Study of security aspects for Session Initiation Protocol

Jonas Kullenwall

LiTH-ISY-EX-3234-2002
2002-04-19

Study of security aspects for Session Initiation Protocol

Master thesis

Division of Information Theory
Department of Electrical Engineering
Linköping University

Jonas Kullenwall

Reg nr:LiTH-ISY-EX-3234-2002

Supervisors:	Anders Hellström	Ericsson Infotech AB
	Hans Hedbom	Ericsson Infotech AB
Examiner:	Viivke Fåk	LiTH

Linköping, 19 April 2002

Avdelning, Institution
Division, Department

Institutionen för Systemteknik
581 83 LINKÖPING

Datum
Date
2002-04-19

Språk
Language
Svenska / Swedish
X Engelska / English

Rapporttyp
Report category
Licentiatavhandling
X Examensarbete
C-uppsats
D-uppsats

Övrig rapport

ISBN

ISRN LITH-ISY-EX-3234-2002

Serietitel och serienummer
Title of series, numbering

ISSN

URL för elektronisk version
http://www.ep.liu.se/exjobb/isy/2002/3234/

Titel
Title

Analys av säkerheten kring Session Initiation Protocol

Study of security aspects for Session Initiation Protocol

Författare
Author

Jonas Kullenwall

Sammanfattning
Abstract

The objective with this thesis is to describe security mechanisms that are integrated or are proposed to be integrated with the Session Initiation Protocol (SIP). SIP is used for establishing, modifying, and terminating multimedia sessions over the IP network. This thesis is divided into two main parts, where the first part describes the implemented security mechanisms in SIP and the second part describes a number of proposed security mechanisms that may be implemented in SIP. At the end of the report there is a section that presents the scripts and results from different security tests that were performed on two implementations of SIP. Apart from describing different security mechanisms in the first part of this thesis, this section also contains an analysis on how possible security threats against SIP may be used to launch different attacks. The analysis also describes how these attacks may be prevented, if possible, by using the security mechanisms provided by SIP. The second part also contains an analysis section, which is focusing on finding the advantages and disadvantages of using a specific security mechanism compared to a similar security mechanism that is currently used or has been used in SIP. In the last section of this thesis I present my conclusions and a summary of the results.

Nyckelord
Keyword

SIP, security, signalling

Abstract

The objective with this thesis is to describe security mechanisms that are integrated or are proposed to be integrated with the Session Initiation Protocol (SIP). SIP is used for establishing, modifying, and terminating multimedia sessions over the IP network.

This thesis is divided into two main parts, where the first part describes the implemented security mechanisms in SIP and the second part describes a number of proposed security mechanisms that may be implemented in SIP. At the end of the report there is a section that presents the scripts and results from different security tests that were performed on two implementations of SIP.

Apart from describing different security mechanisms in the first part of this thesis, this section also contains an analysis on how possible security threats against SIP may be used to launch different attacks. The analysis also describes how these attacks may be prevented, if possible, by using the security mechanisms provided by SIP.

The second part also contains an analysis section, which is focusing on finding the advantages and disadvantages of using a specific security mechanism compared to a similar security mechanism that is currently used or has been used in SIP.

In the last section of this thesis I present my conclusions and a summary of the results.

About the thesis

This thesis was written at Ericsson Infotech AB in Karlstad during the last month of 2001 and the beginning of 2002. It was supervised by Viiveke Fåk at the Department of Electrical Engineering at Linköping University.

About the author

My name is Jonas Kullenwall and I am the author of this thesis. I have studied the Computer Science and Engineering programme during the last four and a half years. This project is the last step before I get my degree in Master of Science in Engineering. My profile during my studies has been Telematic, which includes telecommunication theory, cryptography and network security, image coding and computer networks. This thesis covers almost every one of these areas, so this thesis has been a great pleasure to write.

I also hope that the knowledge I have gained during this project may be useful in my future work.

Acknowledgements

I will like to thank my two tutors at Ericsson Infotech AB: Anders Ellström and Hans Hedbom. Hans, with his knowledge in security, has been a great help in structuring this thesis and answering my questions regarding security. Anders, with his knowledge in SIP and experience from his own thesis, has been a great help in answering general question and pushing me forward to the goal. Without them, this thesis would not have been possible.

I will also thank my supervisor Viiveke Fåk for answering my questions and giving me proposals how to improve the quality of this thesis.

Karlstad, March 2002

Table of Contents

1 Introduction

This chapter serves as an introduction to the rest of the thesis. First, the background of the project is presented. Following this is a section on the structure of the thesis, giving the reader an overview of the contents as well as a reference for the specialized terms.

1.1 Background

During recent years the telecommunication industry has made tremendous progress in their development of systems that offer more bandwidth to the end user. Today, the performance of the systems is high enough to offer multimedia sessions, which are very bandwidth demanding. A session is divided in two phases. The signaling phase, which controls the session, and the media phase, which handles the transportation of the data stream.

If the participants in the session should be able to understand each other, a set of rules is required. These rules are specified in a protocol. The most likely protocol to be used in an IP based multimedia session for the signaling phase is the Session Initiation Protocol (SIP), which is designed by the Internet Engineering Task Force (IETF). Several protocols with similar functionality exist and the H.323 protocol is the second most used protocol.

During the signaling phase, several parameters are exchanged between the end users. Some of these parameters may be sensitive to the users and should be kept secret, e.g. the location of the user and the user's name. It is also important that each user identifies himself to the other users and that unauthorized users are incapable of modifying, inserting or removing messages sent during the signaling phase. Different security mechanisms as encryption, authentication, secure hash functions and digital signatures may be implemented in the protocol to provide a secure session between the end users.

1.2 Purpose

- Identify the security mechanisms that are implemented in SIP and what type of protection they provide.
- Identify possible security threats against SIP.
- Evaluate which security mechanisms that may be added to SIP to increase the protection against possible security threats.
- Present a test specification for security tests performed on different SIP implementations.

1.3 Reading instructions

- The chapter "Background theory" on page 11 gives some background theory on cryptography, network security and signaling.
- The chapter "Security mechanisms in SIP" on page 21 presents the security mechanisms in SIP and identifies possible security threats against SIP.

- The chapter “Proposed security mechanisms in SIP” on page 39 lists several security mechanisms that have been proposed to be integrated in SIP or used in conjunction with SIP.
- The chapter “Security tests” on page 85 presents several security tests that were performed on two different implementations of SIP.
- The chapter “Conclusions” on page 103 gives the conclusions from the thesis.

1.4 Glossary

Active Attack	An attack which results in an unauthorized state change, such as the manipulation of files, or the adding of unauthorized files. [65]
AIS	Automated Information System - any equipment of an interconnected system or subsystems of equipment that is used in the automatic acquisition, storage, manipulation, control, display, transmission, or reception of data and includes software, firmware, and hardware. [65]
Authenticate	To establish the validity of a claimed user or object. [65]
Authentication	To positively verify the identity of a user, device, or other entity in a computer system, often as a prerequisite to allowing access to resources in a system. [65]
Buffer Overflow	This happens when more data is put into a buffer or holding area than the buffer can handle. This is due to a mismatch in processing rates between the producing and consuming processes. This can result in system crashes or the creation of a back door leading to system access. [65]
Block Chaining	A procedure used during symmetric block encryption that makes an output block dependent not only on the current plaintext input and key, but also on earlier input and / or output. The effect of block chaining is that two instances of the same plaintext input block will produce different ciphertext blocks, making cryptoanalysis more difficult. [1]
Block Cipher	A symmetric encryption algorithm in which a large block of plaintext bits (typical 64) is transformed as a whole into a ciphertext block of the same length. [1]

Client	A client is any network element that sends (SIP) requests and receives (SIP) responses. Clients may or may not interact directly with a human user. User agent clients and proxies are clients. [59]
Codec	Refers to audio or video compression / decompression (codec) algorithms used by an application.
Confidentiality	Assuring information will be kept secret, with access limited to appropriate persons. [65]
Cracker	One who breaks security on an AIS. [65]
Cryptanalysis	Definition 1) The analysis of a cryptographic system and / or its inputs and outputs to derive confidential variables and / or sensitive data including cleartext. Definition 2) Operations performed in converting encrypted messages to plain text without initial knowledge of the crypto-algorithm and / or key employed in the encryption. [65]
Cryptographic Checksum	An authenticator that is a cryptographic function of both the data to be authenticated and a secret key. Also referred to as a message authentication code (MAC). [1]
Cryptography	The art of science concerning the principles, means, and methods for rendering plain text unintelligible and for converting encrypted messages into intelligible form. [65]
Cryptology	The science which deals with hidden, disguised, or encrypted communications. [65]
Data Encryption Standard	Definition 1) (DES) An unclassified crypto algorithm adopted by the National Bureau of Standards for public use. Definition 2) A cryptographic algorithm for the protection of unclassified data, published in Federal Information Processing Standard (FIPS) 46. The DES, which was approved by the National Institute of Standards and Technology (NIST), is intended for public and government use. [65]
Decryption	The translation of encrypted text or data (called ciphertext) into original text or data (called plaintext). Also called deciphering. [1]
Denial of Service	Action(s) which prevent any part of an AIS from functioning in accordance with its intended purpose. [65]

Dialog	A dialog is a peer-to-peer SIP relationship between two UAs that persists for some time. A dialog is established by SIP messages, such as a 2xx response to an INVITE request. A dialog is identified by a call identifier, local tag, and a remote tag. A dialog was formerly known as a call leg in RFC 2543. [59]
Digital Signature	An authentication mechanism that enables the creator of a message to attach a code that acts as a signature. The signature guarantees the source and the integrity of the message. [1]
Encryption	The conversion of plaintext or data into unintelligible form by means of a reversible translation, based on a translation table or algorithm. Also called enciphering. [1]
Firewall	A system or combination of systems that enforces a boundary between two or more networks. Gateway that limits access between networks in accordance with local security policy. The typical firewall is an inexpensive micro-based Unix box kept clean of critical data, with many modems and public network ports on it, but just one carefully watched connection back to the rest of the cluster. [65]
Hacker	A person who enjoys exploring the details of computers and how to stretch their capabilities. A malicious or inquisitive meddler who tries to discover information by poking around. A person who enjoys learning the details of programming systems and how to stretch their capabilities, as opposed to most users who prefer to learn on the minimum necessary. [65]
Hash Function	A function that maps a variable-length data block or message into a fixed-length value called hash code. The function is designed in such way that, when protected, it provides an authenticator to the data or message. Also referred to as a message digest. [1]
Header Field	A header field is a component of the SIP message header. It consists of one or more header field values separated by comma or having the same header field name. [59]
Integrity	Assuring information will not be accidentally or maliciously altered or destroyed. [65]

