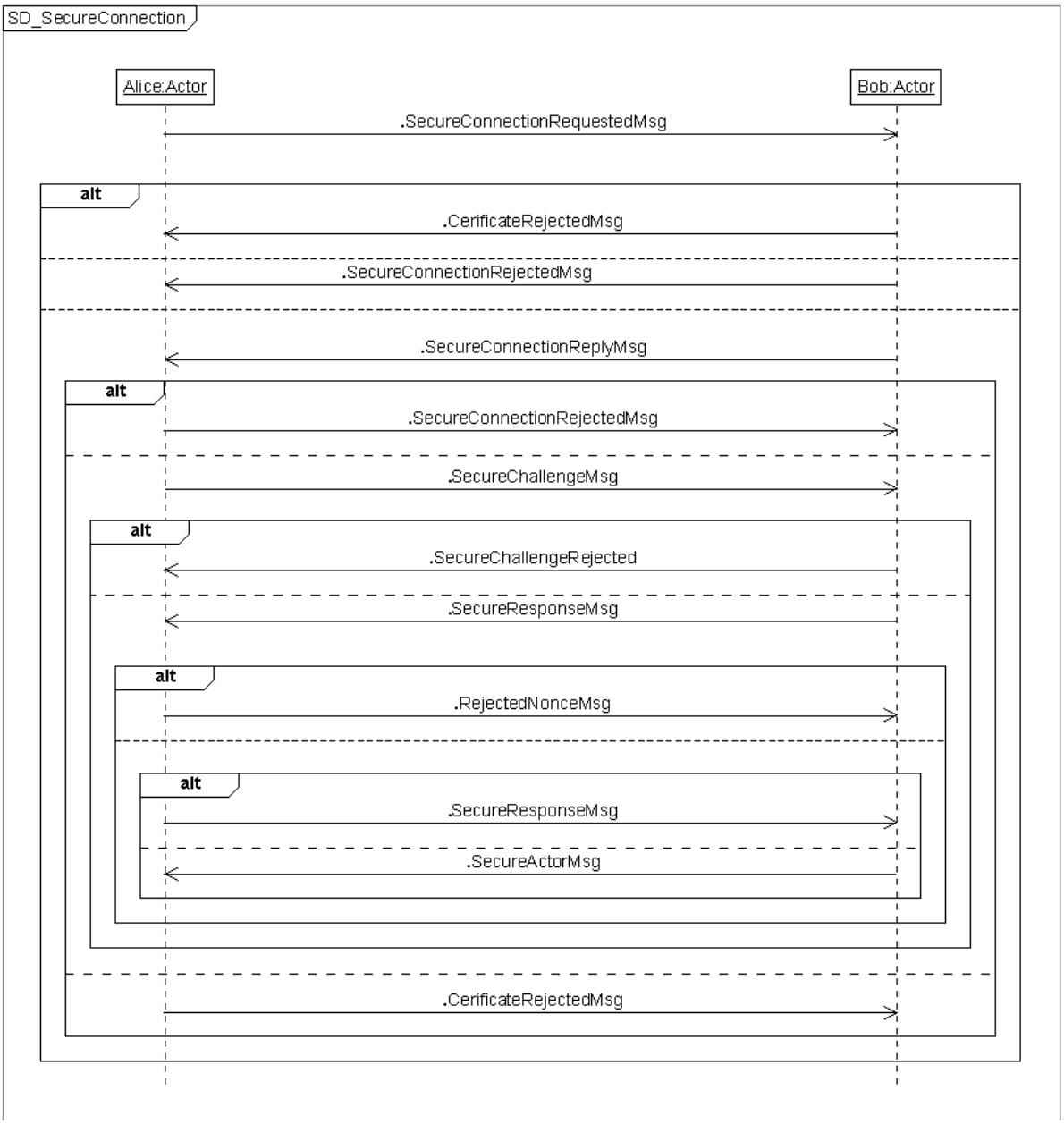

Figure 6.1-2 Sequence diagram for the implementation of the protocol

In the sequence diagram, Alice sends a SecureConnectionRequestMsg message with her certificate to Bob's EstablishSecurityCS state machine. The certificate is then checked to see if it is signed by a trusted root certificate. If it is not valid, then a rejection message is sent. If the certificate is valid Bob sends a SecureConnectionReplyMsg message with his certificate. This message also contains a session key and a "NonceBob" signed with his private key and encrypted with Alice's public key. Alice then decrypts the message and verifies Bob's certificate. If Bob's certificate is not valid, then Alice sends a reject message. Otherwise Alice signs and encrypts a SecureChallengeMsg message with Bob's "NonceBob" and a new "NonceAlice". This message is actually a response message to Bob, but also challenges him with her "NonceAlice". Bob receives this challenge message and verifies that she really is Alice because only she can reply with his "NonceBob". Bob then replies to Alice with her "NonceAlice" in a SecureResponseMsg message encrypted with the session key. Alice then decrypts this response message and checks if Bob replied with her "NonceAlice". Any subsequent messages are signed with an HMAC digest of the message and encrypted with the

session key. The sequence flow is precisely the same as the protocol described in Chapter 5, except that error conditions are handled. In our implementation we use RSA for encryption of the key exchange and AES for the encryption with the session key.

In order to make the components reusable we have chosen to implement the authentication and key exchange protocol as part of a composite state (submachine state) in ActorFrame. We have named this submachine state SecureConnectionState as shown in the class diagram in Figure 6.1-1 Shows an excerpt of the class structure with our SecurityAgent The whole protocol and key exchange can be performed inside this submachine state. We have defined two entry points for this composite state and two exit points, as shown by Figure 6.1-3 below. Both Alice in her default state machine and Bob in his EstablishSecurityCS state machine enter this composite state when the protocol is used. The two entry and the two exit points correspond to the two different parties in the protocol. The entry point ENTRY_WAIT_FOR_REQUEST is the entry point used by Bob's EstablishSecurityCS when it waits for an incoming secure connection request message. The entry point ENTRY_WAIT_FOR_REPLY is used by Alice's default state machine when she sends a secure connection request message and waits for a reply. Three error conditions exit through the default exit point EXIT_ERROR. As seen by the submachine state diagram below there are two simple states for each entry point corresponding to the protocol message flow.

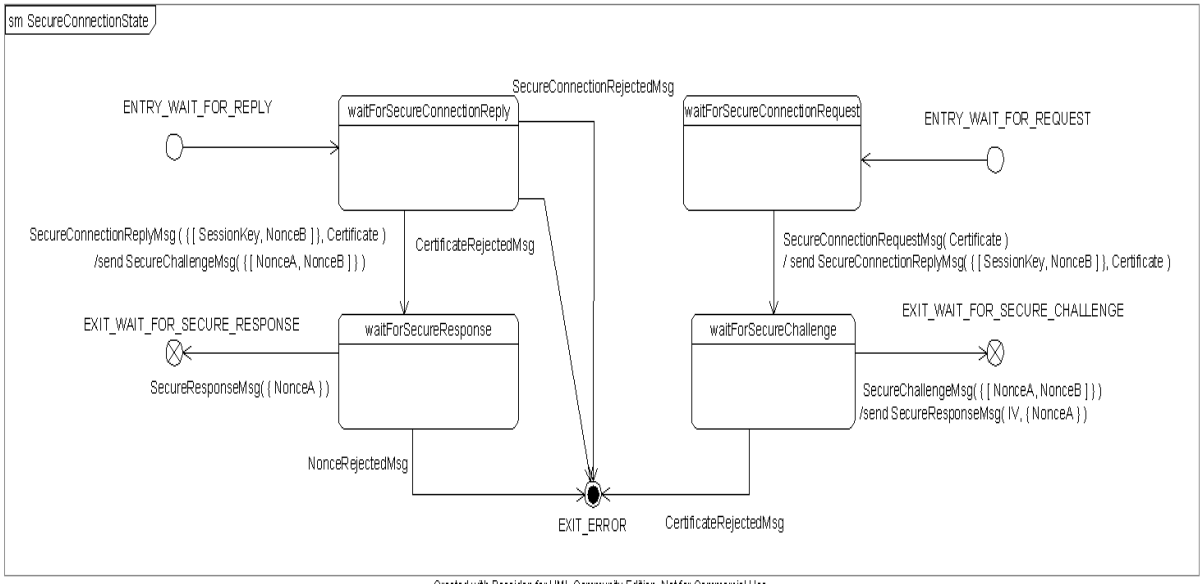

Figure 6.1-3 The composite state SecureConnectionState for the security protocol

The state diagram for the role EstablishSecurityCS is given in Figure 6.1-4 below. As seen in the diagram, the EstablishSecurityCS role receives a RolePlayMsg message when the role is created by the SecureAgent through a RoleRequestMsg message at start-up. When the preparation for the role is completed it receives a StartPlayingMsg message and enters the composite state through the entry point. Only one of the entry points and one of the exit points are used as this role acts as the server in the protocol. The default exit point is shown as the arrow in the middle. This is used for the error conditions.

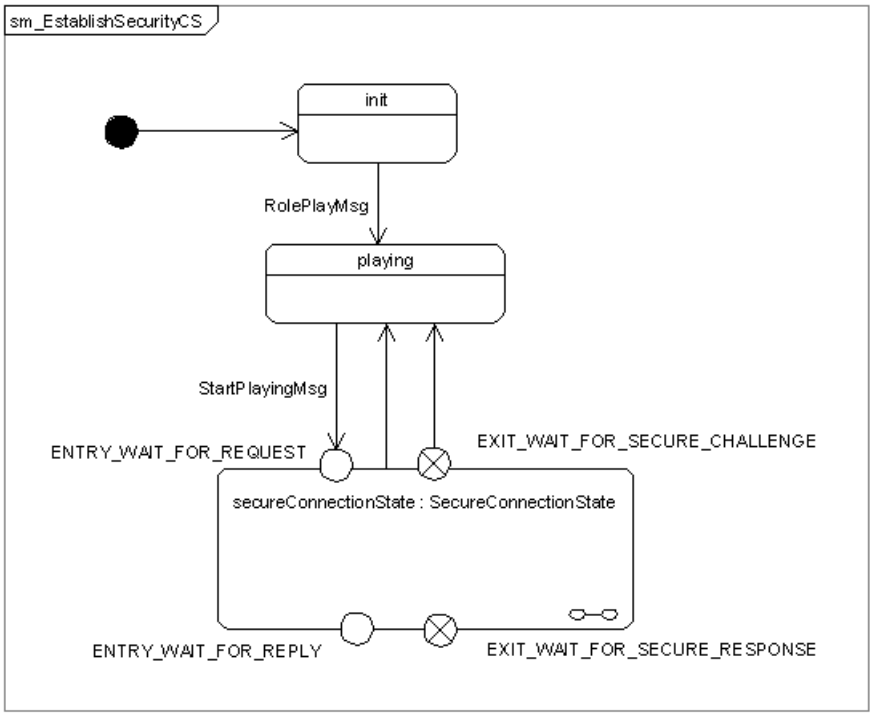

Figure 6.1-4 EstablishSecurityCS role state machine diagram

The state diagram for the SecureAgent is given Figure 6.1-5 below. SecureAgent sends a RoleRequestMsg for the EstablishSecurityCS role upon start-up. Further behaviour of the SecureAgent has to be defined in the new agent inheriting the generic SecureAgent.

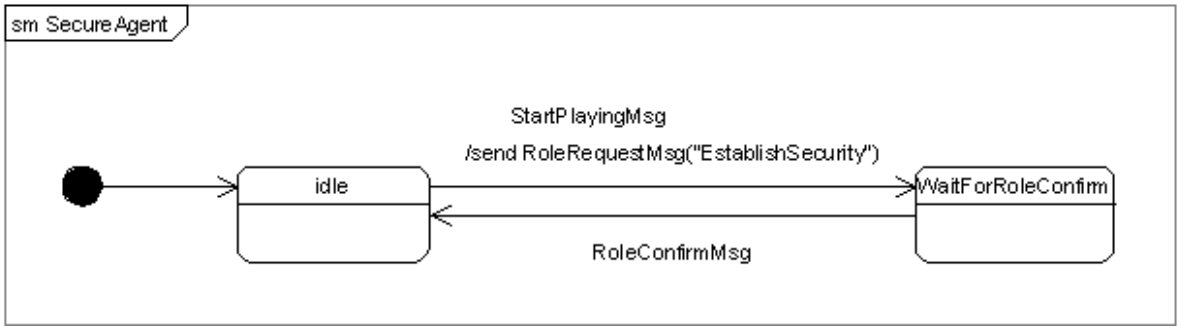

Figure 6.1-5 Shows the state machine diagram for the SecureAgent

The default state machine diagram for a new agent inheriting SecureAgent has similarities to the state machine for the EstablishSecurityCS role. The difference is that the default state machine is the one acting as a client. It enters the composite state through the ENTRY_WAIT_FOR_REPLY entry point after receiving a custom message. In this case a StartSecurityTestMsg message. It is up to the programmer of the agent inheriting from SecureAgent to implement the sending of a SecureConnectionRequestMsg message. The

programmer must also make sure that he enters the composite state SecureConnectionState through the correct entry point. Code handling the default exit point and the code for the EXIT_WAIT_FOR_SECURE_RESPONSE exit point must also be explicitly programmed.

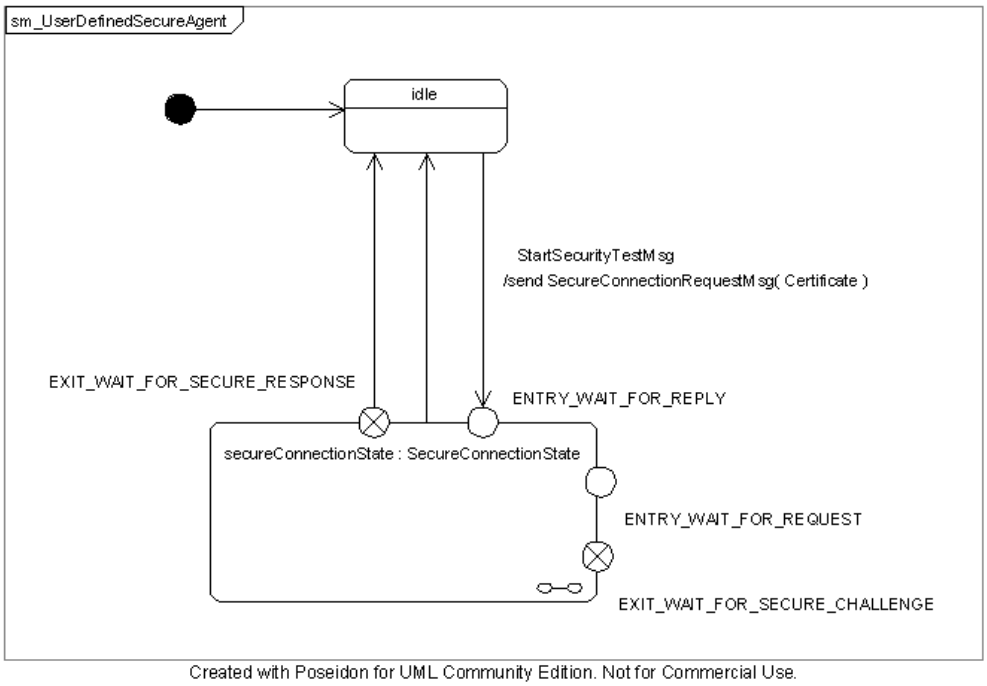

Figure 6.1-6 Example of state machine of an agent inheriting the SecureAgent

We have also defined some generic helper classes such as the AFKeyStoreHandler, AFCertificateHandler and the RSABlockWrapper. These classes are used in addition to the already existing Java classes for security. A class diagram of the helper classes is shown in Figure 6.1-7. The AFKeyStoreHandler takes care of the reading of certificates and keys from the Java keystore file. The keystore file for each secure agent is used when validating a client certificate and when loading the agent's own certificate and keys. The actual validating of the certificates is taken care of in the AFCertificateHandler. The RSABlockWrapper class has been defined to split the encryption into blocks before applying the RSA encryption or decryption. This is because the RSA cipher in Java only handles blocks of a size less than 117 bytes because of a few bytes used for padding. In our code we are encrypting with RSA twice, once for the signing and once for the encryption. The result of the first encryption gives a block of 128 bytes which necessitates splitting the result into two blocks for the next encryption.

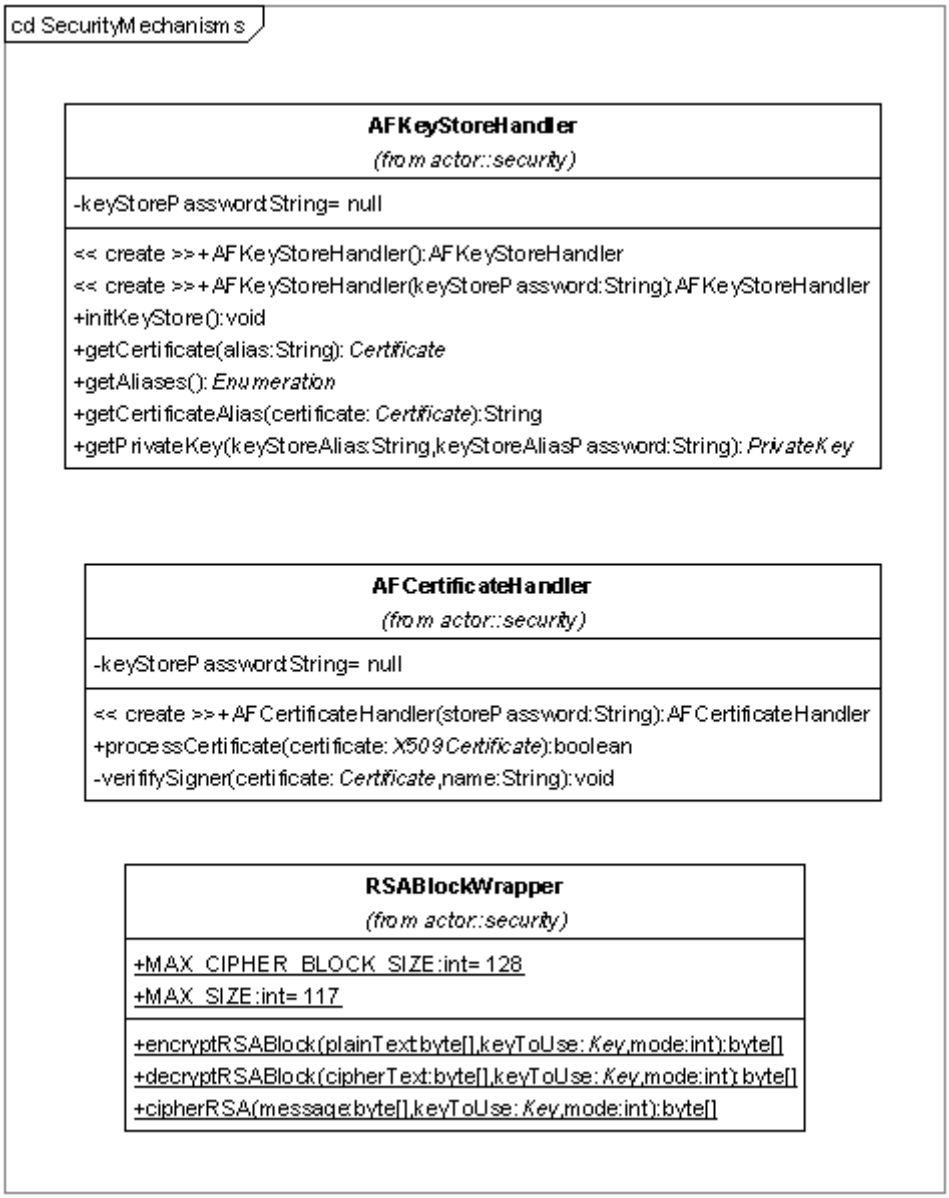

Figure 6.1-7 Shows some of the helper classes

We have also designed wrapper classes for the contents of the ActorMsg messages. The protocol requires the session key and the two 'nonces' to be encrypted. We therefore create a class SecureContent which contains these data. The data from the SecureContent are returned as a byte array and encrypted. The result is placed in a wrapper class called SecureContentWrapper. Additional information such as the certificate or initialisation vector if the encryption mode is CBC, are also placed in the wrapper class.

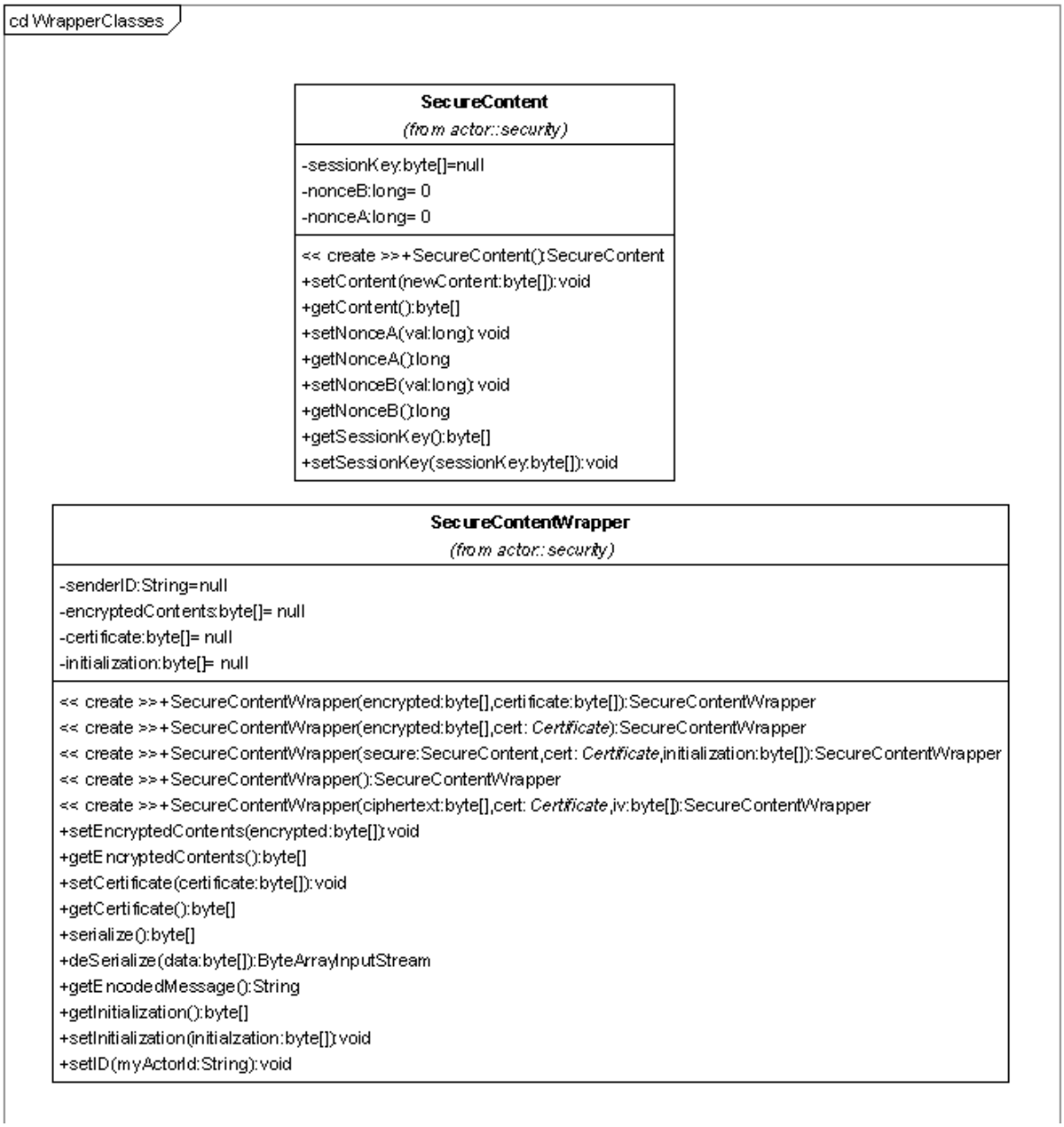

Figure 6.1-8 Wrapper classes used in the protocol when sending encrypted contents

We have also defined custom actor messages inheriting from ActorMsg for the various messages in the security protocol. One of the important ones is the SecureActorMsg message which is designed to wrap in or to contain another encrypted ActorMsg message. The recipient would just extract and decrypt the ActorMsg.

6.3 Implementation

We created the new SecureAgent using the ActorFrame template and modifying the Ant build script. The Ant script in ActorFrame is used for automatic code generation of new actors or for compiling source code. The security mechanisms and the classes such as the SecureActorMsg, SecureContent, and SecureContentWrapper are implemented.

Sample of the source code is shown below.

```
..

SecureContentWrapper newWrapper = new SecureContentWrapper();
SecureContent newContent = new SecureContent();
newContent.setNonceA(nonceA);

Cipher aesCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

aesCipher.init( Cipher.ENCRYPT_MODE, sessionKey);

byte [] cipherText = aesCipher.doFinal( newContent.getContent());
byte [] iv=aesCipher.getIV();

newWrapper.setEncryptedContents(cipherText);
newWrapper.setInitialization(iv);

SecureResponseMsg msg = new SecureResponseMsg(newWrapper);

asm.sendMessage( msg, sig.getSenderRole());

..
```

The code above is a part from the SecureConnectionState composite state. It is part of the code executed in the 'waitForSecureChallenge' state when it receives a SecureChallengeMsg. The code is actually what is being performed in the transition between the waitForSecureChallenge state and the next state. As seen in the code a SecureContentWrapper class and a SecureContent class are instantiated. A nonce is placed in the SecureContent and encrypted with session key using AES. The resulting ciphertext is placed in the SecureContentWrapper along with the initialisation vector. The wrapper is then placed in a SecureResponseMsg and returned to the sender. This is precisely what should be done according to the protocol.

6.4 Summary

The implementation of the security mechanisms for ServiceFrame has been implemented in the agent SecureAgent. This is a generic class that inherits from Agent and can be specialised by new agents that need security mechanisms. Upon initial execution of new agents based on SecureAgent, the role EstablishSecurityCS is created and run as a separate state machine. Other secure agents can send a secure connection request to the EstablishSecurityCS role. Both the secure agents will enter their own composite state, SecureConnectionState, the state that handles the protocol. Different entry points and exit points are used accordingly. If the establishment of the secure connection succeeds, the agents can communicate securely using encryption with a session key. Currently there are some limitations of the implementation of the security mechanisms. It is not yet possible to re-establish a session with a new session key if the session is too long. There are also some limitations due to the fact that the security resides on agent level and is not available for actors. This result is that existing actors cannot be secured and new services should be based on SecureAgent instead. As an overall view the prototype of security mechanisms offers security mechanisms such as Authentication, Integrity and Confidentiality for agents in ActorFrame. The next chapter will describe a test case and demonstrate the functionality of the security mechanisms through a test scenario.

7 Test Case

7.1 Introduction

In this chapter we describe three possible test case scenarios which use the security mechanisms implemented in the prototype. Because of the time limitations we have only implemented a test case for one of the test scenarios. The test case will show whether or not our proposed security mechanisms implemented in our prototype actually work. We have chosen a scenario where a user, represented as an agent, wants to send a SMS message securely. We make the assumption that the message only should be transferred securely over the internet between two agents. We do not address the security issue of the telecommunication network, but instead focus on securing the agents' communication.

7.2 Test Case scenarios

The Figure 7.2-1 shows three scenarios where the security mechanisms in ServiceFrame can be used. The ServiceFrame application server in domain #1 is connected to the application server in domain #2 through an insecure Internet connection. Eve is the person in red and she is a potential attacker located somewhere in the Internet. She may act as a "man-in-the-middle" attacker and she may fabricate or replay messages. The ServiceFrame application server in domain #2 is connected securely to the NRG Parlay gateway, possibly through a VPN connection. The NRG Parlay gateway provides services in the telecom network such as sending and receiving SMS, position services and call control. Charlie and Sam are two typical mobile phone users. Alice and Bob are two mobile phone users with MidletFrame, the J2ME version of ActorFrame running on a mobile phone.

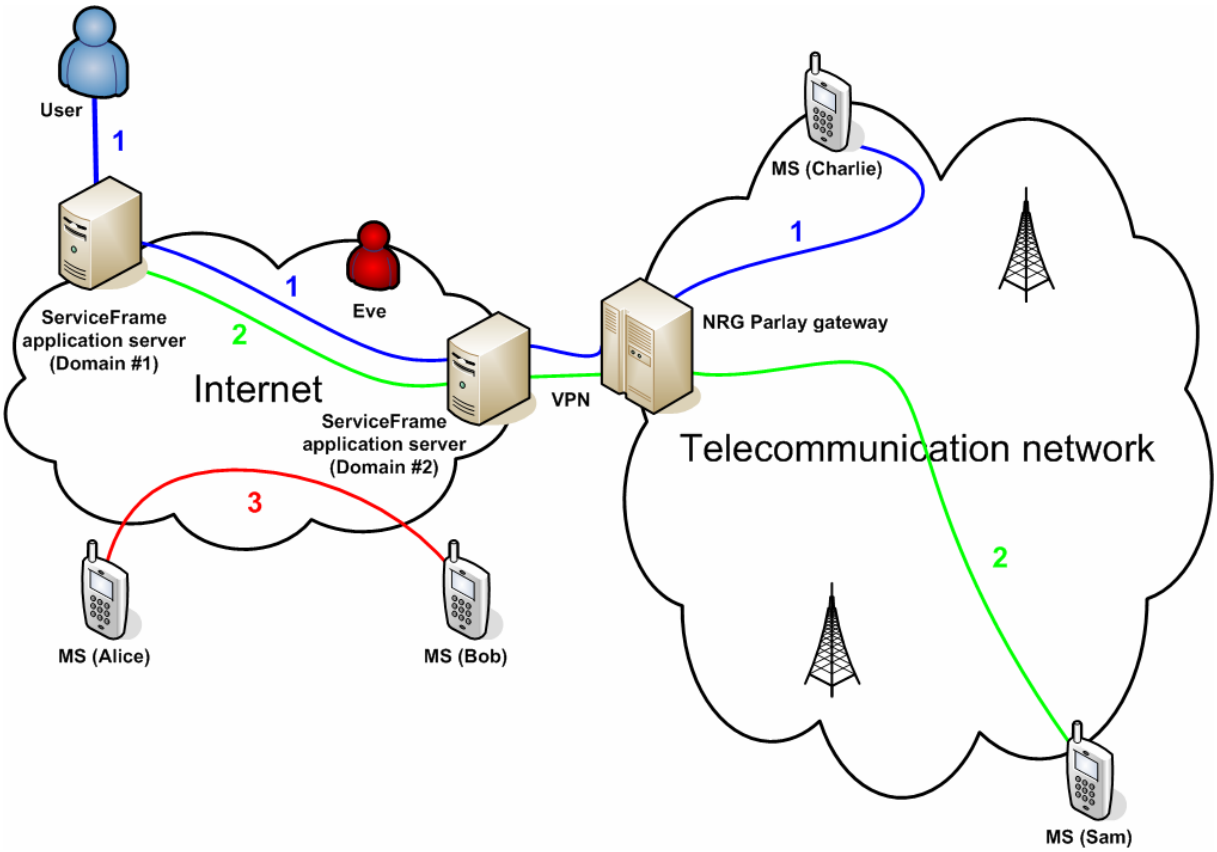

Figure 7.2-1 Possible test scenarios using the proposed security mechanisms

Test scenarios:

1. The user in blue wants to send an SMS message securely from the ServiceFrame application server in domain #1 to Charlie's mobile phone. The message must be sent securely through the Internet. The message is sent from an agent in domain #1 to a SecureSMSEdge agent in domain #2 after a secure connection between the two agents is established. The SecureSMSEdge then sends the message to the NRG Parlay gateway which delivers the message to the mobile phone.
2. A service component located in the ServiceFrame application server in domain #1 wants to securely receive positioning information from Sam's mobile phone. The positioning information must not be intercepted by an attacker located in the Internet. The service agent in domain #1 first establishes a secure connection to a SecurePositionEdge agent in domain #2. Then the SecurePositionEdge agent requests positioning information from the NRG Parlay gateway, for Sam's mobile phone. Position information is then sent securely from the SecurePositionEdge to service agent in domain #1.
3. Alice and Bob both have mobile phones running MidletFrame with the security mechanisms implemented. Alice and Bob want to send ActorMsg messages securely between each other. Both MidletFrame applications will have agent components that extend the SecureAgent. The agent running on Alice's phone sends a secure connection request message to Bob's agent. They both use the security protocol to establish a secure connection.

Due to time limitations we have chosen only to implement a test case with the first scenario, "User Sends SMS".

7.3 Test Case: User sends SMS

We have implemented a test scenario where an Agent represents a user who sends an SMS to a mobile phone. We have used the name TestCase for this agent.

The user interface could have used JAAS for authentication of the user and then asked for a username and a password. This would only be to demonstrate the authentication for practical purposes in a real scenario. We have chosen a simpler variant for our test case, as we only concentrate on securing authentication and communication between agents. We do not provide an end user security. Therefore in our example we assume that the Test Case agent has already authenticated the end user.

We want the Test Case to use its own certificate to authenticate itself to the SecureSmsNrgEdge agent. The SecureSmsNrgEdge agent is an agent functioning as a security proxy agent for the SmsNrgEdge. We assume that the communication between the SecureSmsNrgEdge and the SmsNrgEdge is secure, i.e. they are running in the same application server or Java Virtual Machine (JVM).

We assume that the TestCase agent and the SecureSmsNrgEdge are at different ActorDomains and therefore the communication between them passes through unsecured Internet. In this scenario the important part is to mutually authenticate the two agents and then

secure the messages using integrity and confidentiality. An overview of the test case is given in Figure 7.2-1.

The TestCase agent is situated in domain #1 while the SmsNrgEdge is in the domain #2. Both of the agents are inheriting properties from our SecureAgent, i.e. the one we implemented as part of the security mechanisms prototype. This means that the TestCase and the SecureSmsNrgEdge agent are of type SecureAgent and are able to set up a secured connection using the proposed protocol and security mechanisms.

Both of the agents will have their own certificate signed by the ServiceFrameCA root certificate. They will also have their own pair of public and private keys. Both agents also have the ServiceFrameCA root certificate as a predefined trusted CA, or else they would be unable to trust each other. The ServiceFrameCA could have been replaced by a trusted CA such as VeriSign and Thawte. We have not used these as we are only developing a test case. Actually, since we are in control of the signing of other certificates, through the ServiceFrameCA root certificate, we can control which agents are allowed to set up a secured connection. In this way we may create a certain level of authorisation. All the agents approved by us will have their certificate signed by our root certificate. Since this is the only certificate trusted in our example, all agents that need to communicate securely need to be signed with this root certificate. The restriction is that all secure agents must trust this root certificate and the CA. This might not be the case and therefore it is generally better to use publicly and worldwide trusted CAs.

All secure agents must have the EstablishSecurityCS role. This role is automatically created when the secure agent is initialised at start-up. When a StartPlayingMsg message is received at the role's state machine, it enters the composite state using the entry point, ENTRY_WAIT_FOR_REQUEST, as shown in Figure 6.1-4 under the "Implementation" chapter. The composite state shown in Figure 6.1-3 is responsible for listening for incoming SecureConnectionRequestMsg messages.

In our test case the TestCase agent first sends a SecureConnectionRequestMsg message with its own certificate, using our protocol as Figure 6.1-2 in the previous chapter.

The TestCase's default state machine then enters the composite state SecureConnectionState through the entry point ENTRY_WAIT_FOR_REPLY as shown in Figure 7.3-1.

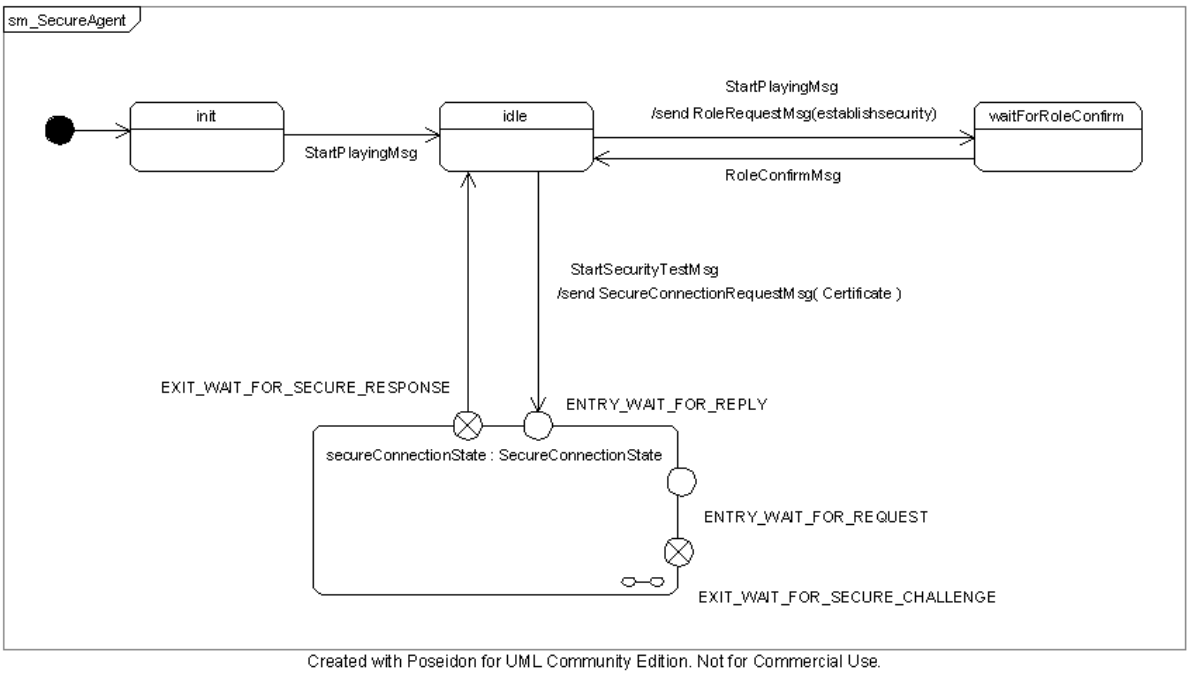

Figure 7.3-1 State diagram for the SecureAgent

All secure agents will have a default state machine, like the one all actors have, and a state machine for the EstablishSecurityCS role. The EstablishSecurityCS state machine is used as a "server" process, as it is waiting for incoming secure connection requests. The default state machine will of course contain normal user code. The setting up of a new secure connection will have to be specified in the default state machine by the programmer. As mentioned earlier, the programmer must make sure the default state machine enters the composite state SecureConnectionState through the right entry point in Figure 7.3-1 after sending the SecureConnectionRequestMsg message. Basically this means that the EstablishSecurityCS is used for accepting secure connection request, but the actual request is performed in the default state machine by the programmer.

7.4 The creation of certificates and keys

In our implementation we us OpenSSL to create CA root sign certificates, meaning that there is a CA issuing the certificate for a party. This party can add other parties to its chain. As mentioned earlier we use the keystore as a file that holds the data for the private keys and certificates. Keytool is, as previously mentioned, a utility supplied by the Java Runtime Environment (JRE) and is a command-line interface. We use the standard X.509 for the certificates.

Keytool

The Java Keytool supplied with the Java Runtime Environment (JRE) and the Java Software Development Kit (JDK) can be used for both creation of certificates and a private/public key pair. Keytool has a limitation which is that it is only able to create certificates in the X.509 version 1 format although it may import the X.509 version 3.

Create a keystore with a key entry:
To create a certificate we use the following command in command-line:

```
keytool -v -genkey -alias Alice -keyalg RSA -keysize 1024 -keystore
ericsson.keystore -validity 180
```

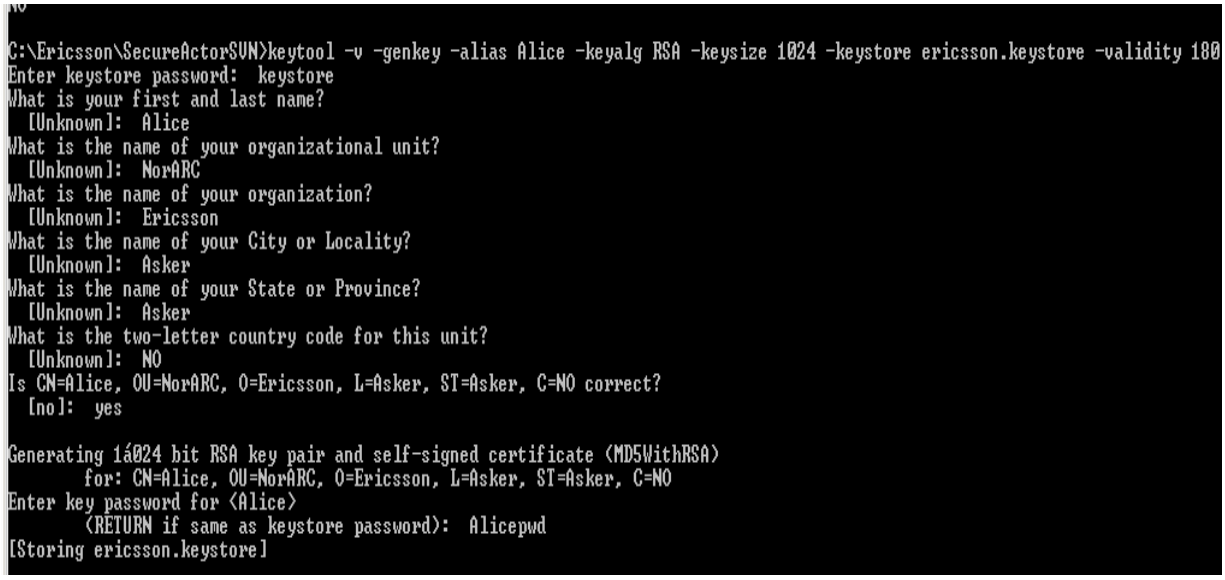

Figure 7.4-1 Creation of a certificate and a key pair using Keytool

The command Keytool starts the whole operation. The "-v" option gives additional (verbose) information. The "-genkey" option tells Keytool to generate a key pair and a certificate and to place it to the keystore. The '-alias' option tells Keytool which alias it should create for the actual certificate in the keystore. The default alias is "mykey". "-Keyalg" is the next option and tells Keytool which algorithm to use for the public and private key. The next option "-Keysize" tells Keytool which key bit-size it should use when it generates the key. The last option "-keystore" specifies the name of the keystore file and the path is the default directory unless otherwise specified. If the keystore does not exist it will be created. The last option tells the length of validity of the key, in days. The default value is 90 days.