IP Spoofing	An attack whereby a system attempts to illicitly impersonate another system by using IP network address. [65]
Key	A symbol or sequence of symbols (or electrical or mechanical correlates of symbols) applied to text in order to encrypt or decrypt. [65]
Network Security	Protection of networks and their services from unauthorized modification, destruction, or disclosure, and provision of assurance that the network performs its critical functions correctly and there are no harmful side-effects. Network security includes providing for data integrity. [65]
Non-Repudiation	Method by which the sender of data is provided with proof of delivery and the recipient is assured of the sender's identity, so that neither can later deny having processed the data. [65]
Nonce	An identifier or number that is only used once. [1]
Packet	A block of data sent over the network transmitting the identities of the sending and receiving stations, error-control information, and message. [65]
Passive Attack	Attack which does not result in an unauthorized state change, such as an attack that only monitors and/or records data. [65]
Plaintext	Unencrypted data. [65]
Private Key	One of the two keys used in a asymmetric encryption system. For secure communication, the private key should only be known to its creator. [1]
Promiscuous Mode	Normally an Ethernet interface reads all address information and accepts follow-on packets only destined for itself, but when the interface is in promiscuous mode, it reads all information (sniffer), regardless of its destination. [65]
Proxy, Proxy Server	An intermediary program that acts as both a server and a client for the purpose of making requests on behalf of other clients. Requests are serviced internally or by passing them on, possibly after translation, to other servers. A proxy interprets, and, if necessary, rewrites a request message before forwarding it. [3]
Pseudorandom Number Generator	A function that deterministically produces a sequence of numbers that are apparently statistically random. [1]

Public Key	One of the two keys used in a asymmetric encryption system. The public key is made public, to be used in conjunction with a corresponding private key. [1]
Redirect Server	A redirect server is a user agent server that generates 3xx responses to requests it receives, directing the client to contact an alternate set of URIs. [59]
Registrar	A registrar is a server that accepts REGISTER requests. A registrar is typically co-located with a proxy or redirect server and may offer location services. [3]
RSA Algorithm	RSA stands for Rivest-Shamir-Aldeman. A public key cryptographic algorithm that hinges on the assumption that the factoring of the product of two large primes is difficult. [65]
Security Service	A service, provided by a layer of communicating open systems, which ensures adequate security of the systems or of data transfers. [65]
Secret Key	The key used in a symmetric encryption system. Both participants must share the same key, and this key must remain secret to protect the communication. [1]
Server	A server is a network element that receives requests in order to service them and sends back responses to those requests. Examples of servers are proxies, user agent servers, redirect servers, and registrars. [59]
Session Key	A temporary encryption key used between two participants. [1]
Signaling System 7 (SS-7)	A protocol used by phone companies. Has three basic functions: Supervising, Alerting and Addressing. Supervising monitors the status of a line or circuit to see if it is busy, idle, or requesting service. Alerting indicates the arrival of an incoming call. Addressing is the transmission of routing and destination signals over the network in the form of dial tone or data pulses. [65]
Spoofing	Pretending to be someone else. The deliberate inducement of a user or a resource to take an incorrect action. Attempt to gain access to an AIS by pretending to be an authorized user. Impersonating, masquerading, and mimicking are forms of spoofing. [65]

Symmetric Encryption	A form of cryptosystem in which encryption and decryption are performed using the same key. Also known as conventional encryption. [1]
Threat	The means through which the ability or intent of a threat agent to adversely affect an automated system, facility, or operation can be manifest. A potential violation of security. [65]
Trojan Horse	An apparently useful and innocent program containing additional hidden code which allows the unauthorized collection, exploitation, falsification, or destruction of data. [65]
User Agent Client (UAC)	A user agent client is a client application that initiates the SIP request. [3]
User Agent Server (UAS)	A user agent server is a server application that contacts the user when a SIP request is received and that returns a response on behalf of the user. The response accepts, rejects or redirects the request. [3]
Vulnerability	Hardware, firmware, or software flow that leaves an AIS open for potential exploitation. A weakness in automated system security procedures, administrative controls, physical layout, internal controls, and so forth, that could be exploited by a threat to gain unauthorized access to information or disrupt critical processing. [65]

1.4.1 Acronyms

3GPP	3rd Generation Partnership Project
AH	Authentication Header
AKA	Authentication and Key Agreement
ARP	Address Resolution Protocol
AV	Authentication Vector
CA	Certificate Authority
CBC	Cipher Block Chaining
CK	Confidentiality Key
CMS	Cryptographic Message Syntax
DES	Data Encryption Standard
DoS	Denial of Service
DSA	Digital Signature Standard
EAP	Extensible Authentication Protocol

ESP	Encapsulating Security Payload
HE	Home Environment
HTTP	HyperText Transfer Protocol
IK	Integrity Key
IM CN SS	IP Multimedia Core Network SubSystem
IMSI	International Mobile Subscriber Identity
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange
IP	Internet Protocol
IPSec	IP Security
MAC	Message Authentication Code
MD5	Message Digest version 5
MIME	Multipurpose Internet Mail Extension
OS	Operating System
PGP	Pretty Good Privacy
PKIX	Public Key Infrastructure
PPP	Point to Point Protocol
PRF	PseudoRandom function
QOP	Quality Of Protection
RFC	Request For Comments
RSA	Rivest-Shamir-Adleman
S/MIME	Secure/Multipurpose Internet Mail Extension
SA	Security Association
SDP	Session Description Protocol
SHA-1	Secure Hash Algorithm version 1
SIP	Session Initiation Protocol
SMTP	Simple Mail Transport Protocol
SN	Serving Network
SS7	Signaling System No. 7
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TMSI	Temporary Mobile Subscriber Identity
UA	User Agent

UDP	User Datagram Protocol
UE	User Equipment
UMTS	Universal Mobile Telecommunication System
URI	Uniform Resource Identifier
USIM	UMTS Subscriber Identity Module

2 Background theory

This chapter introduces the reader to some background information regarding cryptography and signaling which may be needed in order to understand the rest of the chapters. Readers that are familiar with cryptography can probably skip the first five sections and continue to read about signaling and the Session Initiation Protocol (SIP [2]).

2.1 Cryptographic goals

The main goal of cryptography is to provide the following services:

1. Confidentiality
2. Authentication
3. Data integrity
4. Non-repudiation

Confidentiality is a service used to keep the information secret to everyone that is unauthorized to access it. Encryption is one method to provide confidentiality, see “Conventional encryption” on page 11 and “Encryption / decryption” on page 12.

Authentication is a service used for identification of information or entities. The identification of the information is often called *data origin authentication* or *message authentication* and identification of the enities is often called *entity authentication*.

Message authentication ensures the receiver that only an authorized party can have created the specific information. An encrypted checksum of the information is often attached to the information which proves that only an authorized entity can have created the information.

Entity authentication enables an entity to verify the identity of another entity. A common way to accomplish entity authentication is to challenge the other entity by giving it some type of information that only an authorized entity responds correctly to.

Data integrity is a service to detect unauthorized manipulation of information. Manipulation includes insertion, deletion and substitution. Secure hash functions may be used to provide data integrity, see “Hash functions” on page 14.

Non-repudiation is a service which prevents an entity from denying previous commitments or actions. Digital signatures is one method of providing Non-repudiation, see “Digital signature” on page 13.

2.2 Conventional encryption

Conventional encryption, also referred to as symmetric encryption or single-key encryption, is the most used encryption technique today. Conventional encryption uses a secret key that only the sender and receiver share. The key and the plaintext are the parameters for the encryption algorithm that produce

the ciphertext. And the key and the ciphertext are the parameters for the decryption algorithm.

There exist several classical encryption algorithms that are easy to cryptoanalyze, i.e. a third party finds the secret key or the plaintext. One of these is the Caesar Chiper [1] which is based on a substitution technique, i.e. each character in the plaintext is mapped to another character by using tables or some mathematical function. The algorithm assigns a number to every character and produces the chiphertext by adding the numerical key to every character. The result is always modulo the number of characters in the alphabet. The number of possible keys is equal to the number of characters in the alphabet and therefore easy to cryptoanalyze with brute-force.

Fortunately there exist modern encryption techniques that are very difficult to cryptoanalyze due to their complex structure. The most famous and most used algorithms are different variants of the Data Encryption Standard (DES [7]). DES and other modern encryption algorithms are based on the Feistel Cipher [8] which is characteristic by its rounds. The first round takes the plaintext and a subkey as parameters and the output is passed to the next round. The result from the last round is the actual ciphertext. This mean that each round can be seen as a independent cipher. The major differences between different modern symmetric encryption techniques is the structure of these independent chipers, i.e. they use different key lengths and bit operations. They also differ in how many rounds they use.

For more information about modern encryption techniques, see [1].

2.3 Public key cryptography

The discovery of public key cryptography has been a revolution in cryptography. Public key algorithms are based on mathematical functions rather than on different types of bit transformations. Another big difference is that public key cryptography is asymmetric, i.e. uses two separate keys instead of one. The keys are referred to as the public key and the private key. The private key is kept secret while the public key is distributed to different individuals that will communicate with the owner of the private key. Encryption / decryption is just one of three different categories that can be used with public key cryptography, these are:

- Encryption / decryption
- Digital signature
- Key exchange

2.3.1 Encryption/decryption

There exist several public key encryption algorithms but the most known is the Rivest-Shamir-Adleman (RSA [9]) algorithm. Another algorithm is Elliptic Curves encryption / decryption which is more complicated but can offer equal security with smaller keys than RSA.

Bellow is the mathematical description of the RSA algorithm:

Key Generation

Select p, q	p and q both primes
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$gdc(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d = e^{-1} \mod \phi(n)$
Public key	$KU = \{e, n\}$
	$KR = \{d, n\}$

Encryption

Plaintext	$M < n$
Ciphertext	$C = M^e \mod n$

Decryption

Ciphertext	C
Plaintext	$M = C^d \mod n$

Figure 1 The RSA algorithm [1].

One way cryptoanalyze the algorithm is to factor *n* into its two prime factors. That can easily be done with small *n* but for large *n* the degree of success will fall dramatically.

2.3.2 Digital signature

To sign a plaintext the sender encrypts it with its own private key and the receiver decrypts the ciphertext with the senders public key. If the result from the decryption is readable, the receiver can be sure that it originates from the sender because the sender its the only one that knows its private key. This way of making digital signatures is not effective because the whole message is encrypted which requires a lot of computational resources. A better solution is to map the message to a smaller, fixed sized, value and encrypt it with the senders private key. The function that maps the plaintext to a fixed sized value is called a strong one-way hash function.

2.3.3 Key exchange

One of the problems with public key cryptography is the distribution of public keys in a secure way. The actual distribution is quite easy to accomplish, but it is much harder to convince a user of the identity of the public key owner. There is no logical connection between a public key and the identity of the owner. For example, if a user C is able to deceive another user A to believe that it has received a public key from user B, then user C may be able to read encrypted messages from user A to user B. User C may also forge user B's digital signature and user A will believe that the source of signed messages is user B.

There exist several methods that solves the key-exchange problem by using some sort of trusted third party that the participants can ask for public keys. Methods that use so-called public key certificates are probably the most popular ones. A user, who requests a public key certificate, sends its public key to a Certificate Authority (CA), which creates a public key certificate for the particular public key and then it signs the public key certificate with its private key. The user may then send its public key certificate to other users and if they trust the CA, then they know that the public key in the public key certificate is valid. More detailed descriptions of public key certificates and other methods for distributing public keys can be found in [1].

Another problem concerning public key encryption / decryption is that it is much slower than conventional encryption. Therefore public key encryption is currently confined to key management and signature applications. The encryption is made with conventional encryption using a secret key, also referred to as a session key. The session key is distributed to the participants by some key exchange algorithm. One of the most famous key exchange algorithms is the Diffie-Hellman algorithm [10]. A detailed description of the Diffie-Hellman algorithm can be found in [1].

2.4 Hash functions

A hash function, H, takes a variable-length block, x, and produces a fixed-length output. There exists many such function, but to be useful in security application it must have special properties:

- $H(x)$ is relative easy to compute.
- For any given output h, it is computationally infeasible to find x such that $H(x) = h$.
- For any given block x, it is computationally infeasible to find y, $y \neq x$, with $H(x) = H(y)$
- It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$.

Hash functions that have all of these properties are often called strong one-way hash functions. Hash functions that do not have the last property are called weak one-way hash functions.

The most used hash functions today are probably Message Digest, version 5, (MD5 [11]) and Secure Hash Algorithm, version 1, (SHA-1 [12]). Detailed descriptions about these can be found in [1].

2.5 Network security

Under recent year there has been a strong focus on information technology and specially on Internet. Almost every modern company is connected to Internet and it also has its own local network that is used by the employees. Almost every task performed by the employees involves some use of the Internet or the local network, e.g. for documentation and e-mail distribution. Therefore the need of stable hardware and security is essential for a modern company.

Regarding the security topic, the following types of attacks on networks have been identified by [1]:

1. **Disclosure:** Release of message contents to any person or process not possessing the appropriate cryptographic key.
2. **Traffic analysis**: Discovery of pattern of traffic between parties. For online communication a person may measure the time it takes to apply a specific algorithm to the message or find how frequent the message exchange is. For both the offline and online case a person can discover how many messages that were sent and the length of the messages.
3. **Masquerade:** Insertion of messages into the network from a fraudulent source. A person may send a message that looks like it origins from another source or send an acknowledgement for a message that it has not received.
4. **Content modification:** Changes to the contents of a message, including insertion, deletion, transposition and modification.
5. **Sequence modification:** Any modification to a sequence of messages between parties, including insertion, deletion and reordering.
6. **Timing modification:** Delay or replay of messages. For online communication a person may replay an old message to gain a certain privilege or delay the messages to decrease the quality of the communication. For offline communication a person may replay an old special message.
7. **Repudiation:** Denial of receipt of message by destination or denial of transmission of message by source.

Security researchers have developed and designed plenty of algorithms and protocols, the purpose of which is to provide protection against one or several of these types of attacks. And there exist several good text books, within the security area, that describe the most popular and useful ones. One of these is [1], that will give the reader enough knowledge how different algorithms and protocols works.

Message authentication may be used as a measure to prevent items 3 through 6. Items 1 and 2 may be prevented by using encryption, and digital signatures

may protect against item 7. Next section will give a more detailed description about what measures that has to be made to protect against item 6.

2.5.1 Replay attacks

An attack where valid messages are maliciously or fraudlently repeated, by either the orginator or an unauthorized entity, is called a replay attack. Replay attacks may be launched to gain authrozied access to a service or to impersonate another entity.

Replay attacks are a threat to almost every system where messages are used to affect the state of the system. For example, if an attacker is able to replay a valid request, in a client-server system, then the attacker may be able to delete a specific file on the server or change a password.

It seems very easy to accomplish a successful reply attack, but one has to consider that messages may be encrypted. This eliminates the attackers ability to read the content in the replayed messages and he or she will be unable to predict the result of the replay attack. A skilful attacker may use traffic analysis to predict the content in the message.

The following examples of replay attacks are mentioned in [1]:

- **Simple replay:** The attacker simply eavesdrop a message and replays it later.
- **Repetition that can be logged:** The attacker can replay a timestamped message within a valid time interval.
- **Repetition that cannot be detected:** The attacker eavesdrop a message that does not reach the destination. When the attacker replays the message the receiver accepts is as valid and it can not detect that another authorized entity has sent it before.
- **Backward replay without modification:** The attacker replays the message back to the sender. The attack is possible if a symmetric key has been used to encrypt the message and the sender does not have the ability to distinguish if a message originates from it self.

To get a protection against replay attacks the following approaches may be used:

- **Timestamps:** The entities always attach a timestamp to their messages and the receiver only accepts messages that has a valid timestamp, i.e. the timestamp fits a certain time interval. This is sensitive because the entities need to have synchronized clocks with small divergence depending on the accuracy of the time interval. This definitely applies to offline communication where the entities can not synchronize their clocks to each other for a relative long period of time.
- **Sequence number:** The entities always include a sequence number in the message. The receiver then knows which sequence number the next message should have to be valid and messages with a smaller or a larger sequence number will be discarded. The problem with this method is that each entity has to remember the sequence number for each entity it will

communicate with. One less secure solution to this problem is to always start with a certain sequence number for each new session.

- **Challenge/response:** An entity first sends a nonce to the other entity. Then the entity requires that the subsequent message contains the correct nonce value. The problem with this method is that it is only suitable for online communication. However, this method is preferred in a client-server application where the server responds to an invalid nonce by sending a new generated nonce back to the client. The client may then resend the message with the new nonce and the server accepts it as valid. It is important to mention that the server should not use a predictable old nonce because then we are back were we began.

2.6 Signaling

Signaling refers to the exchange of information between call components required to provide and maintain service. The information is used to establish, route, monitor and terminate a call between one or several calling parties. The calling parties are identified by a unique address, that in traditional telephony is the phone number. Its important to understand that signaling is not involved in the actual transportation of data between the calling parties, but instead makes the transportation viable.

Most signaling today is done over circuit switched networks with the Signaling System No.7[1] (SS7) protocol which is a very flexible protocol with many features. But a more flexible and more cost-effective way is to make calls over an IP network, i.e. IP Telephony. The IP Telephony makes it possible to establish sessions between several parties that can exchange data adapted to different media, i.e. data, voice, pictures and video. Its also possible to modify the sessions during the call, i.e. change media type, add a caller, remove a caller etc. To be able to use all of these features the need for flexible signaling is obvious. There exist several protocols for signaling over a IP network and the most known are Session Initiation Protocol and H.323 [14].

2.7 SIP

Session Initiation Protocol, SIP, is an application layer protocol that has been designed by Internet Engineering Task Force (IETF) Multiparty Multimedia Session Control working group. It defines initiation, modification and termination of interactive, multimedia communication sessions between users.

SIP has incorporated elements from other protocols that are widely used on the Internet. It is a text-based client-server protocol with almost the same structure as the HyperText Transport Protocol (HTTP [13]) and the text-encoding schemes are borrowed from Simple Mail Transport Protocol (SMTP [16]). This makes the structure of the protocol easy to follow and understand.

1. The standard of SS7 contains many different protocols which have their own documentations. But a good overview can be found at http://www.pt.com/tutorials/ss7/index.html.

As mentioned above the SIP messages have the same structure as the messages in HTTP and contain a request line or a status line followed by at least six header fields. After the header field there may be an attached message body, the type and size of which is described by some of the header fields. As for HTTP and SMTP, SIP supports the popular Multipurpose Internet Mail Extension (MIME [18]) for describing the content in the message body. In most cases the message body consists of a Session Description Protocol (SDP [15]) message, which describes the media transfer after the signaling phase. The SDP message has a MIME subtype of *application/sdp*. Even if the SIP message body in most cases contains a SDP message it may also contain other MIME subtypes, like e.g. *text/plain* or *image/gif*.

The client-server structure is based on a client that issues a request for a service and a service server handles the request and responds with a service. A SIP-enabled end-device, SIP User Agent (UA), has both a client and server application. This is natural if you think of a telephone which both makes calls and receives calls.

All requests from a client UA contain a method in their request lines. In current version of SIP [2] there exist six different methods:

Method name	Description
INVITE	Invites UA server to a call and establishes a new connection. It can contain media capabilities.
ACK	Used to acknowledge a response generated by a received INVITE request.
BYE	Terminates the media session between two users.
REGISTER	Used for registering information about a UA clients location.
CANCEL	Terminates a non-acknowledged invitation.
OPTION	Used to get information about supported capabilities.

Table 1 Defined methods in SIP.

The responses from the UA server have the same structure as in HTTP responses. All SIP responses have a status line which contains a status code and a formal description of the response. The status codes are ordered in six different classes and each class represents a specific type of response. For a more detailed description of these classes, see [2]. Most of the header fields in the response are copies of the header fields from the request that the response corresponds to. But some header fields are specified to only be used in

responses. A response may contain a message body and the MIME subtype is in most cases the same as for the corresponding request. The UA server may include a SDP message in the message body for negotiation of the media types that should be used during the media transfer.

There exist other applications than UA clients and UA servers that are able to handle SIP messages, namely:

- Proxy servers
- Redirect servers
- Registration servers
- Gateways

Proxy servers have the task of forwarding requests from the UA client to the address specified in the SIP message. Proxy servers have also the ability to modify some parts of the header in the message, e.g. the Via header field. They can also require authentication to a request before they forward the message. Responses from the UA server are also forwarded down to the client. The responses always take the same way back as the request took, i.e. forwarded through the same proxies. If the SIP UA has been configured to always send its requests through a specific proxy, then this proxy is called the outbound proxy for the SIP UA.