In the previous example the entries were typed in manually, but this can be done automatically. This option is called "-dname" and lets you specify the distinguished name of the certificate. This makes it easy if you want to put the command in a script file. It looks like this:

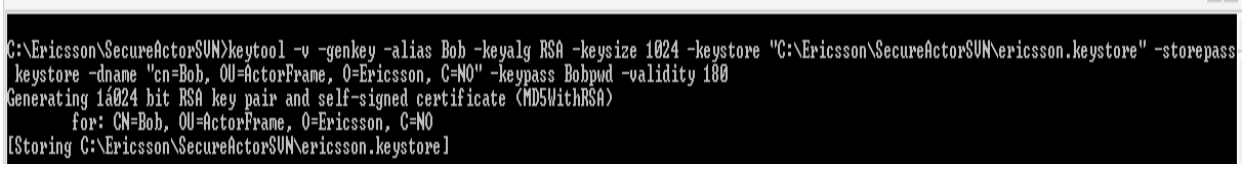

Figure 7.4-2 Creation of a certificate using Keytool

Export a certificate request

The following line is entered in the command-line:
keytool -certreq -rfc -v -alias Alice -keypass Alicepwd -keystore ericsson.keystore -storepass keystore -file AliceCertReq.csr

How to implement CA signed certificate

1. Create a working directory that will hold the actual information about the certificate
2. Copy the file openssl.cnf to the working directory

3. Rename the openssl.cnf ending to an endname in console window to e.g. conf, so that you may access it in textpad or something similar
4. Create an empty file with the name index.txt
5. Put ”01” into the empty file named serial
6. We then have to change some lines in the config file openssl.conf. The actual changes are shown below

```
[ CA_default ]

dir     = .                # <--CHANGE THIS
certs       = $dir/certs
crl_dir     = $dir/crl
database    = $dir/index.txt
#unique_subject = no

new_certs_dir   = $dir/newcerts

certificate = $dir/certs/myca.crt   # <--CHANGE THIS
serial      = $dir/serial
#crlnumber  = $dir/crlnumber

crl     = $dir/crl.pem
private_key = $dir/private/myca.key    # <--CHANGE THIS
RANDFILE    = $dir/private/.rand

x509_extensions = usr_cert
```

The configuration is taken from [116]

7. A certificate is made by entering the following command in the command-line: openssl genrsa –out ServiceFrameCA.key 1024
8. Create a CA self-signed certificate using openssl. The command you enter in the command-line is : openssl req -x509 -new -out ServiceFrame.crt –sha512 -key ServiceFrameCA.key -days 365

   It looks like this in the command-line:

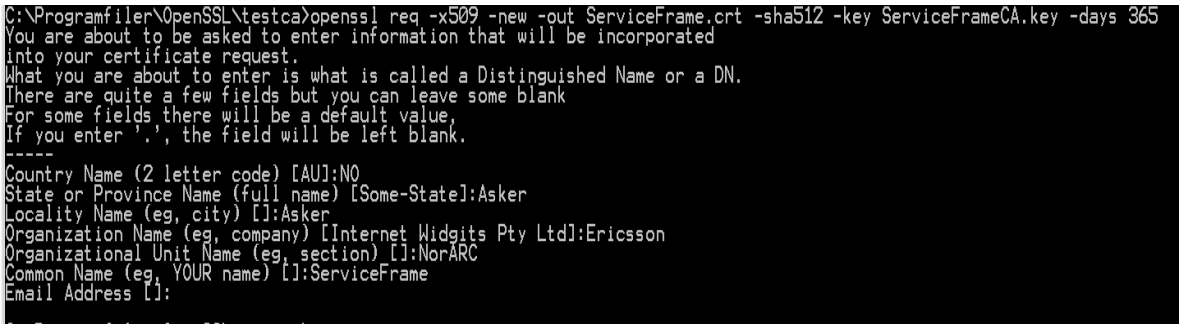

Figure 7.4-3 Creation of a root certificate

9. Create a certificate with keytool and export a certificate request:
   keytool -genkey -v -keystore ericsson.keystore -storepass keystore -alias Bob -keypass Bobpwd -dname "CN=Bob, OU=ServiceFrame, O=Ericsson, L=Asker, ST=Asker, C=NO" -validity 360 -keyalg RSA -keysize 1024
   keytool -genkey -v -keystore ericsson.keystore -storepass keystore -alias Alice -keypass Alicepwd -dname "CN=Alice, OU=ActorFrame, O=Ericsson, L=Asker, ST=Asker, C=NO" -validity 360 -keyalg RSA -keysize 1024

keytool -certreq -v -alias Bob -keypass Bobpwd -keystore ericsson.keystore -storepass keystore -file BobCertReq.csr
keytool -certreq -v -alias Alice -keypass Alicepwd -keystore ericsson.keystore -storepass keystore -file AliceCertReq.csr

10. Import the CA self-signed certificate. The following command is entered in command-line: keytool -v -import -file ServiceFrame.crt -keystore ericsson.keystore -storepass keystore -alias ServiceFrame

    The following appears in the command-line window and is executed:

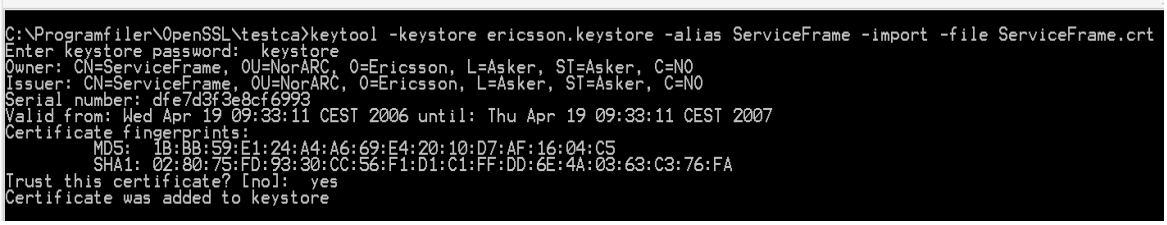

Figure 7.4-4 Importing the CA root certificate

11. CA signs the certificates, and the following command is entered in command-line: openssl x509 -req -in AliceCertReq.csr -CA ServiceFrame.crt -CAkey ServiceFrameCa.key -CAcreateserial -out AliceSigned.crt -days 360
12. Import then the CA signed certificates to the keystore. The following command is entered in the command-line:
    keytool -import -v -alias Alice -keypass Alicepwd -keystore ericsson.keystore -file AliceSigned.crt -storepass keystore

7.5 Execution and simulation

For the User Sends SMS test case we used run the test in the standalone version of ServiceFrame for the J2SE platform. We have not conducted a test on the J2EE platform or the J2ME mobile platform. We have concentrated on making the test case work using the J2SE platform. Considerable changes would not be needed to make it run on the J2EE platform, but for the J2ME platform a few changes will be required. We have used the Ericsson Network Resource Gateway (NRG) Simulator [113] for simulating the telecommunication network. The NRG Simulator can be used to simulate telephone calls and to simulate sending and receiving SMS messages. The NRG edges in ServiceFrame can be configured to connect to the NRG Simulator instead of a NRG Parlay gateway. As mentioned earlier we use the NRG simulator in our testing.

Simulation setup

The simulation setup consists of three actors: “alice”, “secureSmsEdge” and “eve”. A mobile phone with phone number 200 is also simulated in the NRG simulator. The agent “alice” represents the user who wants to send a secure message over the Internet. The agent “secureSmsEdge” is the receiving agent, whereas the agent “eve” is the “man-in-the-middle”. The agent “alice” is an instance of TestCase. The agent “secureSmsEdge” is an instance of SecureSmsEdge and “eve” is an instance of ManInMiddle”. All classes inherit the functionality of SecureAgent. In the test configuration all messages sent from “alice” are sent through “eve” and then to “secureSmsEdge”. This way we can test whether or not “eve” can intercept or alter the messages along the way. The agent “eve” simulates what can happen when a message is sent through an insecure channel or what kind of attacks someone in the

middle can perform. If she can successfully attack, then the protocol is not secure. The Figure 7.5-1 illustrates the experiment setup.

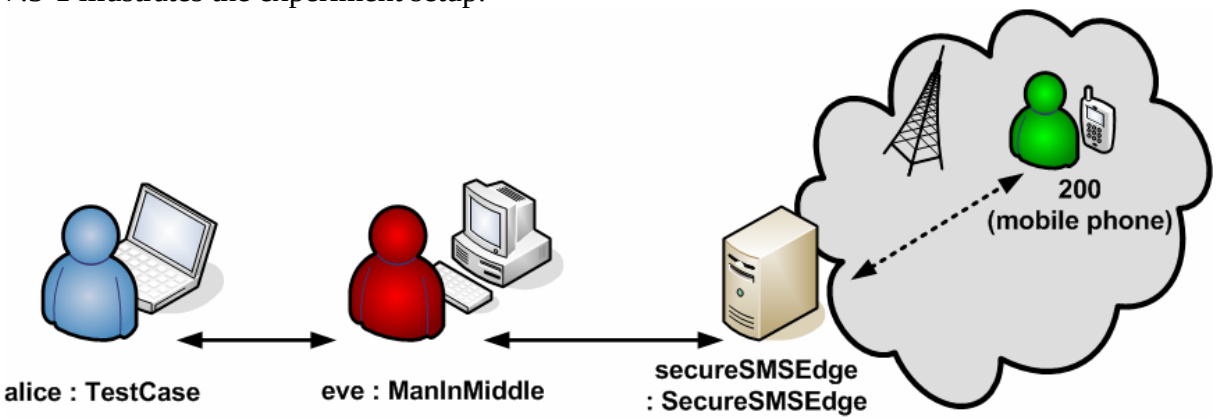

Figure 7.5-1 The "User Sends SMS" test scenario

We have only chosen to test a couple of the attack types such as "man-in-the-middle" and Replay attacks, due to limited time. In the "man-in-the-middle" attack we test attacks against confidentiality, integrity and authentication.

Test runs

We have tested the test case in four different test runs. We performed the tests without the security mechanisms and using the security mechanisms for each test run.

Test run 1: Information disclosure

1 A) We first let Alice send an unencrypted or plaintext message to secureSMSEdge passing through Eve.

Figure 7.5-2 shows the Graphical User Interface (GUI) of the Test Case agent, "alice". Alice can choose to send messages unencrypted or encrypted using the security mechanisms. She can send an SMS message to the phone number in the text field. The message can be typed in the other text field. In this test Alice sends message "This is a secret message!" to phone number "200" unencrypted.

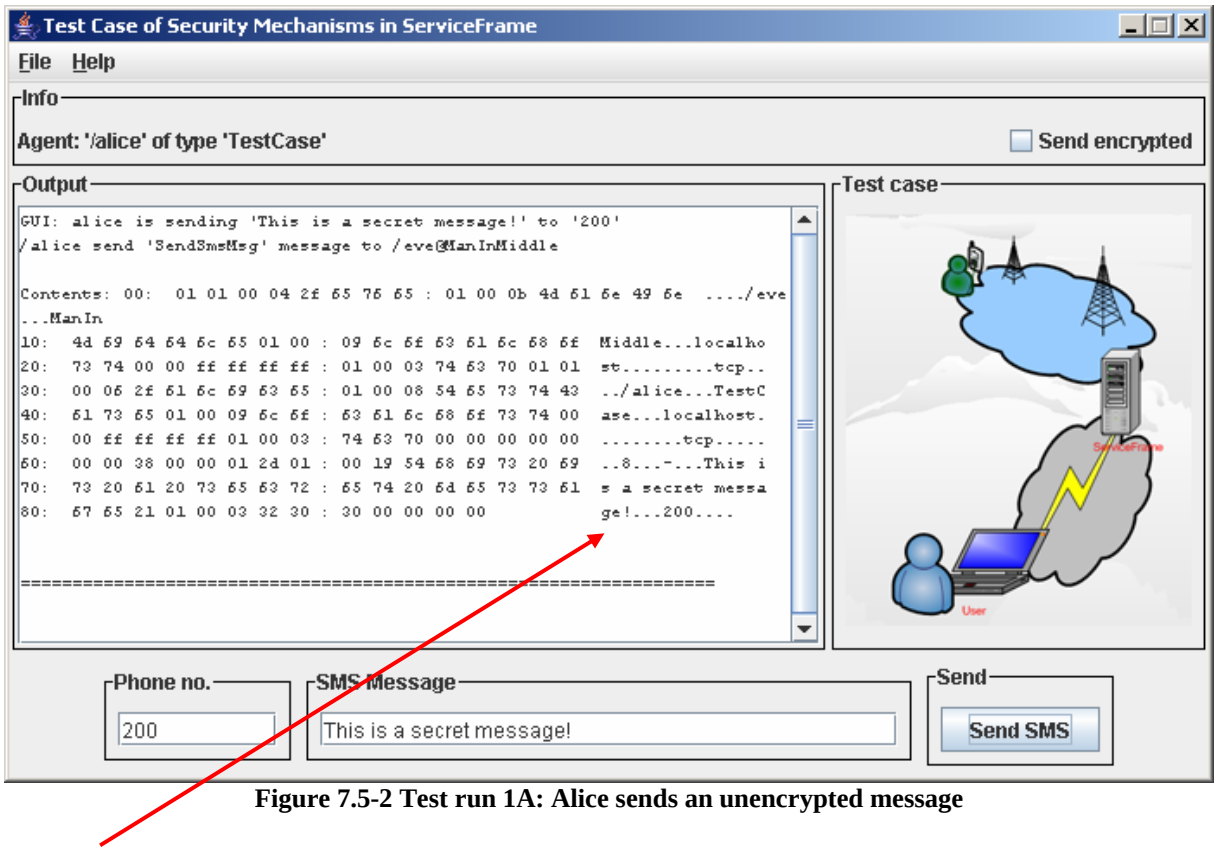

Figure 7.5-2 Test run 1A: Alice sends an unencrypted message

The message contents is sent as plaintext.

1 A) Result

The result of the test in Figure 7.5-3 shows that the "Man-in-the-middle" Eve can see the message because it is not encrypted.

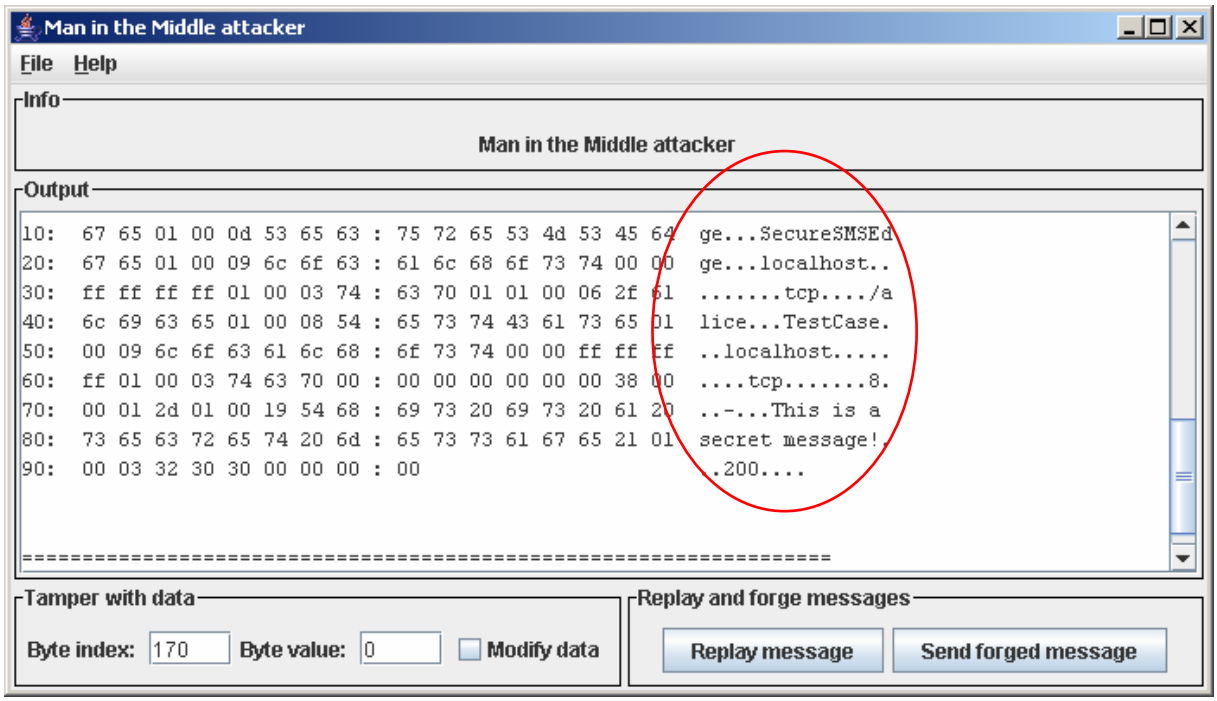

Figure 7.5-3 Result of test run 1A: Eve can see the message sent in plaintext

1 B) We then let Alice send an encrypted message, using the security mechanisms and the security protocol, to the SecureSMSEdge through Eve.

Figure 7.5-4 shows that the same message is sent as in the in the previous test. The only difference is that it is encrypted.

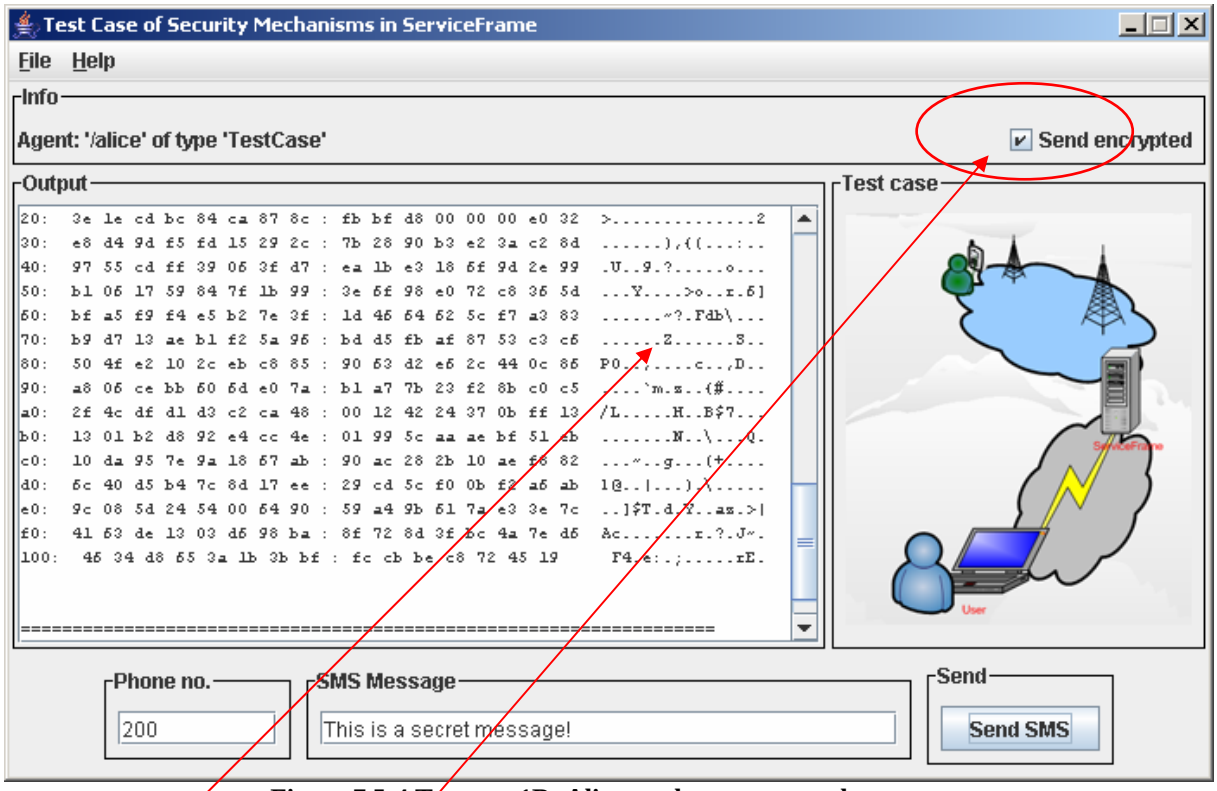

Figure 7.5-4 Test run 1B: Alice sends an encrypted message

The message output is encrypted.

1 B) Result

Figure 7.5-5 shows the result of the test run. The message is encrypted and Eve cannot see what the message contents are.