A redirect server does not issue any request of its own. After receiving a request the server gathers a list of alternative locations and returns a final response.

A registration server, also called registrar, has the task of accepting REGISTER requests from a UA. The request contains information on how to reach the UA. The information is saved so other UA can ask the registrar where the UA is located. The registrar can require authentication.

Gateways are used between different types of networks or when different protocols are used between networks. Gateways between SIP and H.323 is often used as an example.

For detailed description about the SIP protocol, see [2].

3 Security mechanisms in SIP

This chapter covers the security mechanisms that are standard in the current version of SIP [5]. Because SIP still is under development, some security mechanisms that are discussed in this chapter may be excluded in later versions. There may also be new mechanisms added as standard in later versions. Some of these candidates are described in the next chapter.

The last section makes a solid analysis about the protection that these standard mechanisms give.

3.1 Authentication

Authentication ensures that a message has been created by the claimed source and that the claimed source has sent it. Authentication includes protection against modification, delay, replay and reordering, see “Cryptographic goals” on page 11.

SIP provides a stateless challenged-based mechanism for authentication. The authentication mechanism is meant to be used only in one direction. But there is an option to make mutual authentication, i.e. authentication in both directions. There is also support for integrity protection of requests and responses.

IETF SIP working group has not developed a new scheme for authentication, and uses almost the same authentication scheme as is used for HTTP [4]. One difference is the definition of the protection domain, which in the HTTP case is defined by the realm and canonical root URL. The canonical root URL does not exists in SIP, i.e. there are no files to get, put or delete like in HTTP. Therefore is the protection domain defined by the realm, userinfo, host and port part of the Request-URI, see [2] for a more detailed description of this topic.

3.1.1 HTTP Authentication

Two different authentication schemes are standard in HTTP, namely Basic and Digest authentication. The first one is very primitive and insecure because it sends the credentials for the client in plain text. The Digest authentication is much more secure because the authentication scheme uses checksums for credentials. Although it is not a perfect solution for secure authentication, it protects against most of the security flaws in the Basic authentication.

Both authentication schemes use four different header fields to accomplish the authentication. Some of these are specific for proxies and the rest are for UA authentication. The fields are (case-sensitive):

- WWW-Authenticate
- Authorization
- Proxy-Authenticate
- Proxy-Authorization

The Digest scheme can use one more header field, namely

- Authentication-Info

When a UAS receives a request for a protected domain that is not authenticated, it responds with a 401 (Unauthorized) response which contains the WWW-Authenticate header field. The header field provides information about the challenge that the client should respond to. If the client is able to respond to the challenge, it must use the Authorization header field containing credentials information about the client. If the server accepts the credentials, the client is allowed to access the protected domain.

A server may have multiple protected domains and therefore the challenge must specify which protection domain the challenge is applied to. This is accomplished by using the realm parameter in the WWW-Authenticate header field. The realm value can not be excluded for proper authentication because the clients credentials are based on it.

The Authentication-Info header can be used by the server to re-challenge the client if Digest authentication is used. It can also be used by the server to authenticate itself to the client, i.e. mutual authentication.

The Proxy-Authentication and Proxy-Authorization header fields are used when a proxy demands authentication before it forwards a message. The header fields are the same as WWW-Authenticate and Authorization, but with different names.

3.1.1.1 Basic Authentication

As mentioned in the previous section, the Basic authentication gives very poor security because it allows the client's credentials to be sent in plain text. If the server receives a request that needs authentication, it adds the WWW-Authenticate header field in the 401 (Unauthorized) response. The only parameter in the header field is the realm value, i.e.

```
WWW-Authenticate: Basic realm=”Administration”
```

When the client receives the challenge, it tries to find the credential for the specified realm and send a response to the challenge that contains username and password. The response to the challenge is sent in the Authorization header field:

```
Authorization: Basic Radix-64(username:password)
```

The radix-64 [7] function maps its parameter to printable characters. The following example shows the result of user “administrator” with password “foo”:

```
Authorization Basic YWRtaW5pc3RhdG9yOmZvbw==
```

If the credentials are correct the server authorizes the client to perform the request that caused the authentication. The client may then send the Authorization field in every request, with the same credential, to the same protection domain without receiving a challenge by the server. This reduces the number of 401 (Unauthorized) responses to the client and as a consequence increases the performance.

3.1.1.2 Digest Authentication

Digest authentication is more sophisticated than Basic authentication because it uses check sums as response to the challenge. The default checksum algorithm is MD5, but other algorithms can be used. If the server receives a request that needs authentication, it adds the WWW-Authenticate header field in the 401 (Unauthorized) response. The parameters for the WWW-Authenticate header field are described in Table 2.

Name of parameter	Description
realm	String associated with the protection domain.
domain (optional)	List of URIs that defines the protection domain.
nonce	Unique string that is created by the server for each 401 response.
opaque (optional)	A string of data specified by the server, which should be returned by the client unchanged.
stale (optional)	A flag that specifies if the previous request from the client was rejected because the nonce was stale.
algorithm (optional)	Specifies the algorithm to use for the checksum calculations.
qop-options (optional)	Specifies the quality of protection that the server supports. It can be either “auth-int” or “auth” depending on the servers capabilities.
auth-param (optional)	Future extensions.

Table 2 WWW-Authenticate header field parameters.

The nonce value is important because it specifies the current challenge. The format is implementation dependent with the restriction that it should be unique. It can be used for protection against replay-attacks if it is changed for each new request, see “Replay attacks” on page 16. That technique, however, will increase the used bandwidth.

The qop-option value is optional depending on backward compatibility with older versions of Digest authentication. It should be used because it provides mutual authentication and some message integrity protection. The mutual authentication means that the server also has to verify that it knows the username and password for the client. If the qop-options equals “auth-int” then both authentication and message integrity is used. Is the value “auth” then

only authentication is used. It is important to mention that the server to client authentication is made with the Authentication-Info header field, not in the WWW-Authenticate field.

When the client receives the challenge in a 401 response with a WWW-Authenticate header field inside, it uses its password and some of the parameters in the WWW-Authenticate header field to calculate the checksum. The checksum is included in the Authorization header field inside the new request from the client to server. The parameters for the Authorization header field are described in Table 3.

Name of parameter	Description
username	Client's username for specified realm.
realm	String associated with the protection domain. Must contain the same value as the realm value in the 401 response from the server.
nonce	Unique string that is created by the server for each 401 response. Should contain the same value as the nonce value in the 401 response from the server
digest-uri	The URI from the Request-URI of the Request-Line.
response	The calculated checksum, hexadecimal encoded.
algorithm (optional)	Specifies the algorithm to use for the checksum calculations.
cnonce (optional)	If the qop-options in the received 401 response is not empty, this value is used by the server to authenticate it self to the client by using the Authentication-Info header field. The value is generated by the client and it should be unique. If the qop-options is empty in the received 401 response, then this parameter must not be used.
opaque (optional)	A string of data specified by the server, which should be returned by the client unchanged.

Name of parameter	Description
message-qop (optional)	Specifies the quality of protection that the client has applied to the message. It must be some of the qop-options values specified in the 401 response from the server.
nonce-count (optional)	Specifies the number of requests that have used the nonce specified in this header field
auth-param (optional)	Future extensions.

Table 3 Authorization header field parameters.

The parts that are included in the checksum depend on the qop-options and algorithm values. Figure 2a and Figure 2b show the different checksums calculations, where $H(x)$ denotes the hash function and $concat(s_0, \ldots, s_n)$ denotes a concatenation of $n + 1$ strings.

```
checksum = H(concat(H(A1), ":", nonce, ":", H(A2)))
```

Figure 2a The checksum calculation if the message-qop value is unspecified.

```
checksum = H(concat(H(A1), ":", nonce,":" ,nonce-count, ":",
               message-qop, ":", H(A2)))
```

Figure 2b The checksum calculation if message-qop value is either “auth” or “auth-int”.

The values of A1 and A2 depend on the message-qop and the algorithm used for the checksum. Figure 3a to Figure 3d show the calculations of A1 and A2 for different parameter values.

```
A1 = concat(username,":", realm, ":", password)
```

Figure 3a The value of the algorithm parameter is “MD5”.

```
A1 = concat(H(concat(username, ":", realm,":", password)), ":",
     nonce-value, ":", cnonce-value)
```

Figure 3b The value of the algorithm parameter is “MD5-sess”.

```
A2 = concat(Method, ":", digest-uri)
```

Figure 3c The message-qop value is “auth” or unspecified.

```
A2 = concat(Method, ":", digest-uri, ":", H(entity-body))
```

Figure 3d The message-qop value is “auth-int”.

Following explanation about MD5-sess is given in RFC 2617 [4]:

> *“The "MD5-sess" algorithm is intended to allow efficient 3rd party authentication servers; for the difference in usage.”*

The MD5-sess algorithm can not be specified if the message-qop value is not present. The “Method” used in the calculation of A2 is the SIP request method name. The entity-body is not the same as the message body and the following description is given in RFC 1945 [13]:

> *“When an Entity-Body is included with a message, the data type of that body is determined via the header fields Content-Type and Content-Encoding. These define a two-layer, ordered encoding model:*
>
> *entity-body := Content-Encoding(Content-Type(data))*
>
> *A Content-Type specifies the media type of the underlying data. A Content-Encoding may be used to indicate any additional content coding applied to the type, usually for the purpose of data compression, that is a property of the resource requested. The default for the content encoding is none (i.e., the identity function).”*

When the server receives the response from the challenge it calculates the checksum with the same parameters as the client. If the client is authorized to access the protection domain and the checksum result is equal to the value in the received response parameter then the client is allowed to access the protection domain. If the client is authorized, future responses from the server may contain the Authentication-Info header field to provide mutual authentication, some message integrity and new nonce generation. The parameters for the Authentication-Info header field are described in Table 4.

Name of parameter	Description
nextnonce	Nonce that the server whishes the client to use for a future authentication response.
message-qop (optional)	Same as in the Authorization header field.
response-auth (optional)	The calculated checksum, hexadecimal encoded.
cnonce (optional)	Same as in the Authorization header field.
nonce-count (optional)	Same as in the Authorization header field.

Table 4 Authentication-Info header field parameters.

If the server changes the nonce on every request from the client then it must generate new challenges for every request, which is inefficient. To avoid this the server can include the Authentication-Info header field in the responses to the client, so that the client knows the new nonce before the server changes it.

The response-auth value is used by the server to prove that it knows the client's secret. The calculation of the checksum is almost the same as in the Authorization response. Figure 4a and Figure 4b show the differences for the checksums calculations.

```
A2 =  concat(":", digest-uri-value)
```

Figure 4a The message-qop value is "auth" or unspecified.

```
A2 =  concat(":", digest-uri, ":", H(entitybody))
```

Figure 4b The message-qop value is "auth-int".

3.2 Integrity

Message integrity assures that only authorized parties are able to modify the message, i.e. an unauthorized third party can not modify the message without detection by the authorized receiver. In most cases the integrity protection is only applied to special parts of a message, i.e. parts that are not allowed to be modified.

The problem with integrity protection in SIP is that the messages are allowed to be changed by network intermediaries. This means that the parts that are allowed to be changed can not be included in the integrity protection.

The standard of SIP defines how to use Pretty Good Privacy (PGP [20]) to supply signatures of message parts that are not changed by network intermediaries. A signature is one of several ways to supply message integrity by using hash functions. Even if the standard document for SIP, Request For Comments (RFC) 2543, is a standard track with “Draft Standard” [17] status it will probably be obsolete and replaced with the draft “draft-ietf-sip-rfc2543bis-0x”, where x is the version number. In current version, version 5, the use of PGP signatures has been excluded. Today there only exists a very limited integrity protection in SIP which is provided by the Digest authentication scheme, described in the previous section. Although it is wrong to say that SIP does not support other integrity protection, the protocol does not make any standard definition how to implement these.

The section above stated that there is no integrity protection for the complete message in SIP. But there exist definitions on how to protect the integrity of the message bodies by using S/MIME, see “Clear signing” on page 43. The message bodies consist of descriptions of the media transfer which may be interesting to protect against modification.

Even if there is no integrity protections defined for the complete message in SIP, it is always possible to use transport and network layer protocols that provide this feature. In version 5 of the SIP draft from IETF there is a recommendation for SIP endpoints to support Transport Layer Security (TLS [5]). IP Security (IPSec [6]) is also a hot topic for providing integrity protection.

3.3 Confidentiality

Message confidentiality assures that only authorized parties are able to read the content. In SIP, encryption is used to provide message confidentiality.

In RFC 2543 there exist two header fields, the Encrytion and Response-Key, that may be used for end-to-end encryption. There also exists a definition on how to use these header fields for PGP encryption. As in the case of message integrity this definition has been excluded in the coming draft. The header fields have also been excluded which indicates that the encryption will be out of the scope in the SIP protocol and probably be moved to lower layer protocols.

The problem with end-to-end encryption in SIP is that network intermediaries have a need to view certain parts of the message to be able to accomplish their tasks. These parts may be sensitive to the users, which means that most of the benefits of using encryption is lost. There is also a concern about the key-exchange because end-to-end encryption algorithms are based on keys shared by different users. SIP does not define any mechanism for key-exchange as lower layer protocols like TLS and IPSec do. The conclusion is that it is not likely to be any support for end-to-end encryption in the future standard for SIP.

Although the complete message will not be encrypted, it is however possible to encrypt the message body end-to-end by using S/MIME, see “Encryption of a MIME entity” on page 41.

3.4 Analysis

The intention with the following section is to give the reader a careful analysis of the security mechanisms in SIP. The analysis is based on the latest draft of the SIP protocol [2] from IETF.

Modern cryptology schemes may provide protection against different security threats. But there are also other aspects to consider when designing an application e.g. performance, power consumption, user-friendliness etc.

SIP will be implemented on many different platforms with some of them having long round-trip times and limited power supply, e.g. mobile phones. Complex schemes that provide strong protection will probably not be suitable in SIP. Constructing a new scheme based on the requirements and limitations is always possible but will require lot of time to develop. The history shows that new schemes always have serious security flaws when they first are released for public use. Even if the core of the new scheme is kept secret it does not prevent users from making reverse engineering to reveal the secrets.

IETF have obviously thought about the requirements and limitations when they designed the security framework for SIP but there is still much more work left to be done. The framework contains only Basic and Digest authentication today, which is not sufficient to prohibit all security threats. The framework must be extended to prevent more security threats as the man-in-the-middle (MITM) attacks and eavesdropping.

3.4.1 Threats

Threats can be split into different areas, shown in Figure 5.

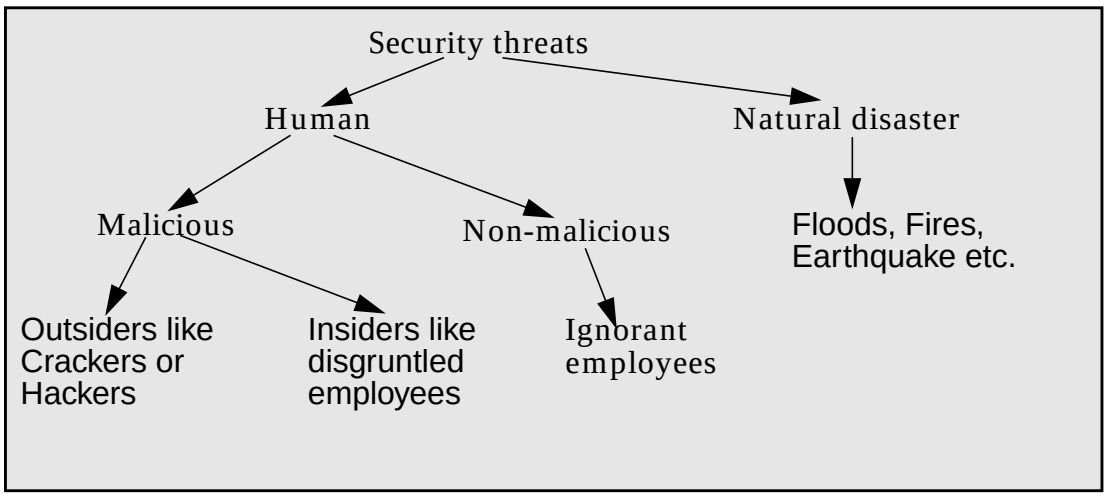

Figure 5 Different areas of threats [21].

The most serious threats are those made by insiders who have both the knowledge and capabilities to make serious harm to the target. The non-malicious persons are often unaware of the threats they cause, i.e. they do not have the intention to harm the target. A unintentional threat could be software bugs and loose security thinking but events like natural disaster also fits this category. Natural disasters are difficult to be protected against but a good recovery plan may decrease the damage to minimum.

The most famous group of malicious outsiders that makes intentional threats is hackers and crackers who often use different types of self developed tools to harm the target. They often gain knowledge about weak points on the target before the actual attack occurs. The attack could be either active or passive. The passive attack is used to obtain information and the active attack is used to modify information.

3.4.1.1 Threat classes

There exists several classifications on security threats and I have chosen the following one, which is broader then the average classification:

- Unauthorized access
- Modification
- Disclosure
- Repudiation
- Denial of Service (DoS)

The Denial of Service is the hardest one to be protected against. The main reason for this is that systems that try to protect against a specific DoS threat often are exposed to other types of DoS threats.

3.4.2 Potential attacks on SIP

There exist several potential attacks on SIP and some of them are easy to accomplish with almost no knowledge about security. Fortunately these attacks are easy to be protected against with supported security mechanisms in SIP. Some potential attacks are more difficult to be protected against because of the limited security mechanisms for encryption and integrity protection in SIP.

3.4.2.1 Spoofing

Spoofing is an active attack. The malicious person, the attacker, constructs a new message which is sent to the target. The headers and the body in the message are written so that the receiver believes that the message originates from another source. If the attacker is able to do this, it may open up several other attacks, e.g. eavesdropping and security downgrading.

The simple spoofing attacks on SIP have properties similar to the spoofing attacks on the SMTP protocol. Spoofing attacks on the SMTP protocols are in most cases used for sending spams, i.e. e-mails that the receiver has not asked for. The receiver is unable to backtrace these e-mails because the source address is spoofed and in most cases the specified source address does not

exist. A simple spoofing attack in SIP would be to spoof the From and / or the Subject header fields. And if the receiver uses a SIP UA that does not make any reverse address lookup and checks it against the Internet Protocol (IP [22]) address, the user is unable to decide if the message really originates from the specified source. Even if the UA makes a reverse address lookup it does not assure a correct IP address, because even the IP address can be spoofed [25]. The following example shows how user Alice wants Bob to believe that the SIP message comes from Charlie by spoofing the From and Subject header fields:

```
INVITE sip:bob@somewhere.com SIP / 2.0
Via: SIP / 2.0 / UDP 10.0.0.3:5060
To: Bob B.<sip:bob@somewhere.com>
From: Charlie C. <sip:charlie@somewhere_else.com>; tag=1234
Call-ID: 1234567
CSeq: 1 INVITE
Contact: Alice <alice@10.0.0.3>
Subject: Greeting from Charlie!!
...
```

Figure 6 Spoofed SIP message.

It can be seen in Figure 6 that Alice has to specify her own IP address in some header fields to assure that Bob sends the response back to her. She must do this because her intention is to have a dialog with Bob and he should believe that he is invited by Charlie. A more advanced variant of this attack would be to use a spoofed IP address. Then Alice would receive the responses from Bob even if she specifies Charlie's IP address.

3.4.2.2 Eavesdropping

Eavesdropping in an IP network is not as easy as it sounds, although it is easier than eavesdropping a telephone call. One exception is the Ethernet network, Ethernet is the most common network protocol running on local networks and is easily monitored because all messages pass all hosts. The messages are passed to all hosts because they share a common cable or hub. A user's network device may then be put in a promiscuous mode that enables it to pass all received messages through the protocol stack even if the messages are addressed to another host. Fortunately, most local networks use switches instead of hubs which prohibit this type of broadcasted messages. A switch uses forwarding tables to decide which of its connected hosts that should receive a particular message. This means that a user only can eavesdrop its own messages in a switched network if the forwarding tables are static.

In an IP network all hosts are addressed by their IP address and not the Media Access Control (MAC) address that is used in Ethernet networks. This means that all hosts and switches need to have a table that translates an IP address to a MAC address to be able to send messages to a Ethernet network device. These tables are often built dynamically by using the Address Resolution Protocol (ARP [23]). An attacker may spoof ARP messages to be able to receive

messages addressed to another host even in a switched network. However, this type of attack may only work inside a local network because routers do not forward ARP messages to an outside network. A router is actually an advanced switch that interconnects local networks with e.g. internet. For more details about ARP spoofing, see [24] and “ARP spoofing” on page 86.