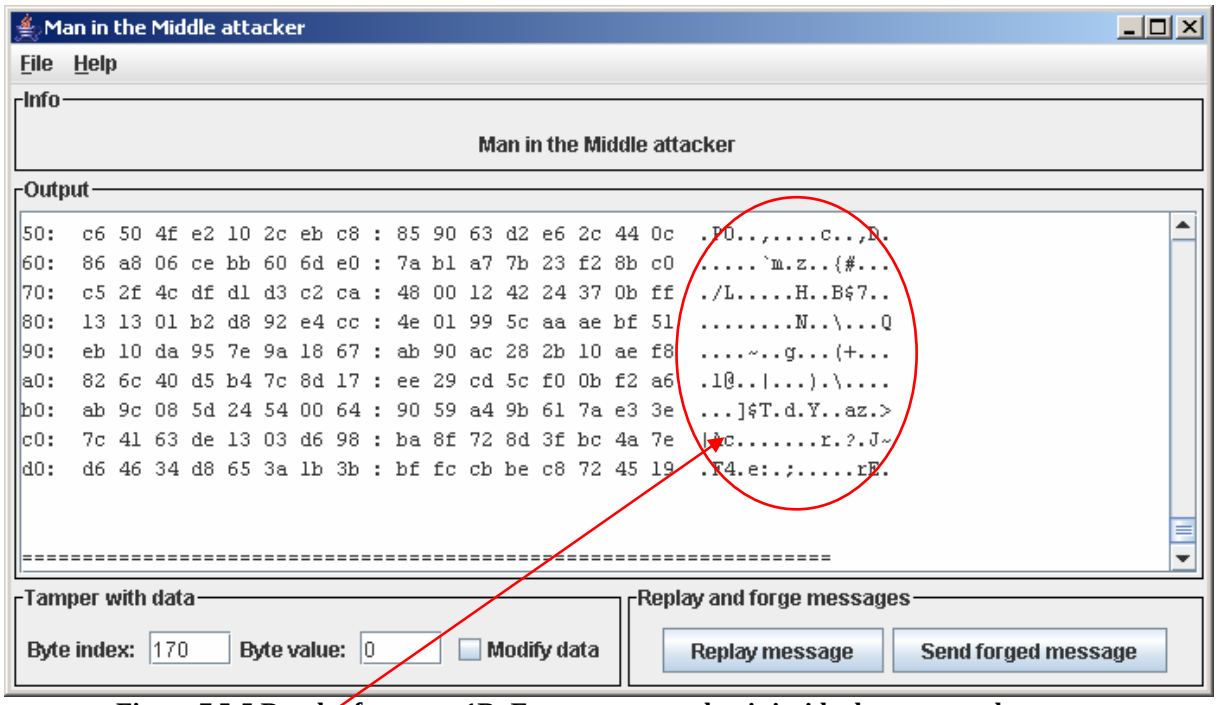

Figure 7.5-5 Result of test run 1B: Eve cannot see what is inside the encrypted message

Eve cannot see the message contents.

Test run 2: Tampering with data

2 A) We let Alice send an unencrypted message to the secureSMSEdge via Eve, but this time Eve chooses to modify the message.

Figure 7.5-6 shows that Alice is sending a plaintext message to the phone number “200”.

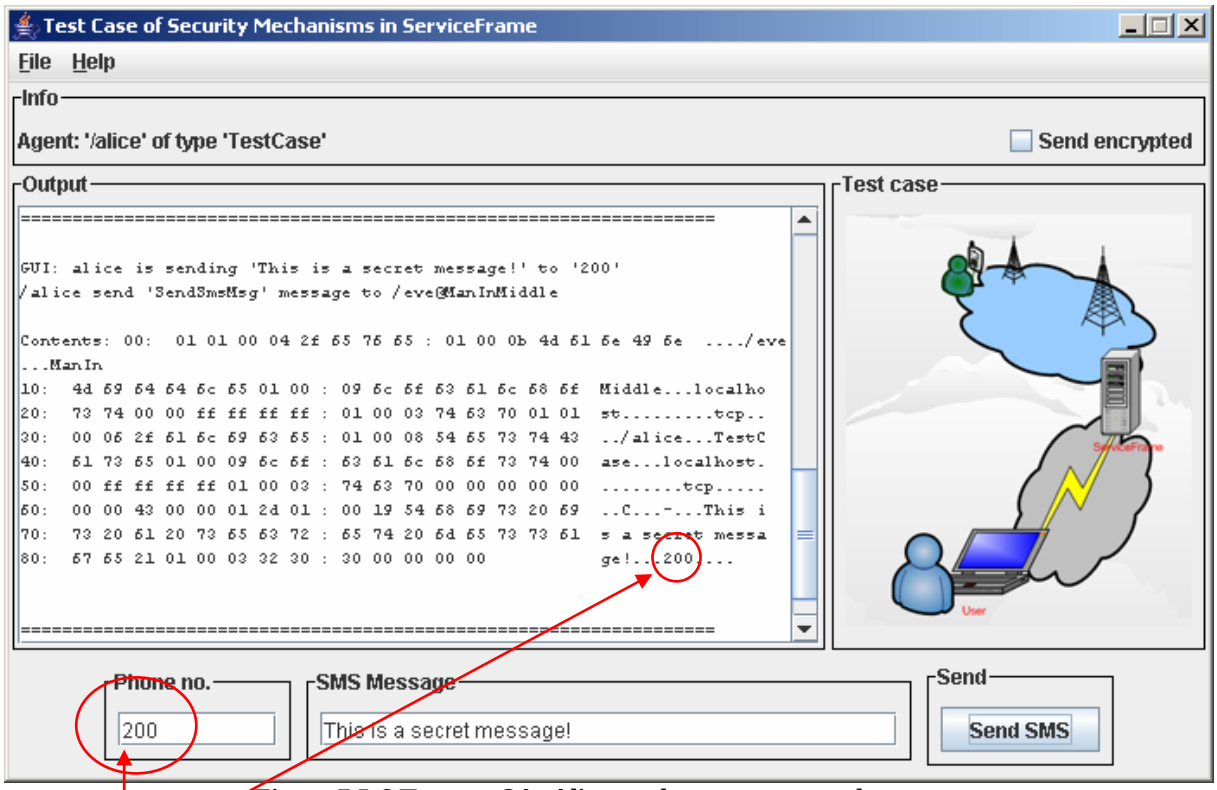

Figure 7.5-6 Test run 2A: Alice sends an unencrypted message

Alice sends a message to phone number “200”.

Figure 7.5-7 shows that Eve has modified byte at index 146 in the message and replaced a byte with the value 49 which is a “1”. Eve has modified the message and changed the phone number from “200” to “100”.

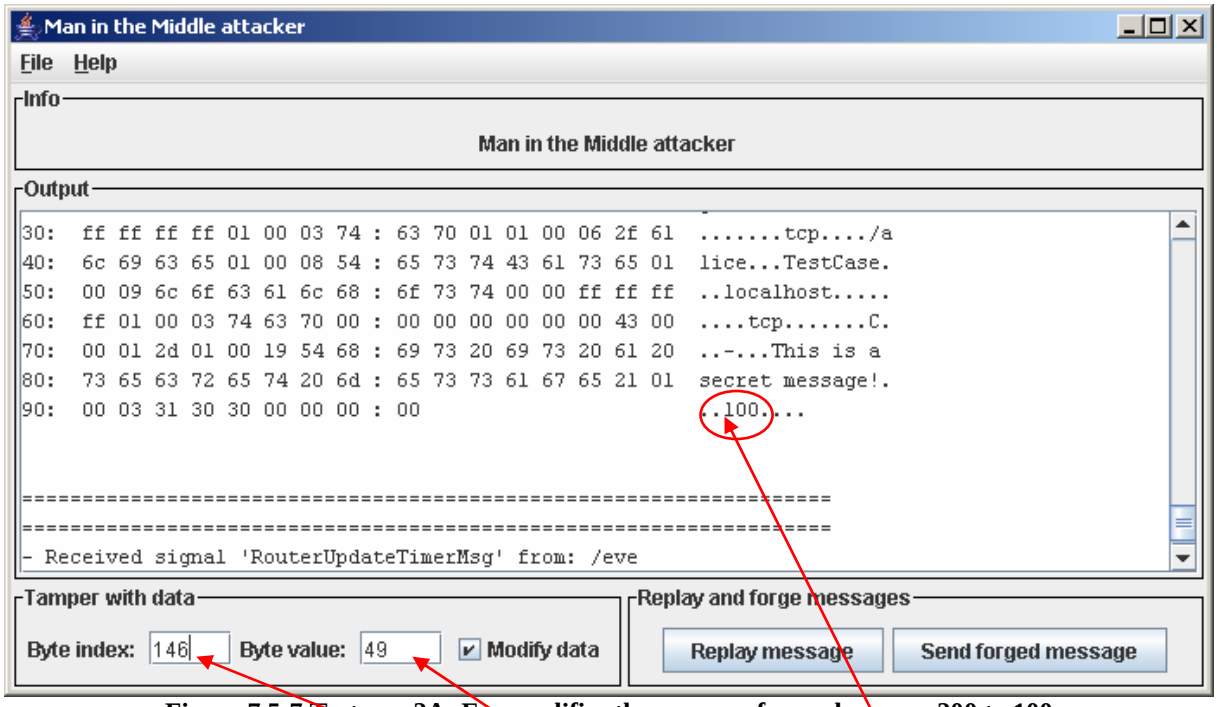

Figure 7.5-7 Test run 2A: Eve modifies the message from phone no. 200 to 100

Eve changes the value at position 146 to 49, which is the character “1”.

2A) Result

Figure 7.5-8 shows that in the NRG simulator the phone number “100” received the SMS message, and not the phone number “200”. The message Alice sent did not reach its destination.

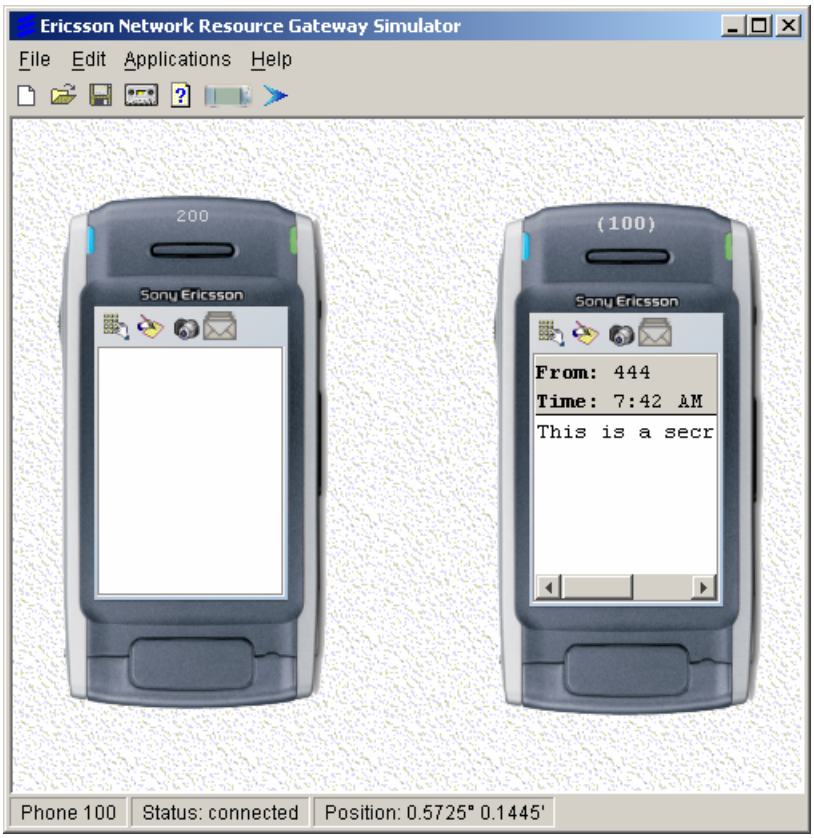

Figure 7.5-8 Result of test run 2A: The mobile no. 100 receives the message

2B) We then let Alice send an encrypted message using the security mechanisms and the security protocol. Eve is still the “man-in-the-middle” and she tries to tamper with the message contents.

Figure 7.5-9 shows that Alice now sends a message to phone number “200”, but this time it is encrypted and signed with a HMAC.

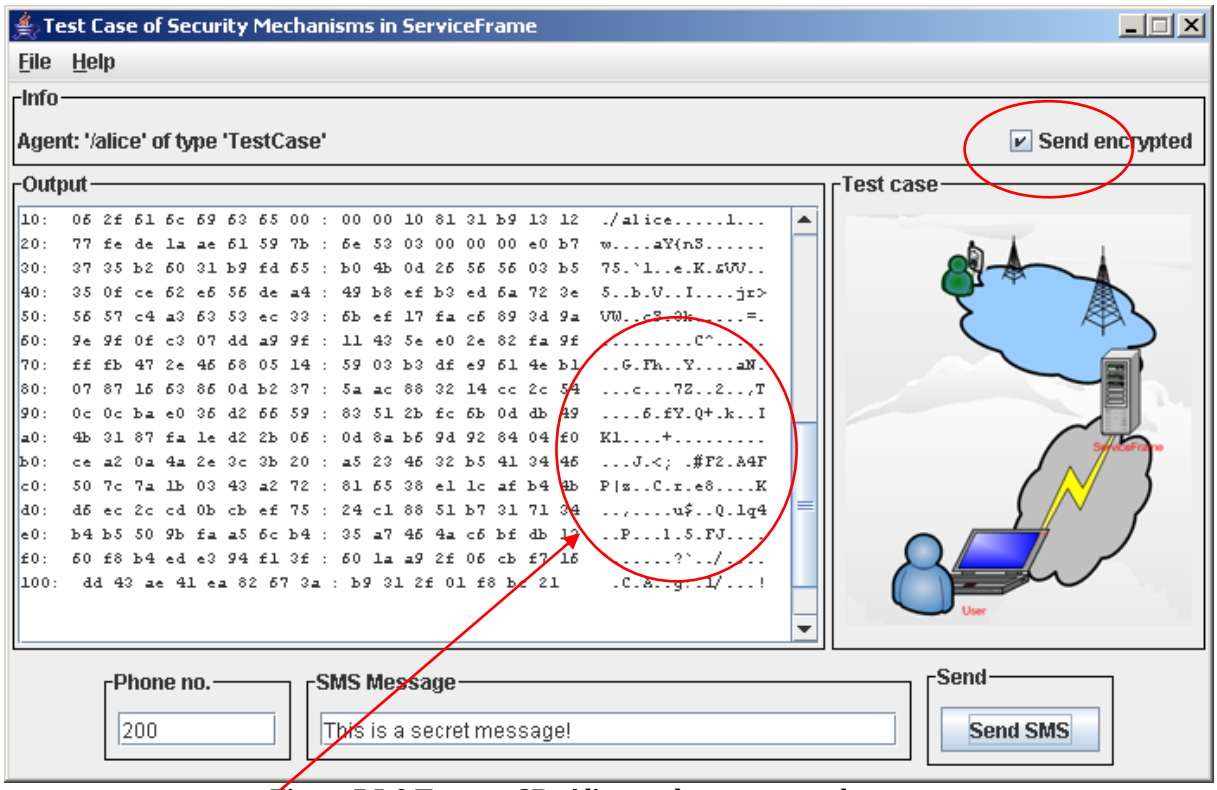

Figure 7.5-9 Test run 2B: Alice sends an encrypted message

The message is encrypted.

Figure 7.5-10 shows that Eve tries to modify the byte at index 170 and set this to zero. The message is encrypted so she does not know what the message contents are.

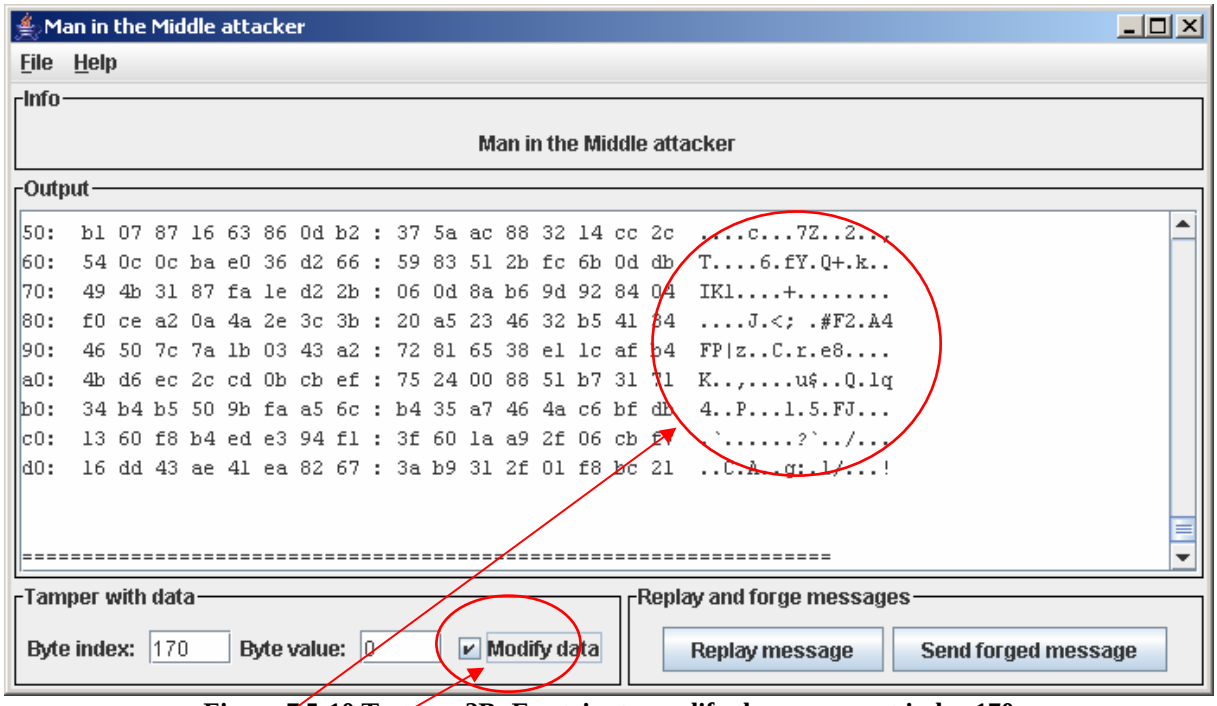

Figure 7.5-10 Test run 2B: Eve tries to modify the message at index 170

Eve tries to modify the encrypted message.

2 B) Result

Figure 7.5-11 shows that the SecureSMSEdge rejects the message because the HMAC from the message is not equal to the calculated HMAC using the session key. The message must have been tampered with.

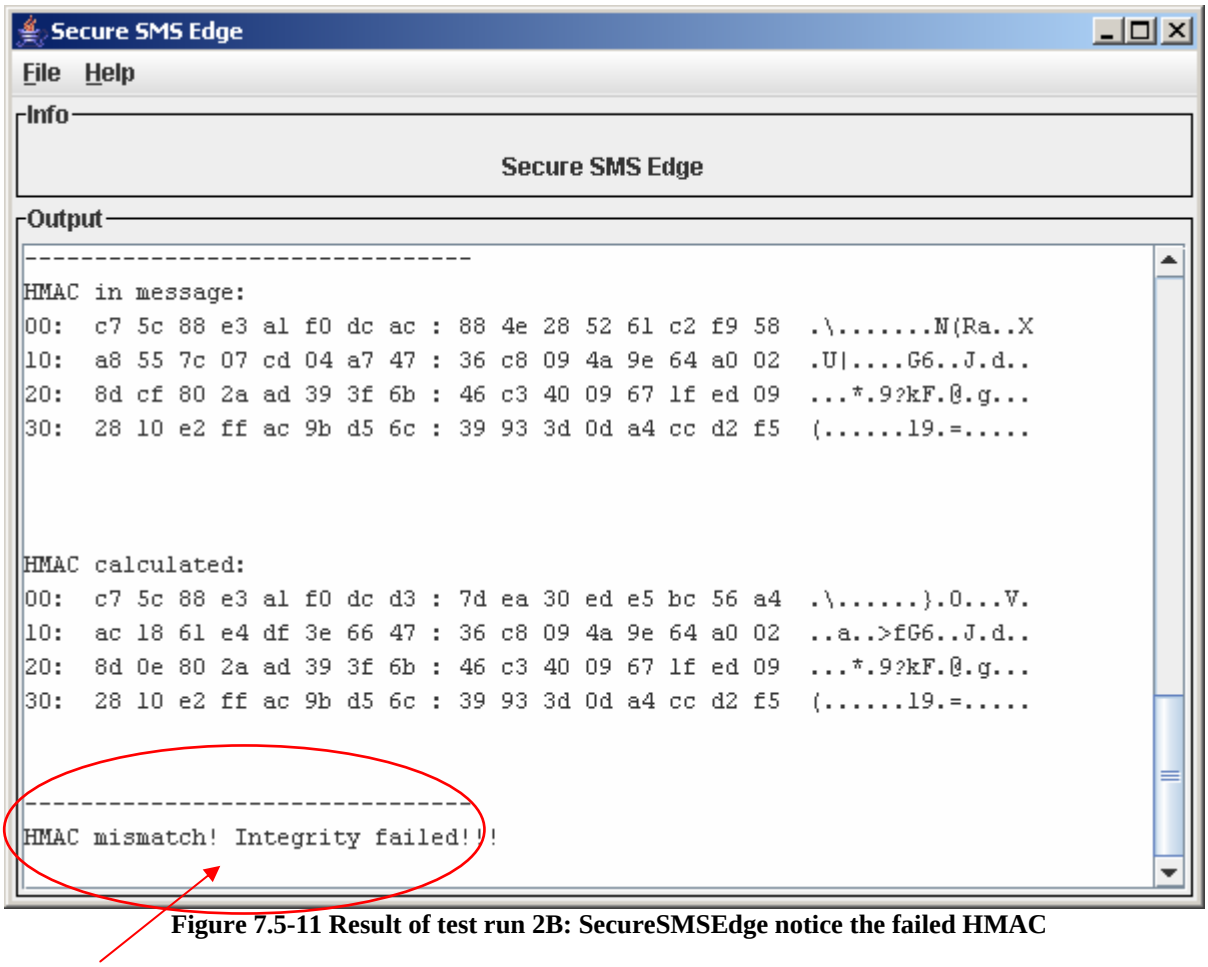

Figure 7.5-11 Result of test run 2B: SecureSMSEdge notice the failed HMAC

The tampering with the encrypted message contents failed.

Test run 3: Spoofing identities

3 A) We let Eve try to create a message and then send it to the SecureSMSEdge using Alice's actor address.

Figure 7.5-12 Shows that Eve is forging a message and pretending it came from Alice.

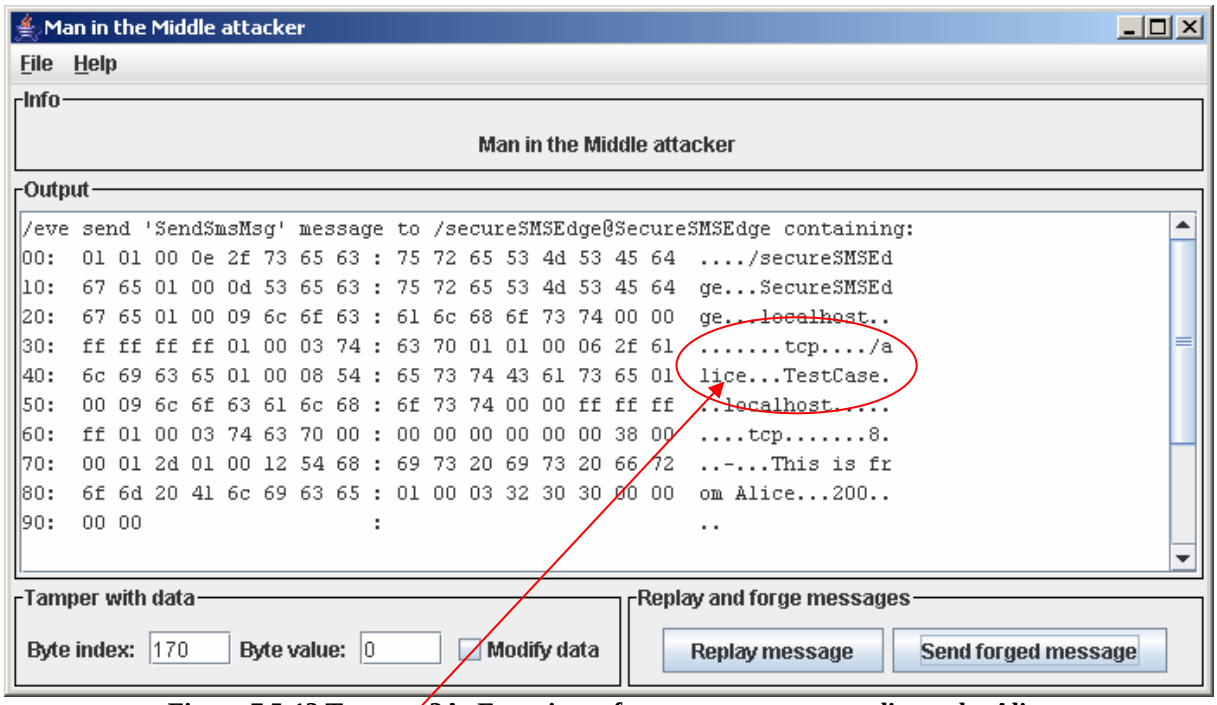

Figure 7.5-12 Test run 3A: Eve tries to forge a message pretending to be Alice

The sender address is changed to "/alice".

3A) Result

Figure 7.5-13 Shows that the SecureSMSEdge incorrectly thinks this message was sent from Alice.

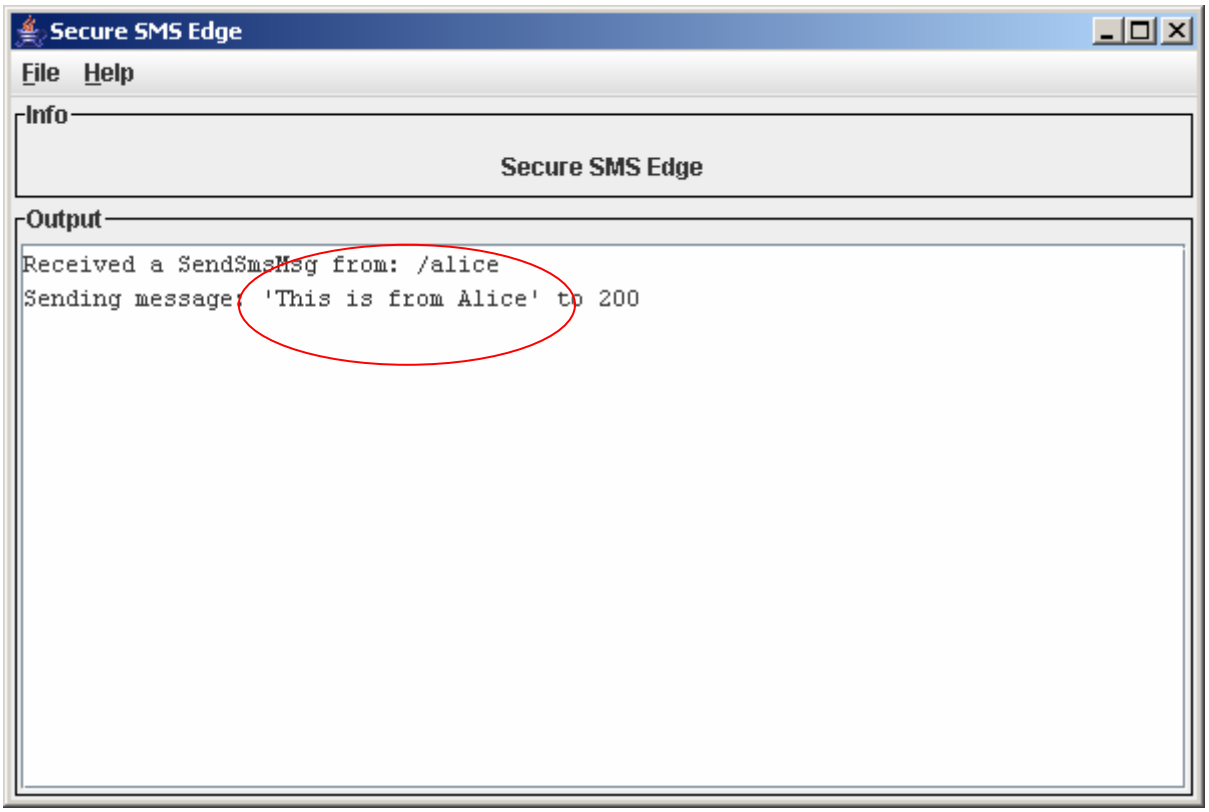

Figure 7.5-13 The SecureSMSEdge incorrectly identifies Eve as Alice

3B) We then let Eve try to impersonate Alice using a fabricated certificate created with a false CA and root certificate. She then tries to start a secure connection with the SecureSMSEdge.

Figure 7.5-14 shows that Eve is trying to send a forged certificate that is not signed with a trusted root certificate.

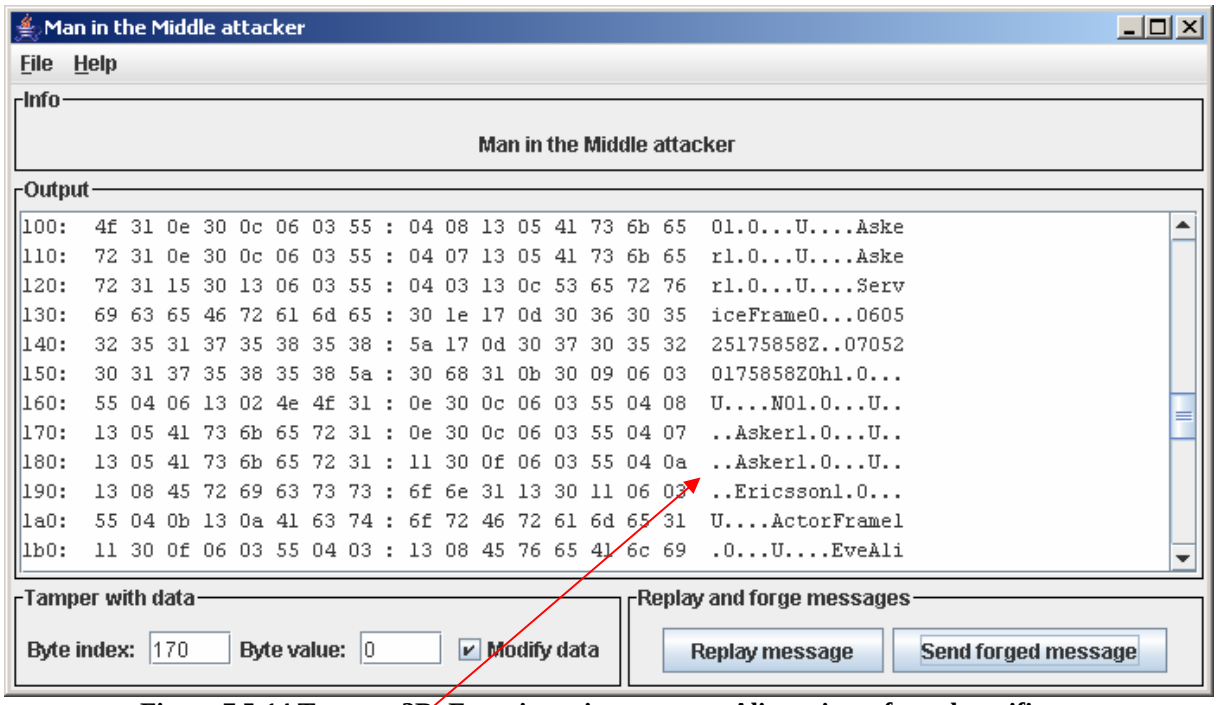

Figure 7.5-14 Test run 3B: Eve tries to impersonate Alice using a forged certificate

Eve sends a forged certificate.

3 B) Result

The result is given below. The console output shows that the certificate was rejected because it was not signed by a trusted CA.

```
The certificate was "signed" by CN=EveAlice, OU=ActorFrame, O=Ericsson, L=Asker, ST=Asker,
C=NO
The Certificate was provided by CN=ServiceFrame, L=Asker, ST=Asker, C=NO
2006-05-29 04:02:21,578 DEBUG [.eve]
START TRANSITION NO: 41 ACTOR: /eve@ManInMiddle TRIGGER: StartTestCaseMsg CURRENT STATE:
maninmiddle /idle NEW STATE:  maninmiddle /secureConnectionState/waitForSecureConnectionReply
  INPUT:   MESSAGE: ID: 96 NAME:StartTestCaseMsg RECEIVER:/eve@ManInMiddle
SENDER:/eve@ManInMiddle CONTENT:  StartTestCaseMsg contents: '/secureSMSEdge'
  OUTPUT:  MESSAGE: ID: 97 NAME:SecureConnectionRequestMsg
RECEIVER:/secureSMSEdge.establishsecurity@SecureSMSEdge SENDER:/eve@ManInMiddle CONTENT:
SecureConnectionRequest contents: ''
  TASK:    enterState called: entry number: 1
END TRANSITION ACTOR: /eve@ManInMiddle NEW STATE:  maninmiddle
/secureConnectionState/waitForSecureConnectionReply
------------------------------------------------------------------->
Trying to verify certificate
Check if CN=ServiceFrame, L=Asker, ST=Asker, C=NO == CN=SecureSMSEdge, OU=ActorFrame,
O=Ericsson, L=Asker, ST=Asker, C=NO
Check if CN=ServiceFrame, L=Asker, ST=Asker, C=NO == CN=ServiceFrame, L=Asker, ST=Asker, C=NO
The certificate was signed by an unknown signer
actor.security.AFCertificateNotVerifiedException
        at actor.security.AFCertificateHandler.processCertificate(AFCertificateHandler.java:50)
        at actor.security.SecureConnectionState.execTrans(SecureConnectionState.java:147)
        at se.ericsson.eto.norarc.agentframe.AgentSM.exec(AgentSM.java:248)
        at se.ericsson.eto.norarc.ejbframe.StateMachine.processMessage(StateMachine.java:597)
        at se.ericsson.eto.norarc.ejbframe.StateMachine.simOnMessage(StateMachine.java:386)
        at se.ericsson.eto.norarc.ejbframe.Scheduler$Worker.run(Scheduler.java:96)
        at util.threadpool.ThreadPoolThread.run(ThreadPoolThread.java:30)
        at java.lang.Thread.run(Thread.java:595)

Certificate rejected
```

The results of test run 3B:
The SecureSMSEdge receives a “AFCertificateNotVerifiedException” and rejected the forged certificate since it has not been signed by a trusted CA.

Test run 4: Replay attacks

4A) We let Alice send a message unencrypted to the SecureSMSEdge. We then let Eve try to replay the message, pretending Alice sent the message twice.

Figure 7.5-15 shows that Alice sends a message in plaintext.

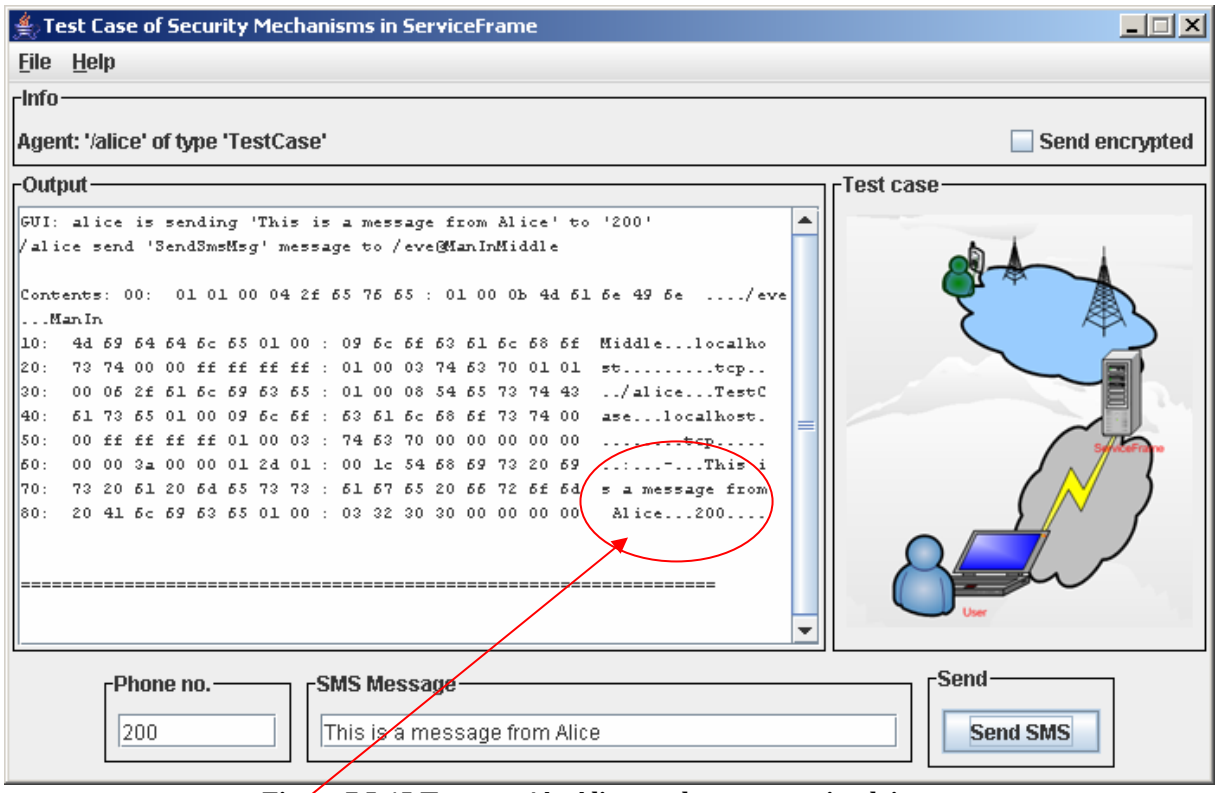

Figure 7.5-15 Test run 4A: Alice sends a message in plaintext

Alice sends a message containing “This is a message from Alice”.

Figure 7.5-16 shows that Eve can see this message and try to replay it.

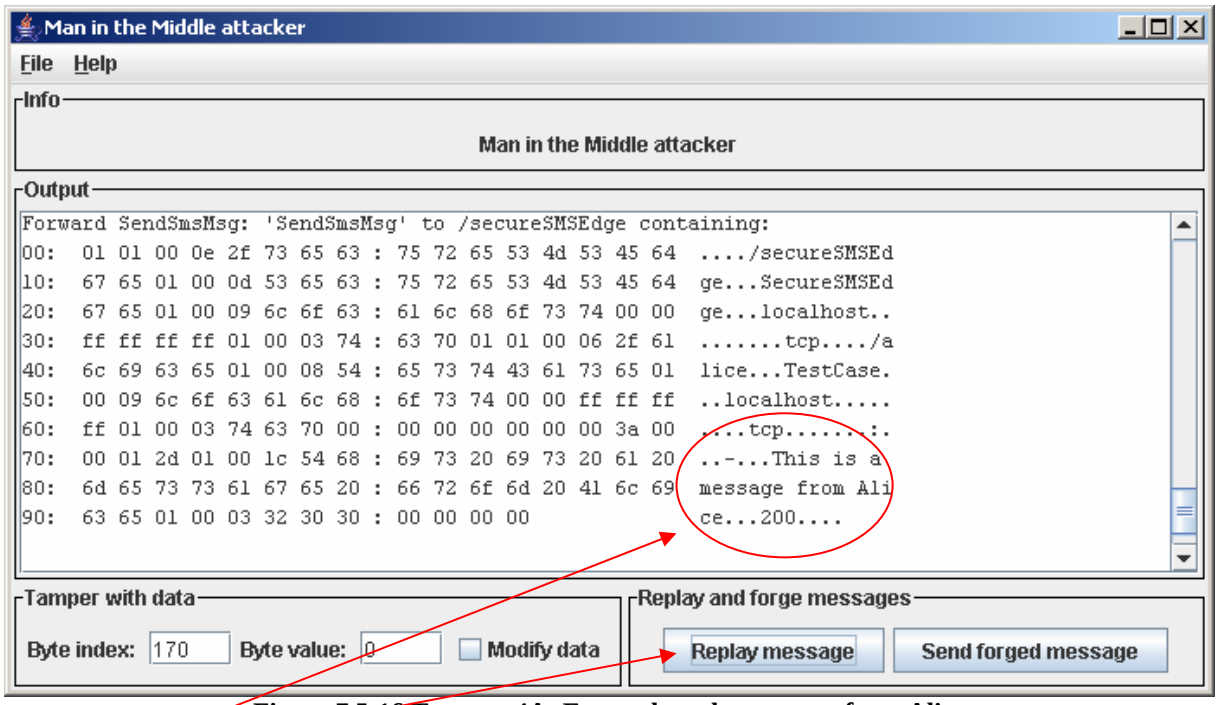

Figure 7.5-16 Test run 4A: Eve replays the message from Alice

Eve can replay the message.

4A) Result

Figure 7.5-17 shows that a message from Alice was accepted twice, even though Alice only sent it once. This is because Eve replayed the message.

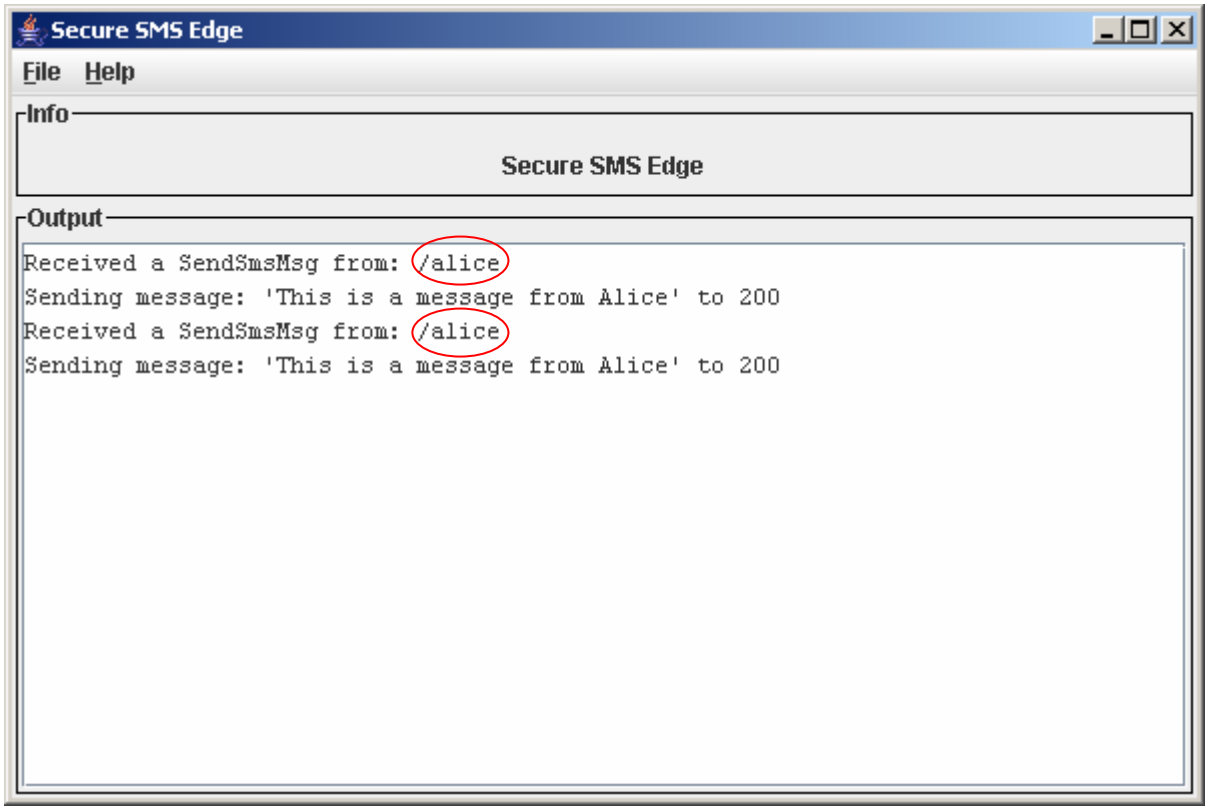

Figure 7.5-17 Result of test run 4A: The message is replayed

4B) We let Alice send an encrypted message to the SecureSMSEdge using the security protocol. We then let Eve try to replay the message.