A similar method to the ARP spoofing is the Internet Control Message Protocol (ICMP [26]) spoofing, see [27] for a more detailed description.

To be able to eavesdrop messages from an arbitrary IP address you may have to use more sophticated methods, e.g. implant a trojan horse in the target, let an insider re-configure a specific router or make some cable / hardware modification to the network infrastructure. The most common ways to distribute a trojan horse is to attach it in an e-mail or to merge it with a public software.

3.4.2.3 Buffer Overflow

Buffer overflow is a witty method for an attacker to gain certain privileges at the target. All operating systems (OS) have so called system calls that programmers may use to access different OS resources. These system calls are often written in some high level language like the C-language. These languages may be delivered with different libraries that contain functions for manipulating strings. Some properties of these functions may be used for launching a buffer overflow attack because they do not check for strings that are too long to fit in the allocated buffer. If a particular system call uses a vulnerable string manipulation function, the attacker may be able to execute arbitrary code with high privileges by passing a string to the system call that is too long. The arbitrary code is often written for opening a terminal window. Fortunately, there are several simple techniques to eliminate buffer overflow vulnerableties. But developers must be aware of it and always check for string lengths.

A complete description how to launch a buffer overflow attack will not be given because it requires a full understanding how a modern processor manage function calls. A detailed example is given in [28].

It is not only system calls that may be the target for the buffer overflow attack but also a SIP stack could be vulnerable to these types of attacks. Of course, this depends on how the stack is implemented. The following example shows what a vulnerable code for parsing the From header field could look like:

```
FromHeader ParseSipFromHeader(char *lpMessageLine) {
   char temporyStorage[256];
   FromHeader header;
   strcpy(temporyStorage, lpMessageLine);
   ...
   return(header);
}
```

Figure 7 Example of vulnerable code in a SIP stack.

If a developer uses the code in Figure 7 when implementing a SIP proxy server an attacker may be able to crash the server by sending a From header field that contains more than 256 characters. The source of the vulnerability is the fourth line in Figure 7 which overflows the allocated buffer on the second line.

3.4.3 Description of potential attacks on SIP

The following tables describe different types of potential attacks on SIP.

3.4.3.1 Intercepted messages

Id	Description	Achievement	Threat class
A1.1	Eavesdropping messages.	Gain knowledge about sensitive information regarding signaling information.	Disclosure
A1.2	Rogue proxy that modifies the received message body and forwards it to the destination.	Able to falsify the information in the message body, e.g. replace some media codec with another that presents the media with lower quality.	Modification, Disclosure
A1.3	Rogue proxy that modifies the SIP message header and forwards it to the destination.	Able to change the request method. Able to force further messages to be forwarded through other proxies.	Modification, Denial of Service, Unauthorized access.

Table 5 Potential attacks for a malicious person who is able to intercept messages.

3.4.3.2 False and spoofed messages

Id	Description	Achievement	Threat class
A2.1	Falsified From header field.	Identity impersonation. In the E-mail case, a user often checks the From value to decide if it originates from an unserious source, i.e. a spam. Same effect might also be achieved in the SIP case, where a user rejects unknown SIP addresses.	Unauthorized access, Repudiation, Denial of Service
A2.2	Spoofed BYE request.	Able to terminate a session.	Denial of Service

Id	Description	Achievement	Threat class
A2.3	Spoofed CANCEL request.	Able to terminate a pending INVITE request.	Denial of Service
A2.4	Spoofed ACK with changed IP address in the SDP message body.	Able to hijack the media transfer.	Unauthorized access, Denial of Service
A2.5	Spoofed REGISTER request.	Able to remove registrations from a user.	Denial of Service, Modification
A2.6	Spoofed REGISTER request.	Able to overload the registrar with registrations.	Denial of Service, Modification
A2.7	Spoofed REGISTER request.	Able to initiate a DoS attack by registrating multiple contacts to same host.	Denial of Service, Modification
A2.8	Security downgrading.	Able to downgrade authentication scheme to Basic authentication.	Unauthorized access, Disclosure
A2.9	Chosen plaintext.	Able to make cryptanalysis easier for MD5.	Unauthorized access
A2.10	Sending false 6xx responses.	Able to fool a user that a called user is not willing to accept calls.	Denial of Service

Table 6 Potential attacks for a malicious person who is able send false or spoofed messages.

3.4.3.3 Repeated or deliberately erroneous messages

Id	Description	Achievement	Threat class
A3.1	Message flooding.	The targets phone never stops ringing and the target is unable to receive non-malicious requests.	Denial of Service
A3.2	Buffer overflow.	Able to make the target to crash or to run arbitrary code.	Denial of Service, Modification, Unauthorized access
A3.3	Replay messages.	Able to make authorized requests.	Unauthorized access

Table 7 Potential attacks for a malicious person who is able repeat or make deliberately erroneous messages.

3.4.3.4 Intercepted media

Id	Description	Achievement	Threat class
A4.1	Spoofed SIP message body.	Able to receive sensitive media information.	Disclosure

Table 8 Potential attacks for a malicious person who is able to spoof the SIP message body.

3.4.3.5 Unwanted and spoofed media

Id	Description	Achievement	Threat class
A5.1	Spoofed INVITE request.	Able to send unwanted media data to the target.	Unauthorized access

Table 9 Potential attacks for a malicious person who is able to send spoofed media data.

3.4.3.6 Password guessing

Id	Description	Achievement	Threat class
A6.1	Online password guessing.	Able to guess a small subset of valid passwords to gain authorized access.	Unauthorized access
A6.2	Offline password guessing.	Able to guess a large subset of valid password to gain authorized access.	Unauthorized access

Table 10 Potential attacks for a malicious person who is able guess passwords.

3.4.4 Protection

The potential attacks that were mentioned in the section “Potential attacks on SIP” on page 30 are all possible to accomplish on an unprotected system. Fortunately, there are several ways to protect against them. This section will, if possible, propose methods to be protected against them.

Id	Protection
A1.1	Because there is no support for encryption between SIP user agents it is impossible to protect the information that is eavesdropped. SIP has to rely on the underlying protocols or network protection against eavesdropping.
A1.2	The only way to protect against this kind of attack is by using the integrity protection that is supported by the Digest authentication, i.e. use “auth-int” as the qop-value in the Authorization header field.
A1.3	The Digest authentication detects modification of the request method if the qop-value is specified. Modification of the header fields are not possible to detect because there is no integrity protection for them.
A2.1	By demanding either Basic or Digest authentication from the source, the receiver can tell from whom the message originates. But this does not guarantee that the From field is valid because some rogue proxy may have changed the value, see A1.3.
A2.2	Using Digest authentication for Bye requests and with a specified qop-value. This guarantees that the request originates from a source that knows the secret and that the request method has not been changed. Basic authentication can also be used to verify the origin of the message.
A2.3	See protection against A2.2.

Id	Protection
A2.4	First, it should be mentioned that there exist two methods of negotiating the media capabilities. In the first method the negotiation occurs between the INVITE request from the calling party and the 200 OK response from the called party. In the second method the negotiation occurs between the 200 OK response from the called party and the ACK from the calling party. This means that this attack is only possible to accomplish in the second method because in the first method the called party ignores the message body contained in the ACK. This attack is quite hard to accomplish if the malicious person is unable to eavesdrop the session because he / she has to guess at least five header values, namely the From, Request-URI, Call-ID, To, Via values. Some of these header values are unique and cryptographically random which makes the task harder. If the called party can not match these header field values against header field values from the INVITE request, then it will discard the ACK. If the malicious person is able to eavesdrop and modify the ACK request, then the called party has to force the calling party to use “auth-int” as the qop-value when it challenges the INVITE request. The need of challenging the INVITE request is necessary because the called party is not allowed to challenge the ACK request because no response can be sent to an ACK. By challenging the INVITE request the called party forces the calling party to use the same credentials in the Authorization header field in the ACK.
A2.5	Basic or Digest authentication should be used by the registrar. Digest authentication is preferred if eavesdropping is possible.
A2.6	Basic or Digest authentication should be used by the registrar. Digest authentication is preferred if eavesdropping is possible. It is also possible to set a limit on the number of registrations a user can have. But that solution may be used for launching a DoS attack.
A2.7	Basic or Digest authentication should be used by the registrar. Digest authentication is preferred if eavesdropping is possible.
A2.8	If the client supports Digest authentication it can be configured to demand it. The client can also remember the strongest authentication scheme used for a particular server. If the server wants a weaker scheme the client should show a warning message before it uses the weaker one.
A2.9	Almost every parameter in the Digest authentication is chosen by the server which can be used by a rogue server to launch a chosen plaintext attack. But the client can use the cnonce value to make the checksum stronger. However, there is today no known way to break the one-way property of MD5.

Id	Protection
A2.10	The client should ignore the responses if it requested authentication. The problem is that new requests from the client will reach the same source as the invalid responses, and the process will repeat. One solution is to choose another signaling path by choosing another proxy, i.e. change outbound proxy.
A3.1	It is very hard for the client to protect against this type of attack because of its nature. Authentication will prevent the annoying phone signals but does not protect against the actual attack.
A3.2	The only protection is to use some sort of safe compiler when building the SIP stack because developers may miss some boundary checks. Dynamically allocated memory may be used instead of local variables. All header values that specify some type of message length should be considered as unsafe and checked carefully. Try to avoid usage of environment variables that an arbitrary user can modify.
A3.3	Replays are only a concern when Digest authentication is used. To get full protection against replays you should change the nonce-value for each challenge. That type of protection requires a lot of resources. Another option is to have strong nonces, i.e. include the clients IP, method, time-stamp and a server key in the nonce generation.
A4.1	The only protection is to use Digest authentication and set the qop-value to “auth-int”. The server can then verify if the content in the message body has been changed or not. The responses from the server can also have modified message bodies and therefore the client should demand the Authentication-Info header, with the qop-value set to “auth-int” in the responses.
A5.1	Basic or Digest authentication will protect against this type of attack.
A6.1	The used password should be hard to guess, i.e. no default password or passwords that can be found in a dictionary should be used.
A6.2	The used password should at least contain eight characters and should be randomly generated.

Table 11 Proposed protections against potential attacks on SIP.

3.4.5 Published attacks

Because the SIP protocol is relatively young and only a few public SIP servers are up and running, no published attacks have been found. This is will probably not be true in the near future because several new SIP applications will soon be released.