Figure 7.5-18 shows that Alice sends an encrypted message using the security mechanisms.

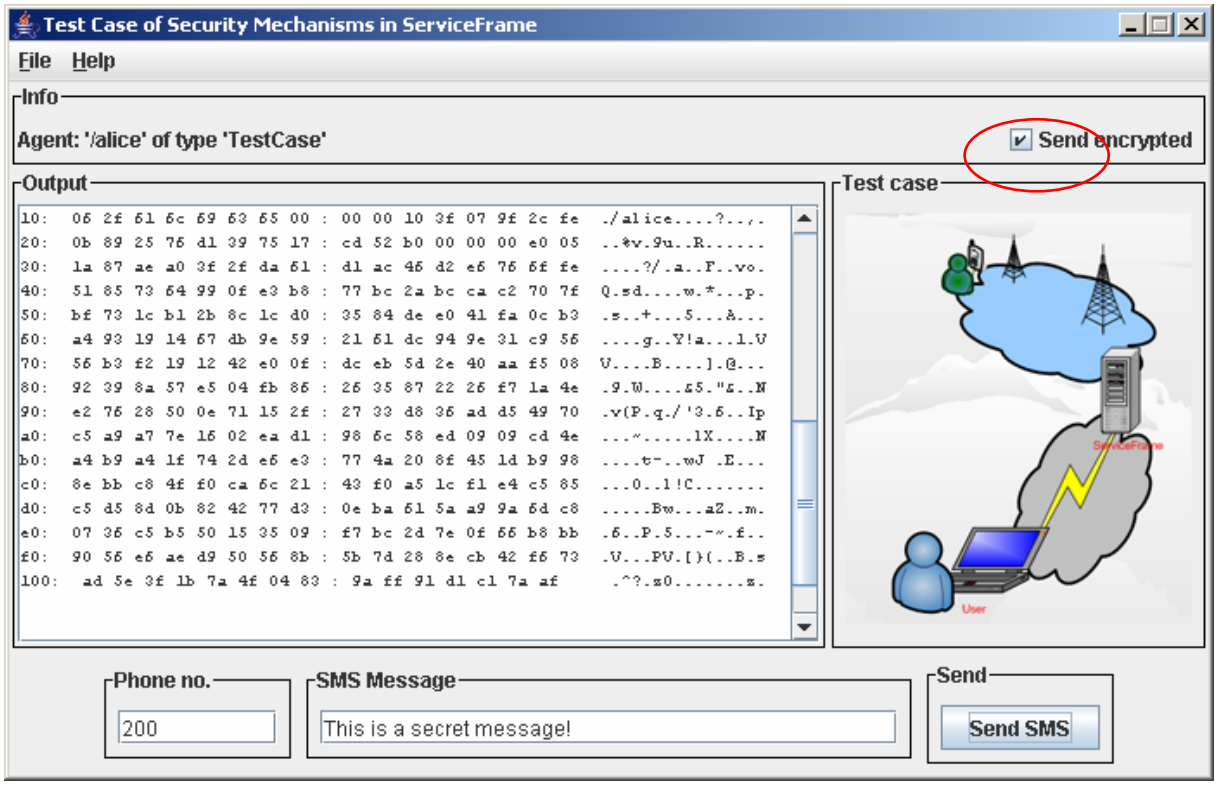

Figure 7.5-18 Test run 4B: Alice sends an encrypted message

Figure 7.5-19 shows that SecureSMSEdge accepts the message because it came from Alice.

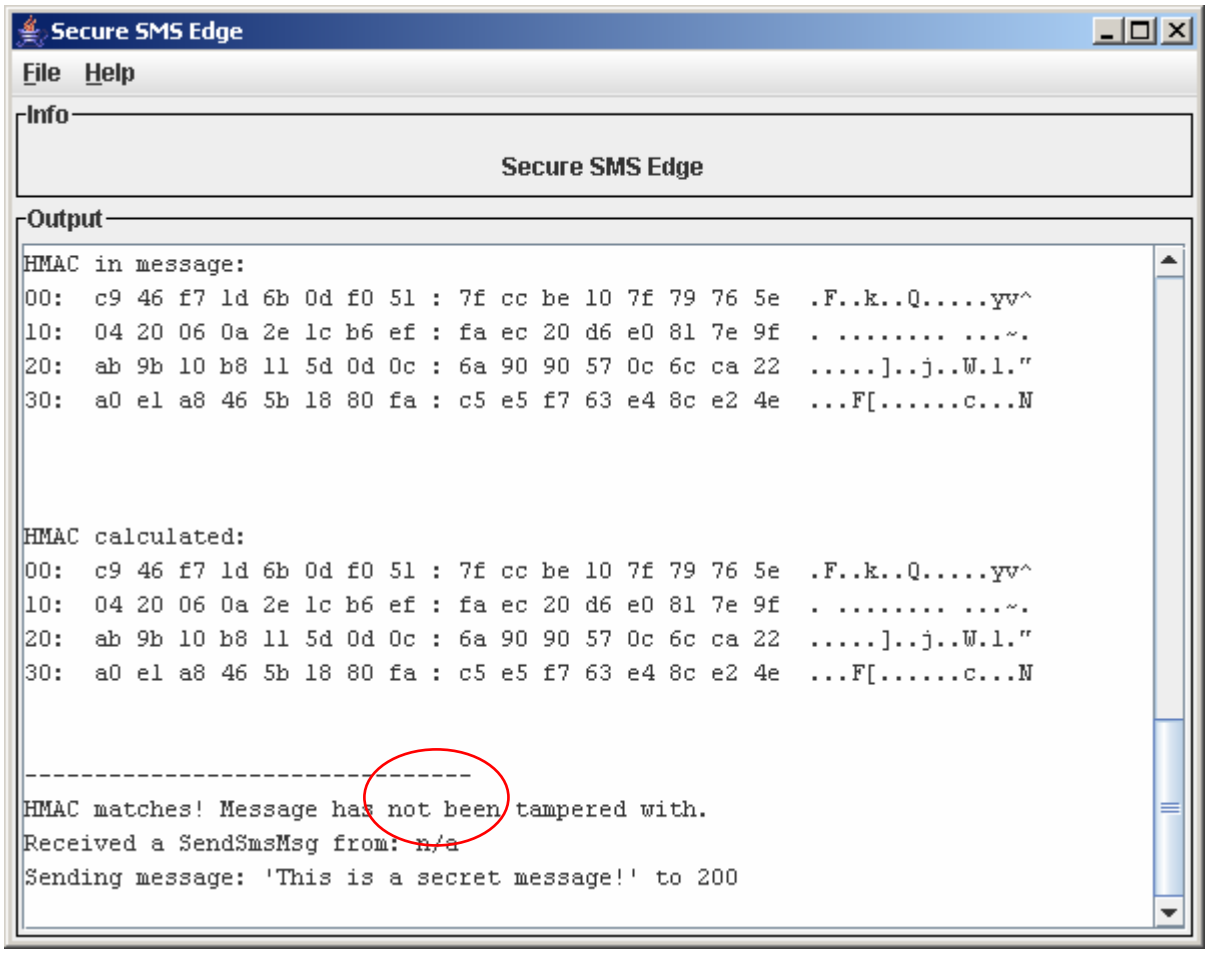

Figure 7.5-19 Test run 4B: The SecureSMSEdge accepts the message from Alice

Figure 7.5-20 shows that Eve cannot see the message contents but she can still try to replay the message.

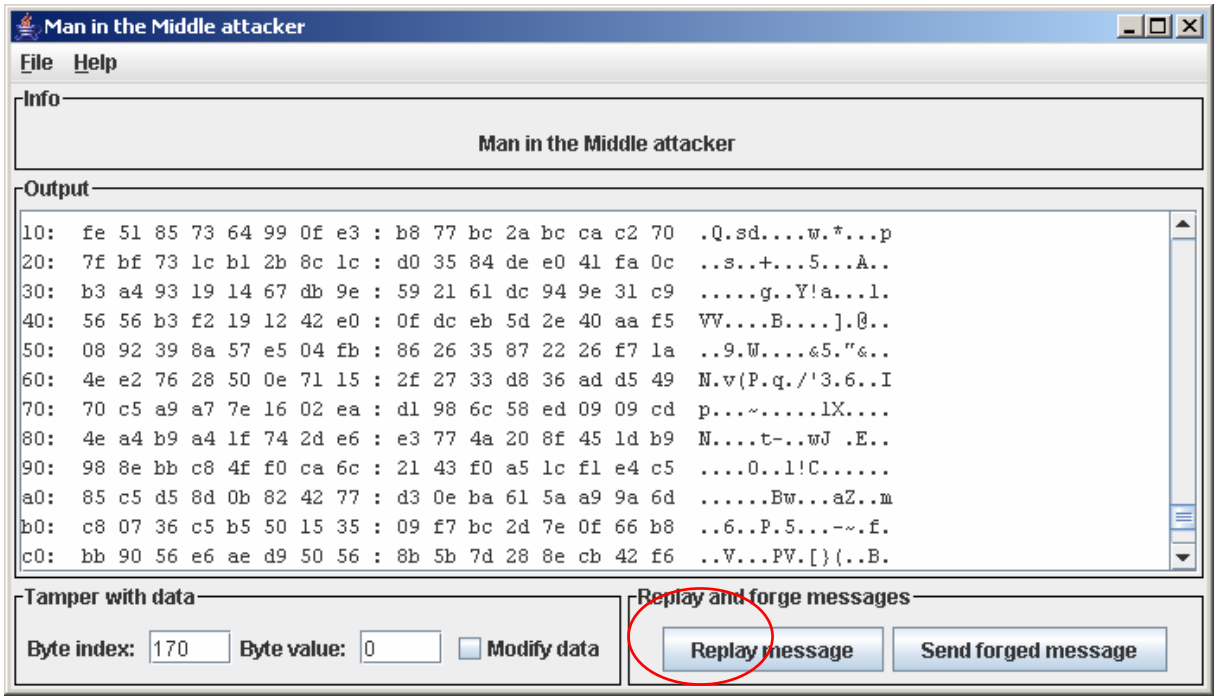

Figure 7.5-20 Test run 4B: Eve replays the message from Alice

4B) Result

Figure 7.5-21 Shows that the message from Eve has a valid HMAC because it is the same message Alice sent. The SecureSMSEdge cannot separate between the original message from Alice and the replayed message from Eve.

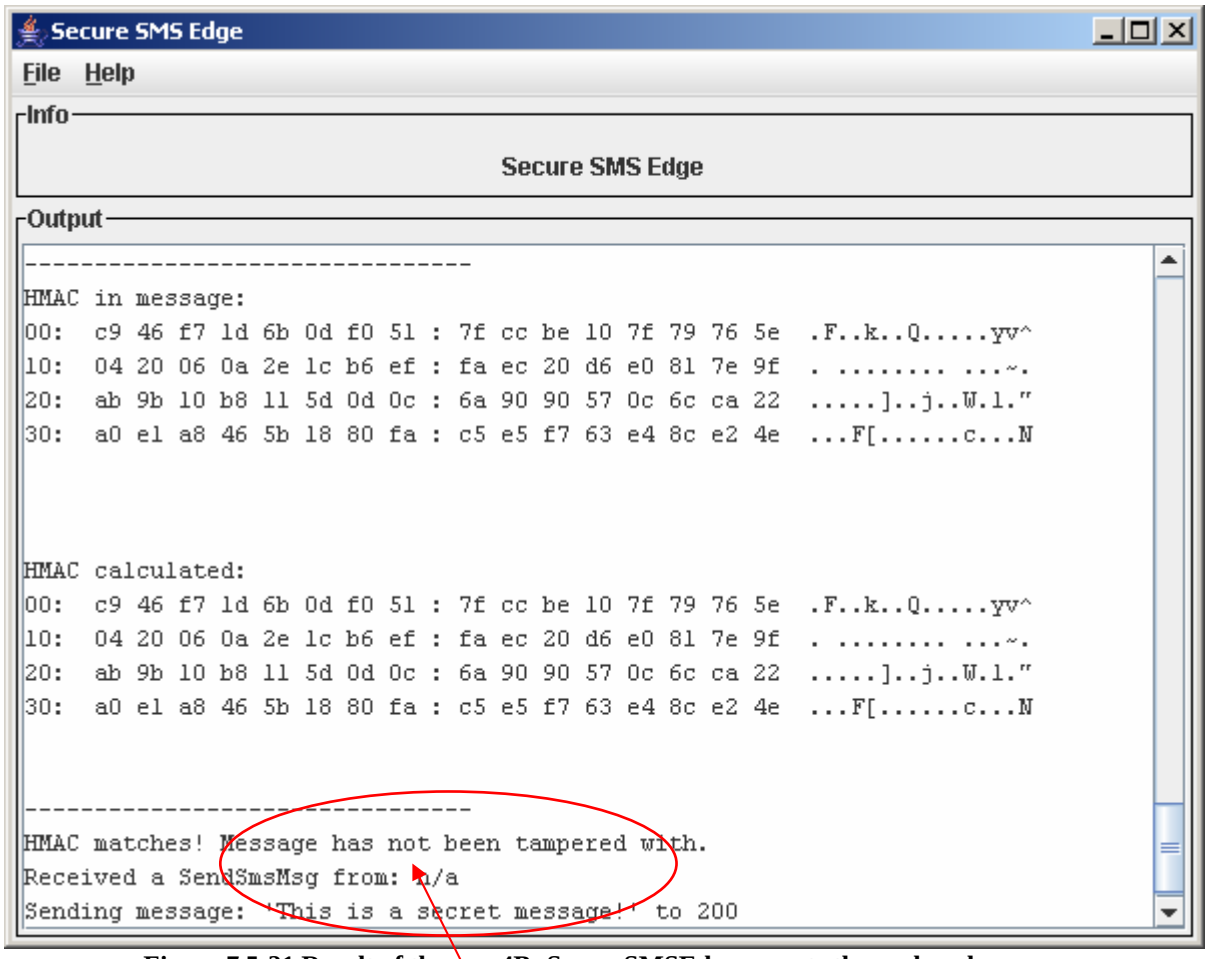

Figure 7.5-21 Result of the run 4B: SecureSMSEdge accepts the replayed message

The message is still accepted by the SecureSMSEdge even though it has been replayed.

Figure 7.5-22 Shows that in the NRG simulator the mobile phone with number “200” has received two SMS messages although Alice only sent one. This is because the replay attack by Eve is successful.

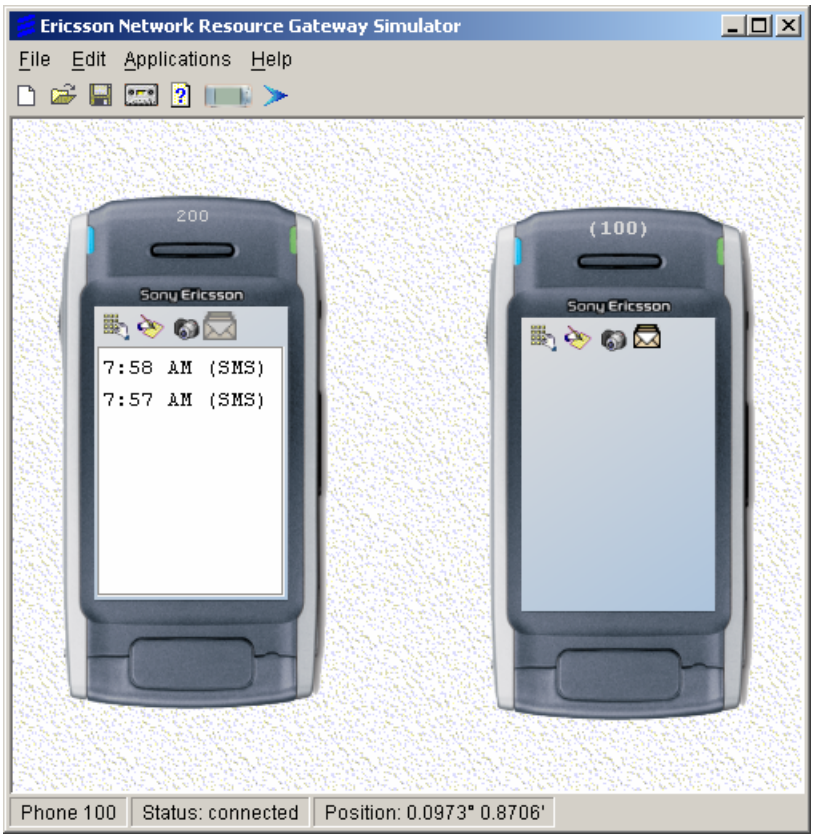

Figure 7.5-22 Result of test run 4B: The NRG simulator showing two messages

7.6 Results

The result from the test runs in the previous section can also be divided into two categories: without using the security mechanisms and using the security mechanisms.

Without using the security mechanisms:

- It is possible to see the message contents
- It is possible to tamper with the message contents
- It is possible to spoof the sender address in the actor message
- It is possible to replay the messages

Using the security mechanisms:

- It is not possible to see the contents of the message because it is encrypted
- It is not possible to tamper with the messages because of the HMAC integrity check
- It is not possible to spoof the sender identity (message origin) although it is possible to relay (forward) the messages. The test showed that a forged certificate signed by an unknown CA root certificate was rejected.
- It is still possible for a “man-in-the-middle” attacker to replay the messages because there are no timestamps in the protocol

The following paragraphs describe what are not implemented or tested in the test case:

- The protocol implementation does not support concurrency. Only one active “session” is possible at any time. The result is that an attacker can halt the protocol or cause the “client” and the “server” to end up in a deadlock.
- The authorisation or access control mechanisms are not implemented. It is possible for an attacker to gain access to all services in ServiceFrame since the access control is not yet implemented. If a service requires authentication and an establishment of a secure connection, agents that failes to get authenticated could be denied access, but this would still not be a proper access control mechanism.
- No security mechanisms are implemented for achieving non-repudiation. It is only possible to authenticate the parties, but it is not possible to hold the agents accountable for their messages. The security mechanisms cannot yet be used for charging telecom services. The ServiceFrame framework provides logging functionality as shown in the results.
- There are no loadbalancing or redundant services supported in the security mechanisms. The security mechanisms have not implemented any detection or prevention of DDoS and attacks on availability. The implementation is, as mentioned earlier, rather fragile to multiple concurrent sessions and an attacker can create a deadlock.

The fFigure 7.6-1 shows how the secret message is received as an SMS message at the mobile phone using the Ericsson NRG simulator. The message is only encrypted over the Internet. Security weaknesses of the telecom networks are still relevant.

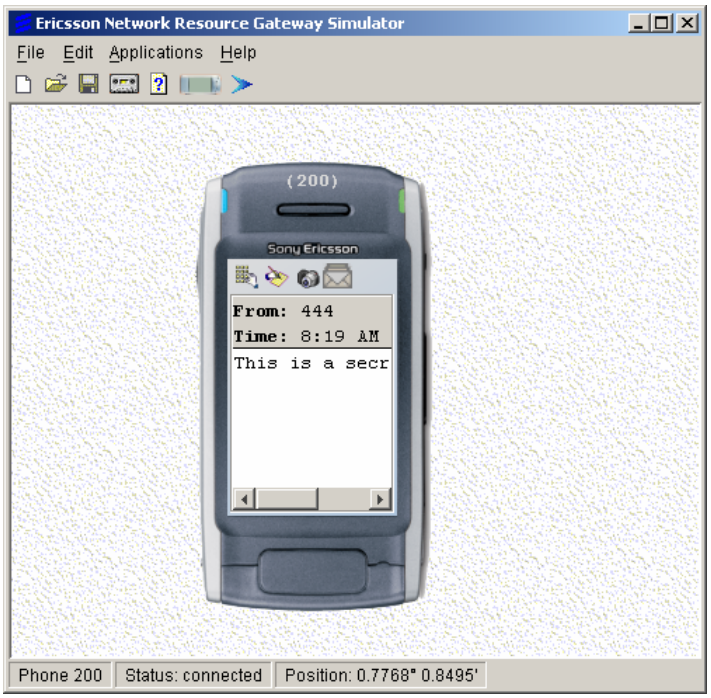

Figure 7.6-1 The NRG simulator receiving a message

7.7 Summary

In this chapter we have described how we implemented and conducted the test case. The test results will be discussed and analysed in the next chapter, “Discussion and conclusion”. The discussion will also consider the proposed security mechanisms and implementation in comparision with the research questions from Chapter 1.

8 Discussion and conclusion

8.1 Introduction

In this chapter we will discuss the proposed security mechanisms in relation to the implementation, the test case and the decisions we made. We will discuss the three research questions and other related issues. We summarise in each section, explain further work and draw a final conclusion.