4 Proposed security mechanisms in SIP

This chapter covers different security protocols or schemes that may either be integrated with the SIP protocol in the future or used together with SIP for improving the security. Most of these protocols or schemes are suggestions and recommendations from the IETF, but some of them originate from telecommunications communities.

4.1 S/MIME

Secure / Multipurpose Internet Mail Extension (S / MIME [19]) is a security extension to the MIME standard.

4.1.1 MIME

MIME is an extension to general message format of E-mails, defined in RFC 822.The main purpose for extending this standard was to solve the problem of decoding and encoding different file formats and national language characters, but there are also other minor problems which MIME solves.

Implementations that support MIME, like SIP, should support the three header fields, described in Table 12.

Header field	Description
MIME-Version	Its value must be “1.0”.
Content-Type	Describes the data in the body of the message and it should be specific enough, that the receiver can decide how to represent the data to the user. The content type may also be extended with one or several parameters.
Content-Transfer-Encoding	Specifies the type of transformation that has been applied to the data before it was sent, e.g. radix-64 or binary encoding.

Table 12 MIME header fields.

SIP, as well as HTTP, does not have the Content-Transfer-Encoding header field defined, but instead it defines the Content-Encoding header field with a slightly different purpose. Both SIP and HTTP are “8bit-clean” transfer protocols and therefore these protocols has no explicit need of the Content-Transfer-Encoding header field. The Content-Encoding header field is used to specify a more advanced coding, e.g. compression.

The Content-Type header field is the most important header field in MIME and its values are organized in different content types. Each content type is

organized in different subtypes. For SIP, the most important and most used content type is the *application* type with *sdp* as its subtype, which means that the data is a SDP message, see “SIP” on page 17. Other content types and subtypes are rarely used, but sometimes the *multipart* content type is used. The *multipart* content type allows more than one type of data in the message body and each content type is separated with a boundary string. One example of using the *multipart* content type in SIP is when a user wants to send a SDP message, but also wants the other user to see a picture of the caller when he or she receives the invitation.

4.1.2 S/MIME Functionality

S/MIME offers the ability to sign, encrypt or both sign and encrypt MIME entities. If both encryption and signing is desired then nestled S/MIME entities should be used. Depending on the situation it may be preferred to encrypt first and then sign or vice versa.

To be able to offer these abilities, S/MIME introduces some new MIME subtypes for different content types, described in Table 13.

S/MIME entities	Description
multipart/signed	A *multipart* content type containing two parts. The first part can be of any MIME content type and the second part is the signature of the first part. The main reason to separate the signed data from the signature, is that entities, which do not support S/MIME, will be able to read the signed data. This functionality is often called clear-signing.
application/pkcs7-mime	A signed, encrypted S/MIME entity or just an entity that contains a public key certificate, depending on what parameter that has been specified for this content type.
application/pkcs7-signature	The content type of the second part in the *multipart/signed* content type, described above.
application/pkcs10-mime	Content type used for requesting a public key certificate from a certification authority.

Table 13 Entities defined in S/MIME.

The application/pkcs10-mime entity is seldom used and will not be covered in this document, more information of this content type may be found in [19].

The *application/pkcs7-mime* entity has a defined *smime-type* parameter with a value which decides what the result of the S/MIME processing consists of, i.e. encrypted data, signed data or a single public key certificate. The result of the S/MIME processing is called an object. Different values of the *smime-type* parameter are described in Table 14.

smime-type value	Description
signed-data	The S/MIME entity contains an object which among others contains the signature of the MIME entity.
enveloped-data	The S/MIME entity contains an object which among others consists of the encrypted MIME entity.
certs-only	The S/MIME entity contains an object which only contains a public key certificate.

Table 14 Values of the *smime-type* parameter.

The difference between the object of an *application/pkcs7-signature* entity, contained in the *multipart/signed* entity, and a object of an *application/pkcs7-mime* entity, with the *smime-type* parameter value of “signed-data”, is that the signed messages are not included in the object for the *application/pkcs7-signature* entity.

The object, which is the result of the S/MIME processing, often consists of radix-64 encoded binary data that have been structured according to the Cryptographic Message Syntax (CMS [29]). This specification is quite complex and consists of several parameters and attributes that define how an implementation should among others store signatures, encrypted data, public key certificates and cryptographic capabilities. This document will not cover how to construct objects according to the CMS specification, but instead it will mention what the object contains in a more formal way.

4.1.2.1 Encryption of a MIME entity

Encryption of a MIME entity, in S/MIME, is accomplished by creating an *application/pkcs7-mime* entity with a *smime-type* parameter of “enveloped-data” and then attaching an object that among others contains the encryption of the MIME entity, the senders public key certificate, the used encryption algorithm and a public key encypted session key.

The following procedure, borrowed from [1] and [19], describes how to construct a S/MIME entity that contains an encrypted MIME entity:

1. Prepare the MIME entity according to standardized rules and canonicalize the MIME entity. The canonicalization procedure is described in the MIME specification [18].

2. An pseudorandom session key is generated for a symmetric encryption algorithm. The supported algorithms are Triple DES and RC2 [48] with a key size of 40 bits. Descriptions about these algorithms may be found in [1].
3. Encrypt the session key by using the RSA algorithm and use the public key in the receivers public key certificate.
4. Encrypt the MIME entity with the chosen algorithm and attach the result in the CMS object. The information of used algorithm to encrypt the MIME entity, used algorithm for encryption of the session key, used session key and the senders public key certificate are also included in the object.
5. Encode the CMS object with radix-64.
6. Attach the encoded CMS object to the *application/pkcs7-mime* entity.

All S/MIME implementations support the Diffie-Hellman algorithm for generation of the session key. If this method is used the points 2 and 3 above are excluded. Instead the session key is generated according to the Diffie-Hellman Key Agreement Method, described in RFC 2631 [30].

An example, from RFC 2633, of a S/MIME entity that consists of an encrypted MIME entity is seen in Figure 8.

```
Content-Type: application/pkcs7-mime; smime-type=enveloped-data;
name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m
rfvbnj756tbBghyHhHUujhJhjH77n8HHGT9HG4VQpfyF467GhIGfHfYT6
7n8HHGghyHhHUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H
f8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
0GhIGfHfQbnj756YT64V
```

Figure 8 Example of an encrypted MIME entity.

4.1.2.2 Signing of MIME entity

Signing of a MIME entity, in S/MIME, is accomplished by creating an *application/pkcs7-mime* entity with a *smime-type* parameter of “signed-data” and then attaching an object that among others contains the signature of the MIME entity, the senders public key certificate, the used algorithm for encryption of the message digest and the algorithm used to calculate the message digest.

The following procedure, borrowed from [1] and [19], describes how to construct a S/MIME entity that contains a signed MIME entity:

1. Prepare the MIME entity according to standardized rules and canonicalize the MIME entity. The canonicalization procedure is described in the MIME specification [18].
2. Choose algorithm to use when calculating the message digest. Supported algorithms are MD5 and SHA-1.

3. Calculate the message digest by processing the MIME entity through the chosen algorithm.
4. Encrypt the message digest by using the RSA algorithm and use the public key in the receivers public key certificate.
5. Attach the encrypted message digest in a CMS object. The information of used algorithm for the message digest calculation, used algorithm for encryption of the message digest, the sender public key certificate and the content being signed are also included in the object.
6. Encode the CMS object with radix-64.

All S/MIME implementations support the Digital Signature Standard (DSS [31]) and if it is used point 4 above is excluded. Instead, the signature is calculated by using the Digital Signature Algorithm (DSA [31]). The public key used to calculate the message digest from the signature is supposed to be included in the public key certificate, which is included in the CMS object.

An example, from RFC 2633, of S/MIME entity that consists of a signed MIME entity is seen in Figure 9.

```
Content-Type: application/pkcs7-mime; smime-type=signed-data;
name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m
567GhIGfHfYT6ghyHhHUujpfyF4f8HHGTrfvhJhjH776tbB9HG4VQbnj7
77n8HHGT9HG4VQpfyF467GhIGfHfYT6rfvbnj756tbBghyHhHUujhJhjH
HUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H7n8HHGghyHh
6YT64V0GhIGfHfQbnj75
```

Figure 9 Example of a signed MIME entity.

4.1.2.3 Clear signing

One problem with the signing procedure described in the previous section is that users who do not support S/MIME neither can read the signed MIME entity nor verify the signature. Clear signing provides non-supported S/MIME users with the ability to read signed MIME entities. To be able to verify the signature, users must support S/MIME.

When a user wants to clear sign a MIME entity it creates a *multipart/signed* S/MIME entity. This entity contains two parts, where the first part is a copy of the MIME entity, including its MIME headers, and the second part is an *application/pkcs7-mime* entity with a *smime-type* parameter of “signed-data”. The procedure for creating the *application/pkcs7-mime* entity is almost the same as described in previous section. The difference is that the content being signed and the information of used algorithm for message digest calculation is not included in the CMS object, so-called detached signature. Instead, the content being signed may be found in the first part and the information of used message digest algorithm is included in the *micalg* parameter of the *multipart/signed* entity.

An example, from RFC 2633, of a S/MIME entity that consists of a clear signed MIME entity is seen in Figure 10.

```
Content-Type: multipart/signed;
protocol="application/pkcs7-signature";
micalg=sha1; boundary=boundary42
--boundary42
Content-Type: text/plain
This is a clear-signed message.

--boundary42

Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s
ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756

--boundary42--
```

Figure 10 Example of a clear signed MIME entity.

4.1.2.4 Public key certificates

Both signing and encryption of MIME entities uses public key certificates. A certificate binds an entity's distinguished name to a public key with a digital signature. In most cases the certificates are created and signed by a certificate authority (CA). To be able to receive a certificate from a CA, a user has to know the CA's public key. When the user requests for a new certificate or updates an existing one it sends its public key and some personal information to the CA, which creates and signs the certificate with its private key. The CA also adds the personal information to the certificate which refers to the subject of the certificate. The user may then distribute this certificate to other users. These users in their turn should verify the certificate with the issuing CA before they trust the owner of the certificate. If the owner of the certificate is trusted, then the user saves the certificate in a so called keyring, for future communication.

A certificate may be created and signed by the holder of the private key, a so-called self-signed certificate. Self-signed certificates should not be considered to be secure, but it is a flexible way for end users to distribute their public keys. One reason why S/MIME may fail to be a popular standard for securing communications between end users is the lack of a prevalent public key infrastructure.