8.2 Authentication

The first research question was: *How to authenticate Agents in ServiceFrame?* Authentication of agent is the key factor for security in ServiceFrame, because security mechanisms are useless without first authenticating. ActorFrame does not provide any reliable security because this has not been an issue from the beginning. As earlier mentioned there is a security problem in ActorFrame in which two actors could have the same actor address but on different domains. It is possible to spoof actor addresses. ActorFrame does not currently provide any reliable authentication mechanisms.

We proposed a solution where Agents are authenticated in a key exchange protocol. We have chosen to use certificates in the key exchange protocol. We suggest that this is adequate to authenticate the actors. Using certificates as part of the authentication process is one of the most used options. In reality there is no alternative to using certificates in the authentication process. Our proposed security mechanisms should not be dependent on a trusted third party or a central server as mentioned in Chapter 4.5 . A common example of a security protocol is the Secure Socket Layer (SSL). In SSL the certificates are used in the authentication process mostly for authenticating the server, but could also be used for authenticating the client. We use certificates in our protocol for authentication. The main advantage of using certificates is that certificates are hard to forge and easy to verify. This of course assumes that the signing algorithm used is not easily broken and the implementation has to be without fault. We chose to use OpenSSL as a Certificate Authority (CA) when creating the root certificates. The certificate chain of the actor certificate is signed by OpenSSL using the ServiceFrame root certificate. Since we do not use a trusted third party to sign and issue certificates as earlier mentioned, we do not maintain a Certificate Revocation List (CRL). All the root certificates are hard coded in the Actors in their Java keystore. This is much the same as is done in Web browsers using SSL. Java Keytool creates certificates in the X509 version 1 format, although it can import version 3. In OpenSSL it is possible to create certificates in version 3. It is not necessary to use the extensions provided in version 3. The way we use the information provided in the certificate, version 1 is sufficient.

As explained in Chapter 4 we chose to sign the certificates using SHA-2 (SHA-512). We have found that the SHA-1 algorithm is broken, as explained in Chapter 2. We still feel that this algorithm is sufficient because we use the SHA-2 variant with 512 bits. Another reason for choosing SHA-512 is that this is provided in the standard Sun Java Security provider. This is essential for our purpose. Java is the programming language used in ActorFrame. In Java it is up to the security providers to define which cryptographic algorithms are supported. This means that new and better signing algorithms can be added later. We have to create a solution that works using standard Sun Java security providers because the security mechanisms must work on a mobile platform. In order to make the security mechanisms work on most mobiles it should not be necessary to install extension libraries. It is possible to add new providers, should the mobiles support them. Currently J2ME does not support all the Sun Java security

providers. Standard Java running on a computer does not have these restrictions concerning security providers. Once the mobiles become more advanced there would not be these kinds of restrictions for the mobiles.

We have chosen to base our protocol on the Needham-Schroeder-Lowe public key protocol. As far as we are aware, no real weaknesses have been discovered in the Needham-Schroeder-Lowe protocol. We do know that if an already existing protocol is changed, it is possible to introduce new weaknesses. Our protocol is based upon challenge and response with nonce. In our protocol the authentication uses nonce, signing and encryption. We argue that this is sufficient for authentication of both parties in the communication. On the other hand we have not conducted a mathematical analysis of the protocol. In order to make sure that the protocol does not contain any weaknesses, an analysis should be performed. A mathematical analysis of the protocol using formal methods is beyond the scope of our thesis definition. This would have been a complete study of its own and we do not have sufficient time for this.

In our work we have chosen not to authenticate the end user. It would have been ideal to do this if the security should be complete and the system used in a commercial product. Therefore in our test case we have not implemented authentication of the end user because this is beyond the scope of our thesis definition. Authentication of the end user can be performed using passwords or certificates. In the test case we executed a test where an agent with a valid certificate was authenticated using the protocol implementation. We also conducted a test with an agent without a valid certificate signed with our root certificate. The test showed that our security mechanism handled incorrect certificates. On the other hand this depends on how good the CA is at verifying the identity of the certificate owner when signing certificates, as mentioned in Chapter 2.9 . If a certificate with a false identity is accepted and signed by a CA, our prototype will accept this certificate. This outcome is relatively hypothetical because it is hard to forge the digital certificates.

From the discussion we draw the following conclusion: *We have proposed a solution that authenticates through the use of certificates in a security protocol. We have tested this through the execution of a test case. We tried to send an invalid certificate and this was rejected. The proposed authentication solution is a step in the direction of a more secure ServiceFrame.*

8.3 Authorisation

The second research question was: *How to authorise Agents in ServiceFrame?* Authorisation is important for securing access to a system. ServiceFrame has access to the telecommunication network through the edges. Consequently access controls of these components are an option to prevent attackers.

Kerberos is a possible solution for access control but requires a central server and that is not applicable for loosely coupled components. Kerberos is also mainly used in a local network. As an alternative we proposed a solution for authorisation and access control based on Access Control Lists. The agents maintain a list of other agents and their access right to this agent. There are some advantages of C-lists as opposed to ACLs, but C-lists are user- or subject-oriented which is not the focus. We focus instead on securing the resources in our proposed access control mechanisms. The proposed mechanisms have not yet been implemented in the prototype. As a result we have not been able to test whether or not our proposed solution is workable. RBAC is another possible solution for access control which could have been used. This would have required a complete master thesis study in itself and therefore we have only

proposed some elementary access control mechanisms. The use of ACLs or C-lists are fundamental mechanisms that can be used in RBAC as well. RBAC could have been used for securing access control for the whole system, whereas we, in this thesis, only concentrate on securing the agents.

From the discussion we draw the following conclusion: *Our proposed solution for access control offers authorisation of agents, although it has not been fully implemented yet. The access control mechanisms are currently simple but essential and could easily be extended using RBAC.*

8.4 Integrity and Confidentiality

The third research question was *How to achieve integrity and confidentiality on messages in ServiceFrame?* Integrity and confidentiality are important security mechanisms to prevent tampering with data, "man-in-the-middle" attacks and replay attacks. Confidentiality also prevents unwanted disclosure of data.

Integrity

Integrity of messages is a key security issue. In the key exchange phase of the protocol we use signing of the messages with the RSA private key. In this way the messages are given an origin integrity as well as confidentiality. Another solution could have been to just create a message digest of the message and then sign that with the RSA private key. Data integrity is closely connected to authentication of the user or the agent. We discussed that in the authentication segment.

We have not found any other relevant solution that achieves integrity without using certificates or a HMAC of the messages. On the other hand we could have used a trusted third party server instead of using certificates or HMAC, but this is not possible in our situation. When we send messages after the key exchange we use a HMAC of the message to achieve integrity. In the SSL protocol they have solved this in a different way. They create a hash using all the previous sent messages, but we feel that this is not necessary and that the SSL protocol is too complex and features unnecessary elements. In order to prevent replay attack it could have been possible to put time stamps in the messages, but this demands synchronisation. Due to limited time we have not implemented time stamps of the messages.

Under the execution of the Test Case we tested if it was possible to achieve message integrity. We first conducted a test without using the integrity check. This test showed that the integrity was not maintained. In the next test run we used the security protocol to test if the integrity was achieved. We altered some of the bits in the message and demonstrated that the HMAC integrity check worked. The test showed that the attacker must be aware of which bits to tamper with. If the attacker is not aware of which bits to tamper with, it is likely that it would result in a null pointer exception in Java at the receiver end. This is because of the de-serialisation that is performed on the messages when reconstructing the message from a decrypted byte stream. Another test showed that if the message was long and an attacker knew roughly where the message data was encrypted, he could alter this only. This would not result in a null pointer exception but the message failed the HMAC integrity check. We have not had enough time to analyse the security of the HMAC implementation.

For this section we draw the following conclusion: *Currently there is no integrity on messages in ActorFrame. In the security protocol we use mechanisms for integrity on messages. We use an HMAC of the message encrypted together with the message to provide*

integrity. In our test case we demonstrated that the implementation in our prototype worked. We therefore believe that we have achieved integrity on messages in ActorFrame.

Confidentiality

It is essential that sensitive information is encrypted to avoid disclosure, this is confidentiality. In the protocol we provide confidentiality using a public/private key encryption in the authentication process and in the key exchange. It also provides confidentiality after the key exchange through the use of encryption with the session key.

Secure Socket Layer (SSL) is a widely used alternative for achieving confidentiality. SSL is used in NTT DoCoMo i-mode and to secure communication to web servers. For use in ActorFrame the SSL protocol resides on a too low layer. In ActorFrame it is essential to have the security on agent level because services are vulnerable to attacks. The security should be implemented on agent level because we do not want agents to be able to eavesdrop or modify messages. With the SSL solution the security would have been on a lower layer and it would not secure the agents. Our protocol shares similarities with SSL in that we use private and public keys in the key exchange phase and use a session key. In contrast to SSL our protocol achieved point to point security between agents. This would not have been possible with SSL because it operates on the socket layer.

In our protocol we have used the RSA and AES cryptographic algorithms, because we mean that this gives the best known security at the moment. When we are speaking of the best security we mean the best security provided in the standard Sun Java security provider. There are restrictions on cryptographic algorithm in the standard Java Development Kit (JDK). In standard Java there is a restriction on the key size of AES and RSA, which limits the AES key size to 128 bits and the RSA key size to 512 bits. This means that it is not possible to have unlimited strength cryptography without using a special security policy file in the JDK. We use the AES encryption algorithm because this is the new NIST standard. Before the algorithm used for AES was chosen by NIST the finalists in the election were thoroughly analysed. AES is a block cipher and has modes that can achieve better security. We chose to AES with CBC since, in our opinion, this gives the best encryption. AES can also use the PKCS #5 padding standard to fill blocks before encrypting. A theoretical weakness has been discovered and two side channel attacks against AES, but it is still our opinion that AES is the best algorithm supported in standard Java. In our protocol implementation, AES is used as session key. Normally a session is short and thereby only leaves the session key valid for a short time. In ActorFrame it is possible that the sessions can be long. There should have been a mechanism for handling long sessions. A new session key should have been created if the session lasts more than a few minutes or if the amount of data transferred is too large. This is similar to what is implemented in the SSL protocol. We have not had time to implement this in our prototype, but this could have been done to make the prototype optimal.

RSA is chosen as the asymmetric cipher in the key exchange protocol because throughout RSA's nearly 30 years of existence it has withstood all attacks if used properly. There are some weaknesses in RSA, for example if one chose a small n, as mentioned earlier in Chapter 2 . We have not found a relevant alternative to RSA although Elliptic Curve Cryptography could be an option. Elliptic Curve Cryptography could be better suited for small devices such as mobile phones. Elliptic Curve Cryptography has only been focused on recently and therefore the security still needs further research. In RSA the OAEP padding scheme can be used to secure the data further. We became aware of the OAEP padding option at a late stage in our implementation and therefore we did not have time to implement the OAEP padding in

the prototype. Asymmetric ciphers are considered much more secure than symmetric ciphers but they require more processing power. Therefore we have only used RSA for the key exchange and the authentication phase. We use the AES session key encryption for the rest of the messages.

We performed a test where the messages are sent without using the proposed security mechanisms in ServiceFrame. The test showed that a "man-in-the-middle" attacker could see the message because it is not encrypted. In the next test we sent the message using the security mechanisms in the prototype. Under the execution of the test we demonstrated that the messages actually were encrypted when watching a print of the messages. On the other hand this does not demonstrate how good the encryption is, but only demonstrates that the encryption worked. Since we use CBC mode we have an initialisation vector (IV). We first thought of using a common IV for all the messages, but instead chose to use a new IV for each message. We also considered encrypting the IV, but this would not have achieved any further security because the IV is meant to be publicly known.

From the discussion we draw the following conclusion: *Confidentiality in ActorFrame could have been done using SSL, but this would not achieve confidentiality on agent level. Our solution achieves confidentiality between agents, which gives end to end security. This cannot be performed in ActorFrame using SSL. In our opinion, the combination of RSA and AES in our protocol contributes to the best encryption in standard Java. In our test case we demonstrated that the encryption actually worked and achieved confidentiality on messages between agents.*

8.5 Other implementation issues

The connection to the Parlay gateway is an important part of ServiceFrame because of the NRG edges. Parlay/OSA and Parlay X have some weaknesses concerning security as mentioned in Chapter 3. We will address some alternative solutions to where the security mechanisms should be placed, and present our alternative. We will also give a short description of attacks that may succeed in our protocol.

The Trust and Security Management (TSM) protocol used in Parlay contains weaknesses in the application server, as described in Chapter 3. Parlay/OSA has access control, but it has some flaws. It is possible to steal a secret object reference in Parlay/OSA. Parlay X uses Web Services and the security provided by WSS. WSS has some flaws mentioned in Chapter 3, such as "rewrite attack" and the password can be faked. The problems can be avoided by placing the security at a higher level. Parlay X has also weaknesses in SAML mentioned in Chapter 3. Parlay X is layered on a higher level than Parlay/OSA but still operates on a lower layer than ServiceFrame.

Considering the security flaws in Parlay, we propose a solution that can be used. Our solution can be used so that third party developers do not need to authenticate directly to the Parlay application server, when using telecommunication services. As an alternative they can develop services using ServiceFrame and our authentication mechanisms. In this way the service providers do not need to worry about weaknesses in the Parlay security because security is handled at a higher level. On the other hand this would require the security mechanisms in ServiceFrame to be much better than those already provided and existing in Parlay. We hope that our solution is a step in the right direction towards a secure ServiceFrame. Our security mechanisms are incorporated as a component in ServiceFrame.

It is generally accepted by security experts that security protocols and algorithm should be open for the public to ensure that their security can be analysed. This is the complete opposite of what is called security by obscurity. Some claim that it is a security issue to have all the algorithms publicly known. A group of systems where the algorithms and mechanisms are kept secret is defence systems. In NTT DoCoMo i-mode they use a proprietary network and keep the underlying security secret. As earlier mentioned in Chapter 2 it is not possible to analyse the security of these systems. We have chosen to use publicly known cryptographic algorithms that have been analysed by many security experts.

In standard Java it would have been possible to use the SignedObject class for automatically providing integrity check through certificates. We could not use this approach because this solution is not available on the mobile platform. J2ME on the mobile platform does not support serialisation yet and therefore cannot use the SignedObject class for signing. As a result we have had to implement the integrity check manually, which is not optimal. If the mobile platform in the future can use standard Java and have support serialisation, it would be easier to implement good solutions for the mobile platform. We have not focussed on making a solution for the mobile platform, but only kept in mind that the solutions should be adaptable to the J2ME platform. We have not evaluated how good the pseudo-random generator in Java is to generate random nonces. This in itself is a whole field of research and we have therefore only based our solution on what is provided in standard Java.

We considered at an early stage putting the security in ActorRouter, but then the security would only apply to the domain level. This would be equivalent to the security provided by VPN and would not have achieved point to point security between agents. The reason we considered this options is that it is between actor domains that the network traffic passes through unsecured networks, and therefore it is more subject to attacks. It would also provide the same certificates and security for all the agents within the domain. It would not provide individual security and a rogue agent inside the domain could be disastrous. It would not prevent attacks from inside. The rogue agent could also attack other domains. This is possible because the whole domain is authenticated and not the individual agents.

An alternative solution was brought up early in our thesis work. The solution consisted of placing the security in the actor ports. After a thorough investigation we discovered that some of the actor messages bypassed the port, which lead to a security breach. In order to prevent this we would have to rewrite some of the code for the actor port. We decided not to use this solution because we did not want to change already existing functionality. The connection security would have to be manually specified in a XML file. If the establishment of the secure connection had failed, the user of the system would not have recognised the failure. It would have been a hard to find the error because it would require looking at the log files. The connection would also have to be reconfigured manually in the XML file.

We wanted to test the performance with and without our security mechanism. We did not have the time to perform this kind of testing. We have not been able to test how scaleable the mechanisms are in a distributed system. Our implementation has only been tested on the J2SE platform and we have not tested the J2EE and the J2ME platforms. We realise that the persistency of the data in the ActorContext in the J2EE platform is a bottleneck, but have not fulfilled this task.

We chose to use the Needham Schroeder Lowe public key protocol, but we are aware that Diffie Hellman with signatures is an alternative, but early on we were forced to make a

decision. The protocol was going to be implemented in the security mechanisms for ServiceFrame and is only a part of the prototype we have implemented. We decided to use Needham Schroeder Lowe after considering both options. We have modified the Needham Schroeder Lowe protocol to suit ServiceFrame better. The main differences between our protocol and Needham Schroeder Lowe are that our protocol uses certificates, session key and signing.

We considered using a procedure call in the protocol, but we discarded this option since it was unnecessarily complicated and it would not have been implemented in time. Instead we have implemented a simpler solution which leaves much of the job to the programmer. Some of the code required to set up a secure connection has to be manually added. We had hoped for a solution that could make the establishment of a secure connection seem transparent to the programmer. The remote procedure call might have solved this problem. Since this has not been implemented the programmer must explicitly state that the message should be sent securely. In our solution the programmer would also have to initiate the secure connection establishment himself, on the client side.

We have chosen to let the security reside on the agent level instead of the in actors, because we found this more appropriate. Agents could have their own role state machines. These roles have access to the agent's internal structure and data. The roles are also tightly connected to the agent. As a result, our security protocol can easily be connected to the agent and inform the agent when the protocol is established. This means that the establish security role informs the agent when a new secure connection is established and what kind of session key that is used for this session. This could not have been done using actors. Actors would have to have another internal state machine in addition to the default one. On the other hand letting the security reside on the actor level could have been better. All actors would then have the ability to communicate securely. Since all components such as agents inherit the functionality from actor, it would then provide security for all components. Currently in our prototype it is only the components that inherit from agent (or SecureAgent) that have the possibility of communicating securely. This is especially if some of the actors are critical components. Another solution would be to let these inherit from SecureAgent instead, but this would require modification of many actors in ActorFrame. In the present prototype implementation the protocol is implemented in a composite state. Should there be improvement of the prototype at a later stage, then this component would not have to change. On the other hand if the security protocol needs to be changed only the composite state would have to be modified.

"Replay attacks" can be preformed in our protocol, and this is a weak point in our protocol. If a replay attack is launched at a service which charges for each request from the system, this can be a serious threat. Prevention of "DOS/DDOS" attacks is not implemented in our system. The whole point of "DOS/DDOS attack" is to halt the system or services. In our implementation Eve may interfere with the communication between Alice and Bob. She can halt the communication and create a deadlock. The implementation does not support multiple concurrent sessions, i.e. only one agent can be handled at any time. This is due to the limitations of the implementation and is not because of the protocol. "Elevation of privilege" is an attack performed to get access to a higher access level. In the proposed solution this is not possible since there is only one access level. "Non-repudiation" is done to prove that an event actually occurred. The protocol does not support "non-repudiation".

From the discussion we draw the following conclusion: *We have considered the security of Parlay because ServiceFrame is tightly connected to the Parlay gateway through the NRG*

edges. Our proposed security mechanisms may contribute to an increased use of ServiceFrame since the security issues of Parlay no longer need to be addressed by third party developers. We considered many different solutions for the security mechanisms, but we ended up implementing the security on the agent level. We prefer to use open standards and publicly known security protocols and cryptographic algorithms. We have placed the security protocol inside a composite state and this means that the protocol can be reused in a different context. It is also possible to replace the protocol with another equivalent security protocol if that is necessary.

8.6 Further Work

We have only used our own Certificate Authority when testing the prototype. In a commercial system a worldwide trusted CA such as VeriSign should be used. This is done by applying for a signing of the certificate request. The CA would thoroughly verify the identity of the applicant before signing the certificate. CRL is an issue that needs to be implemented.

In the future our protocol needs a mathematical analysis using formal methods.

An important aspect of security is to provide security for the end user. The security of the end users needs to be addressed if ServiceFrame is going to be used in a commercial system.

The access control mechanisms can be improved by using mechanisms such as Role Based Access Control (RBAC).

An improvement of our implementation could involve modifying the XML schema to include the location of the keystore and password. The XML file should also be encrypted and signed to avoid disclosure of the password.

Our protocol could be extended so that it handles re-establishment of a session with a new session key. This would prevent the use of the same session key for long period of time. A time stamp could also be placed in the message to prevent replay attacks.

In order for ServiceFrame to be used by third party developers, performance testing of the security mechanisms must be conducted. How the system performed with or without the security mechanisms should be analysed. The scalability of the system should also be evaluated.

Availability and accountability are also two important security issues. This security issues have been beyond the scope of this thesis, but need to be addressed before ServiceFrame can be used by third party developers.

In this thesis work the security mechanisms have only been implemented and tested using the J2SE platform. Adapting the implementation to the J2ME platform is essential for the use of the security mechanisms on mobile phones.

8.7 Conclusion

In this thesis the need for security in ActorFrame and ServiceFrame has been addressed. We have focused on security mechanisms that authenticate and authorise agents, as well as security mechanisms that achieve integrity and confidentiality on messages. We have proposed a solution which uses these mechanisms to achieve security. We have developed a prototype implementing authentication, integrity and confidentiality. We have also developed a test case which demonstrates some of the security mechanisms in the prototype. In our discussion we have discussed the security related to the three research questions defined in the introduction chapter. We evaluated the proposed security mechanisms in relation to the test case and the prototype according to the three research questions.

The main contribution of this thesis is the introduction of security mechanisms in ActorFrame. Security mechanisms for authentication of agents have been proposed and implemented. Security mechanisms for authorisation of agents have been proposed but not yet implemented. Confidentiality and integrity have also been proposed and implemented. An implementation and execution of a test case has demonstrated that the security mechanisms for authentication, integrity and confidentiality seem to work. We hope that our work can be a fundament for further work, and that the security mechanisms can contribute to securing services which access Parlay.

The main conclusion of this thesis is that we have introduced security mechanisms in ServiceFrame. Our security mechanisms achieve point to point security between two agents. The protocol proposed is a key element in our security mechanisms and is an extension of the Needham-Schroeder-Lowe public key protocol. The security mechanisms will assist developers of ServiceFrame when creating secure services.

The study has also shown that some modification will be required before the security mechanisms will work on the mobile platform. The J2ME platform currently does not support Java cryptography or serialisation of objects. These restrictions will probably not be a problem on future mobile phones. We assume that security on mobiles phones will be a key factor in the future. Service creation framework like ServiceFrame is future-oriented and needs to support security mechanisms.

Appendix

Abbreviations

1. ACL- Access Control List
2. ACM- Access Control Matrix
3. AES- Advanced Encryption Standard
4. API- Application Programming Interfaces
5. AS- Authentication Server
6. CA- Certificate Authority
7. CBC- Cipher Block Chaining
8. CerPath- Java Certification Path API
9. CHAP- Challenge Handshake Authentication Protocol
10. CRC- Cyclic Redundancy Check
11. CRL- Certificate Revocation List
12. DAC- Discretionary Access Control
13. DBC- Domain Boundary Controller
14. DDoS- Distributed Denial of Service
15. DES- Data Encryption Standard
16. DoS- Denial of Service
17. FAQ- Frequently Asked Questions
18. HMAC- Hash Message Authentication Code
19. IIOP- Inter-ORB Protocol
20. IN- Intelligent Networks
21. IPsec- security protocol
22. IV- Initialisation Vector
23. J2EE- Java 2 Enterprise Edition
24. J2ME- Java 2 Mobile Edition
25. JAAS- Java Authentication and Authorization Service
26. JCA- Java Cryptography Architecture
27. JCE- Java Cryptography Extension
28. JDK- Java Software Development Kit
29. JGSS- Java Generic Secure Services
30. JRE- Java Runtime Environment
31. JVM- Java Virtual Machine
32. KDC- Key Distribution Center
33. MAC- Mandatory Access Control
34. MAC- Message Authentication Code
35. MAC- message authentication code
36. MDK- Model Developing Kit
37. MIT- Massachusetts Institute of Technology
38. NIST- National Institute of Standards and Technology
39. NorARC-Norwegian Advanced Research Center
40. NRG- Network Resource Gateway
41. NRG- Network Resource Gateway
42. OAEP- Optimal Asymmetric Encryption Padding
43. OCSP- Online Certificate Status Protocol
44. or C-lists- Capability lists
45. PMSK- Pre-Master-Secret-Key
46. RBAC- Role Based Access Control

47. SAML- Security Assertion Markup Language
48. SCF- Service Capability Features
49. SDK- Software Developer's Kit
50. SOAP- Simple Object Access Protocol
51. SSL- Secure Socket Layer
52. TGS- Ticket Granting Server
53. TLS- Transport Layer Security
54. TSM- Trust and Security Management
55. UML- Unified Modeling Language
56. VPN- Virtual Private Network
57. W3C- World Wide Web Consortium
58. WSDL- Web Services Description Language
59. WSS- Web Services Security
60. XOR- exclusive OR

Bibliography

1. Pistoia Marco, Nagaratnam Nataraj, Koved Larry and Nadalin Annthony. "*Enterprise Java security : Building Secure J2EE Applications"*, Addison-Wesley, 2004

2. Steel Christopher, Nagappan Ramesh and LaiCore Ray. "*Core security patterns : best practices and strategies for J2EE, Web services and identity management"*, Prentice Hall, 2005

3. *"FIPS 46-3"*, U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology, 25 October 1999.

4. Hook Davis."*Beginning cryptography with Java*",Wrox, 2005

5. Schneier Bruce. "*Applied cryptography : protocols, algorithms, and source code in C"*, John Wiley & Sons, 1996

6. Oaks Scott. "*Java security"*, O'REILLY, 2001

7. Bishop Matt."*Computer security : art and science"*, Addison-Wesly, 2003

8. Stallings William."*Cryptography and network security : principles and practices"*, Pearson Prentice Hall,2006

9. Gollman Dieter. *Computer security*, Wiley, 2005

10. Weiss Jason. "*Java cryptography extensions : practical guide for programmers"*, Elsevier, 2004

11. Diffie W. and Hellman M. E. *"New Directions in Cryptography"*, IEEE Transactions on Inforamtion Theory, Vol IT-22, 1976

12. Rivest R., Shamir A.and Adleman L. "*A Method for Obtaining Digital Singantures and Public Key Cryptosystems*" ,Communication of the ACM, February 1978.

13. Pender Tom. "*UML Bible*", Wiley, 2003.

14. "*Federal Information Processing Standards Publication 197*", National Institute of Standards and Technology, 26November 2001.

15. Daemen J. and Rijmen V. *"AES Proposal: Rijndael, Version 2"*, Proton World Int.l and Katholieke Universiteit Leuven, ESAT-COSIC , 9 March 1999, Avaiable from http://csrc.nist.gov/encryption/aes, (last accessed: 2006-05-19)

16. Helton Rich and Helton Johennie. "*Java security solutions"*, Wiley, 2002.

17. "*CNSS Policy No. 15, Fact Sheet No. 1, National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information*", CNSS, June 2003.

18. Courtois Nicolas T. and Pieprzyk Josef.“*Cryptanalysis of Block Ciphers with Overdefined Systems of Equations*”, Springer-Verlag, December 2002.

19. Daniel J. Bernstein. “*Cache-timing attacks on AES*”, The University of Illinois at Chicago , Avaiable from http://cr.yp.to/antiforgery/cachetiming-20050414.pdf, (last accessed: 2006-05-25)

20. Osvik Dag Arne, Shamir Adi and Tromer Eran. “*Cache Attacks and Countermeasures: the Case of AES*”, 20 November 2005, Avaiable from http://www.wisdom.weizmann.ac.il/~tromer/papers/cache.pdf (last accessed: 2006-05-25)

21. RSA Laboratories http://www.rsasecurity.com/rsalabs/node.asp?id=2236 (last accessed: 2006-05-25)

22. wikimedia Avaiable from http://upload.wikimedia.org/wikipedia/commons/thumb/d/d5/SHA-1.png/300px-SHA-1.png (last accessed: 2006-05-25)

23. wikimedia Avaiable from http://upload.wikimedia.org/wikipedia/en/d/d3/Cbc_encryption.png (last accessed: 2006-05-25)

24. wikimedia Avaiable from http://upload.wikimedia.org/wikipedia/en/6/66/Cbc_decryption.png (last accessed: 2006-05-25)

25. RSA Laboratories, Avaiable from http://www.rsasecurity.com/rsalabs/node.asp?id=2127 (last accessed: 2006-05-25)

26. RSA Laboratories, Avaiable from http://www.rsasecurity.com/rsalabs/node.asp?id=2129 (last accessed: 2006-05-25)

27. RSA Laboratories, Avaiable from http://www.rsasecurity.com/rsalabs/node.asp?id=2214 (last accessed: 2006-05-25)

28. Bruen Aiden A. and Forcinito Mario A. “*Cryptography, information theory, and error-correction : a handbook for the 21st century”*, Wiley, 2005.

29. Shoup Victor.“ *OAEP Reconsidered*”,Springer-Verlag, 18 September 2001.

30. Bellare M.and Rogaway P. “*Optimal asymmetric encryption*, In Advances in Cryptology|Eurocrypt '94”, 1994.

31. “*RSAES-OAEP Encryption Scheme*”, RSA Laboratories, Avaiable from ftp://ftp.rsasecurity.com/pub/rsalabs/rsa_algorithm/rsa-oaep_spec.pdf (last accessed: 2006-05-25)

32. “*FIPS PUB 186*”, U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology, 27 January 2000.

33. “*FIPS PUB 180-1*”, National Institute of Standards and Technology, 11 May1993

34. Bellare M., Canetti R.and Krawczyk H. "*The HMAC construction*", CryptoBytes, Spring 1996.

35. Stamp Mark. "*Information security: principles and practice"*, Wiley, 2006.

36. M. Bellare, R. Canetti and H. Krawczyk. "*RFC 2104-HMAC: Keyed-hashing for message authentication*", IETF, February 1997.

37. Ravi S. Sandhu, Edward J. Coyne, Hal I. Feinstein and Charles E. Youman. "*Role Based Access Control Models"*, IEEE, 1996.

38. "ITU recommendation ITU-T X.509",ITU, 1988.

39. Kohnfelder L. " *A method for certification*. Laboratory for Computer Science", Massachusetts Institute of Technology, Cambridge, May 1978.

40. "*ITU recommendation ITU-T X.509*", ITU, 2005 .

41. The Computer Language Company, Avaiable from http://www.computerlanguage.com/cahn2.html (last accessed: 2006-05-25)

42. Lampson, B. W., "*Protection*", in Proc. Fifth Princeton Symposium on Information Sciences and Systems, Princeton University, March 1971, reprinted in Operating Systems Review, 8,1, January 1974

43. Butler Lampson, Martín Abadi, Michael Burrows and Edward Wobber. "*Authentication in distributed systems: theory and practice*",ACM Transactions on Computer Systems (TOCS), November 1992

44. Microsoft TechNet, "W*hat Are Security Descriptors and Access Control Lists?*", 2003

45. Ferraiolo David F., Sandhu Ravi, Gavrila Serban, Kuhn D. Richard and Chandramouli Ramaswamy. " *Proposed NIST standard for role-based access control*", ACM Transactions on Information and System Security (TISSEC), August 2001

46. RSA Laboratories, Avaiable from, http://www.rsasecurity.com/rsalabs/node.asp?id=2093 (last accessed: 2006-05-25)

47. Encyclopædia Britannica, Available from http://www.britannica.com/ (last accessed: 2006-05-27)

48. Warinschi Bogdan. "*A Computational Analysis of the Needham-Schröeder-(Lowe) Protocol*", IEEE, 2003.

49. Backes Michael and Pfitzm Birgit. "*A Cryptographically Sound Security Proof of the Needham-Schroeder-Lowe Public-Key Protocol*", IEEE, 2004.

50. Kremer S. and Ryan M. D."*Analysing the Vulnerability of Protocols to produce known-pair and chosen-text attacks*", Elsevier, 2005, Avaiable from http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/Kremer-secco04.pdf (last accessed: 2006-05-25)

51. Millen Jonathan. "*AUTHENTICATION PROTOCOL ANALYSIS*" , Available from http://homepage.mac.com/j.millen/wadis.pdf (last accessed: 2006-05-25)

52. ElGamal T. "A *Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*", IEEE, July 1985.

53. Schnorr C. "*Efficient Signatures for Smart Cards", Journal of Cryptology*, 1991.

54. The Parlay group; http://www.parlay.org

55. Unmehopa Musa, Vemuri Kumar and Bennett Andy. "*Parlay/OSA From Standards to Reality*",Wiley, 2006.

56. "*FIPS 186-2*" U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology, 27 January 2000.

57. Minoru Etoh. "*Next generation mobile systems : 3G and beyond*", Wiley, 2005.

58. Tanenbaum Andrew S. and van Steen Maarten. "*Distributed Systems*", Prentice Hall, 2002.

59. G. Lowe."*An attack on the Needham-Schroeder public key authentication protocol*", Information processing letters, November 1995, Avaiable from http://citeseer.ist.psu.edu/correct/461242 (last accessed: 2006-05-25)

60. Needham Roger M. and Schroeder Michael D. "*Using Encryption for Authentication in Large Networks of computers*", Xerox Palo Atlo Research Center, 1978.

61. Lowe G. "*Breaking and fixing the Needham-Schröeder algorithm*". In Proc. of TACAS'96, Springer-Verlag, 1996.

62. Cremers C.J.F. and Mauw S."*Generalizing Needham-Schroeder-Lowe for Multi-Party Authentication*", Department of Mathematics and Computer Science (The Netherlands), 2006, Avaiable from http://www.win.tue.nl/~ecss/downloads/generalizing-nsl.pdf (last accessed: 2006-05-25)

63. RSA Laboratories, Avaiable from http://www.rsasecurity.com/rsalabs/node.asp?id=2248 (last accessed: 2006-05-25)

64. Diffie W. and Hellman M.E. "*New directions in cryptography*", IEEE Transactions on Information Theory 22, 1976

65. W. Diffie, P.C. van Oorschot and M.J. Wiener. "*Authentication and authenticated key exchanges*", Designs, Codes and Cryptography 2 , 1992 , Avaiable from http://www3.sympatico.ca/wienerfamily/Michael/MichaelPapers/STS.pdf (last accessed: 2006-05-25)

66. Joonsang Baek and Kwangjo Kim."*Remarks on the Unknown Key-Share Attacks*", IEICE Transactions on Communications/Electronics/Information and Systems, Decmber, 2000, Avaiable from http://citeseer.ist.psu.edu/update/504131 (last accessed: 2006-05-25)

67. Krawczyk Hugo." *The Cryptography of theIPSec and IKE Protocols*",Technion & IBM Research , 2003, Avaiable from www.ee.technion.ac.il/~hugo/sigma.ppt 2003 (last accessed: 2006-05-25).

68. Netscape. "*SSL 3.0 specification*", Available from http://wp.netscape.com/eng/ssl3 (last accessed: 2006-05-25).

69. Dierks T. and Allen C. "*RFC 2246*", The Internet Society, January 1999.

70. Steiner J. G., Neuman B. C., and Schiller J. I. "*Kerberos: An authentication service for open network systems*", In Proceedings of the Winter 1988 Usenix Conference, February 1988, Avaiable from http://citeseer.ist.psu.edu/correct/251278 (last accessed: 2006-05-25).

71. Federico D. Sacerdoti, Mason J. Katz and Phillip M . "*411 on Scalable Password Service*", Papadopoulos, July 2005, Avaiable from http://www.rocksclusters.org/rocks-doc/papers/hpdc2005/hpdc2005-411.pdf (last accessed: 2006-05-25).

72. Microsoft, Avaiable from http://www.microsoft.com/technet/security/Bulletin/MS05-042.mspx, (last accessed: 2006-05-25).

73. "*The Parlay Specification*", Parlay Group, Avaiable from http://www.parlay.org/en/index.asp (last accessed: 2006-05-25).

74. ETSI, Avaiable from http://www.etsi.org/ (last accessed: 2006-05-25).

75. "*The 3rd Generation Partnership Project*", 3GPP, Avaiable from http://www.3gpp.org/ , (last accessed: 2006-05-25).

76. "*Ericsson Network Resource Gateway SDK Documentation*" , Ericsson, Avaiable from http://www.ericsson.com/mobilityworld/sub/open/technologies/parlay/index.html, (last accessed: 2006-05-25).

77. "*Parlay X specification*" , Parlay Group,Avaiable from http://www.parlay.org, (last accessed: 2006-05-25).

78. "*Web Services*", W3C, World Wide Web Consortium, Avaiable from http://www.w3c.org/ws, last accessed: (2006-04-04).

79. "*Web Services Description Requirements*", Web Services, W3C, World Wide Web Consortium, Avaiable from http://www.w3.org/TR/ws-desc-reqs/, last accessed: (2006-05-25).

80. Geir Melby. "*IKT502 HIA Introduction Part"1*, Ericsson, HiA, 2005

81. "*RFC 1994 - PPP Challenge Handshake Authentication Protocol (CHAP)*", Avaiable from http://www.faqs.org/rfcs/rfc1994.html, (last accessed: 2006-05-25).

82. "*Strong Security Protection for OSA/Parlay Gateways and Servers*", Xtradyne, 2003

83. OMG, Available from http://www.omg.org/, (last accessed: 2006-05-25).

84. "*Strong Security Protection forOSA/Parlay Gateways and Servers*", Xtradyne, March 2004, Avaiable from http://www.xtradyne.com/documents/whitepapers/Xtradyne_OSA-Parlay_WhitePaper.pdf , (last accessed: 2006-05-25).

85. Corin R., Caprio G. Di, Etalle S., Gnesi S., Lenzini G., and Moiso C. "*Security Analysis of Parlay/OSA Framework*", Avaiable from http://www.ub.utwente.nl/webdocs/ctit/1/00000108.pdf , (last accessed: 2006-05-25).

86. Ricardo Corin, Sandro Etalle and Ari Saptawijaya.. "*Constraint-based Security Protocol Verifier(CoProVe)*" University of Twente, The Netherlands, , , Avaiable from http://wwwes.cs.utwente.nl/coprove/ , (last accessed: 2006-05-25).

87. Tim Eckardt. "*Security for Parlay-X challenges and solutions*", Xtradyne Technologies AG, November 2003, Avaiable from http://www.xtradyne.com/documents/presentations/Parlay-X_Security_Rome_2003_Xtradyne.pdf, (last accessed: 2006-05-25).

88. NTT DoCoMo, Available from www.nttdocomo.jp, (last accessed: 2006-05-25).

89. " *i-mode Technology*", NTT DoCoMo, Available from www.nttdocomo.com/technologies/present/imodetechnology/index02.html, (last accessed: 2006-05-25).

90. "*The unofficial independent imode FAQ*", Eurotechnology Japan, Available from http://www.eurotechnology.com/imode/faq-gen.html, (last accessed: 2006-05-25).

91. "*Wired versus Wireless Security: The Internet, WAP and iMode for E-Commerce*" , IBM Software Group, 2003.

92. W3C, Avaiable from http://www.w3.org/2002/ws, (last accessed: 2006-05-25).

93. SOAP, Avaiable from http://www.w3.org/xx, (last accessed: 2006-05-25).

94. SCO, Avaiable from , (last accessed: 2006-05-25).

95. Bhargavan Karthikeyan, Fournet C´edric, Gordon Andrew D. and O'Shea Greg. "*An Advisor for Web Services Security Policies*" , ACM Press, 2005.

96. Backes Michael and Groß Thomas. "*Tailoring the Dolev-Yao Abstraction to Web Services Realities - A Comprehensive Wish List-*", ACM Press, 2005.

97. "*Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0*", OASIS Standard, 15 March 2005, Avaiable from http://docs.oasis-open.org/security/saml/v2.0/saml-sec-consider-2.0-os.pdf , last accessed: (2006-05-25).

98. Sun, Avaiable from http://java.sun.com/products/jce/index-14.html , (last accessed: 2006-05-25).

99. Sun, Avaiable from http://java.sun.com/products/jce/index-122.html , (last accessed: 2006-05-25).

100. Sun, Avaiable from http://java.sun.com/j2se/1.5.0/docs/guide/security/CryptoSpec.html, (last accessed: 2006-05-25).

101. Sun, Avaiable from http://java.sun.com/j2se/1.5.0/docs/guide/security/jce/JCERefGuide.html, (last accessed: 2006-05-25).

102. Nechvatal James, Barker Elaine, Bassham Lawrence, Burr William, Dworkin Morris, Foti James and Roback Edward. "*Report on the Development of the Advanced Encryption Standard AES*", Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, Technology Administration and U.S. Department of Commerce , Avaiable from http://www.linuxsecurity.com/re source_files/cryptography/r2report.pdf , (last accessed: 2006-05-25).

103. Ericsson Network Resource Gateway (NRG), Available from: http://www.ericsson.com/mobilityworld/sub/open/technologies/parlay/index.html (Last accessed: 2006-05-27)

104. Norm Hardy. "*The Confused Deputy*", Available from: http://www.skyhunter.com/marcs/capabilityIntro/confudep.html (Last accessed: 2006-05-27)

105. Mark Miller, Ka-Ping Yee, Jonathan Shapiro, "*Capability Myths Demolished*", Available from: http://zesty.ca/capmyths/ (Last accessed 2006-05-27)

106. Kelsey J., Schneier B., Wagner D., and Hall C. "*Cryptanalytic Attacks on Pseudorandom Number Generators*", Available from http://www.schneier.com/paper-prngs.html (last accessed: 2006-03-27)

107. Booch Grady, Rumbaugh James and Jacobson Ivar. "*The unified modeling language user guide*", Addison-Wesley, 2005.

108. Rumbaugh James, Jacobson Ivar and Booch Grady. "*The unified modeling language reference manual*", Addison-Wesley, 2005.

109. Geir Melby." *IKT502 HIA ServiceFrame Part 5*", Ericsson, HiA, (2005)

110. Rolf Bræk, Knut Eilif Husa, Geir Melby. "*ServiceFrame whitepaper* ", Ericsson NorARC, 2002-04-22.

111. Knut Eilif Husa, Geir Melby. "*ActorFrame Developer's Guide"* Ericsson, NorARC., 2005.

112. UML Superstructure, Available from http://www.omg.org/technology/documents/formal/uml.htm (last accessed 2006-05-25)

113. Ericsson Mobility World, Available from http://www.ericsson.com/mobilityworld/ (last accessed 2006-05-25)

114. Ellison C. and Schneier B. "*Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure*". Computer Security Journal, v 16, n 1, 2000

115. *"ETSI ES 203 915-3 V1.1.1"*, ETSI, 2005-04, Available from http://www.parlay.org/en/specifications/ (Last accessed 2006-05-25)

116. "*Be your own Certificate Authority (CA)*", Gnot, 2005Available at: http://www.raoul.shacknet.nu/2005/11/10/be-your-own-ca (

Source Code

Source code for the prototype and the Test Case are available on accompanying CDROM.