As mentioned in the two previous sections, the public key certificate for the sender is included in the CMS object of a signed or encrypted entity. In addition to this, S/MIME defines entities that only have the purpose of requesting or sending public key certificates. For requesting a public key certificate S/MIME uses the *application/pkcs10-mime* entity and for sending it S/MIME uses the *application/pkcs7-mime* entity with a *smime-type* parameter of "certs-only".

The public key certificates in S/MIME must use Public Key Infrastructure (PKIX) certificates, described in [32]. PKIX certificates are based on X.509 [33] certificates.

4.1.3 S/MIME in SIP

The proposed use of S/MIME in SIP conforms to the S/MIME specification [19], but with some exceptions:

- Only *multipart/signed* and *application/pkcs7-mime* S/MIME entities are supported in SIP messages.
- SIP User Agents that support S/MIME must support signed MIME entities and S/MIME entities that only contain a public key certificate. A SIP UA may support encrypted MIME entities.
- *multipart/signed* must be used with detached signatures for compatibility with non-S/MIME users.
- S/MIME entities should have a Content-Disposition header field, and the value of the *handling* parameter should be “required”.
- If a SIP UA does not have a public key certificate for the intended receiver in its keyring, it cannot send an encrypted MIME entity. The SIP UA may send an OPTION request to the other SIP UA with a detached signature in order to solicit the certificate of the remote side.
- A SIP UA should include its S/MIME capabilities and preferences in the CMS object for making future communications easier. A SIP UA may encourage the receiver to respond with signed MIME entities, by specifying it in the CMS object.
- SIP UAs that support S/MIME must at minimum support SHA-1 as digital signature algorithm and Triple DES as the encryption algorithm. Other signature and encryption algorithms may be supported.
- Each MIME entity should be signed with only one public key certificate. If a SIP UA receives a signed MIME entity with multiple signatures, the outermost signature should be treated as the single public key certificate for this entity.

4.1.3.1 Public key Certificates

A SIP UA that support S/MIME must have a keyring specifically for end-users’ public key certificates. The keyring maps the username and domainname part of the end-users’ SIP URI, i.e. username@domainname, to its public key certificate.

An overview how a SIP UA obtains, stores and uploads its public key certificates may be seen in Figure 11.

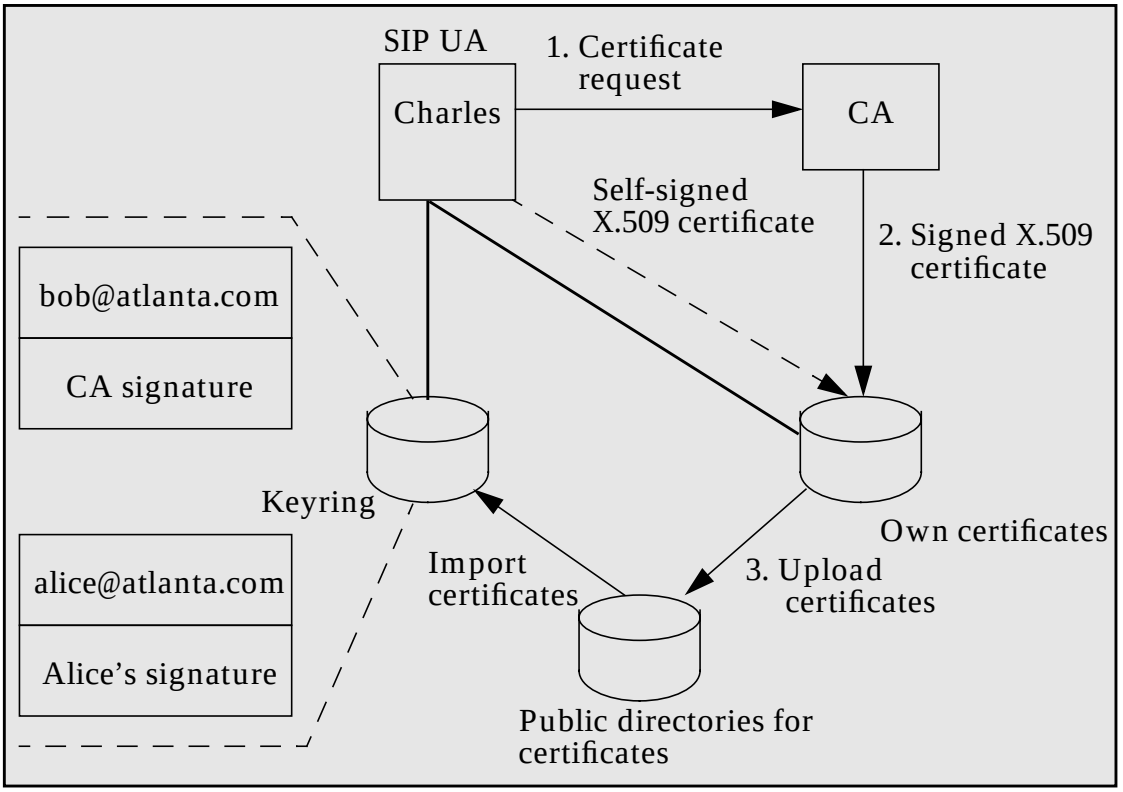

Figure 11 Obtaining, storage and uploading of public key certificates.

The public key certificate should be acquired from known public CAs, but users may create self-signed public key certificates. As mentioned earlier, self-signed public key certificates should not be considered as secure. When the user has received or created a public key certificate, the user should publish it in any public directories. Other SIP UAs may then be able to access these directories and automatically or manually download the public key certificate.

When a SIP UA sends a signed MIME entity it should follow the scheme in Figure 12.

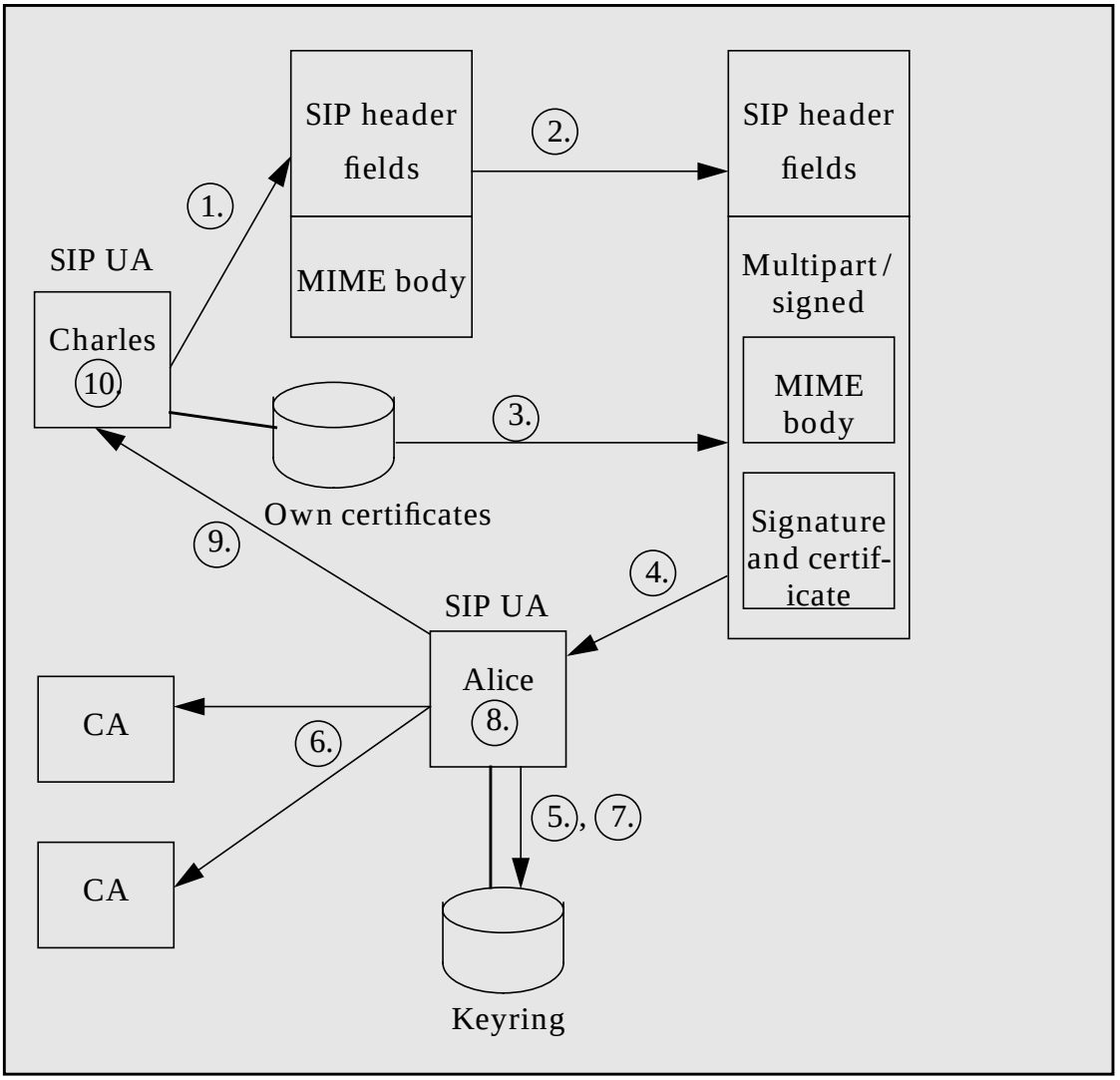

Figure 12 Scheme for sending and receiving a signed SIP message body.

Each item in Figure 12 is described beneath:

1. The SIP UA creates the SIP message and attaches a MIME body to it.
2. The SIP UA creates a *multipart/signed* S/MIME entity, that contains the MIME body and an *application/pkcs7-signature* S/MIME entity.
3. The SIP UA signs the MIME body by using its private key and includes the public key certificate in the CMS object of the *application/pkcs7-signature* S/MIME entity.
4. The SIP UA sends the message. If the receiving SIP UA does not support S/MIME, then items 5-8 should be skipped.
5. The receiving SIP UA takes the message apart and tries to find the included public key certificate in its keyring. If a public key certificate, in the keyring, has a subject that matches the From[1] header field in the SIP message, then the receiving SIP UA

1. The SIP message is assumed to be a SIP request. If it is a SIP response, then should the To header field be compared with the subject of the public key certificate.

should compare it with the received certificate. If there is a discrepancy between them, the SIP UA should notify the user and acquire the user's permission before continuing the session. If the user accepts the received public key certificate, it must add it to the keyring alongside any previous stored public key certificates at that index. Item 6 and 7 may then be skipped.

6. If the received public key certificate is not found in the keyring, then the receiving SIP UA should verify its signature with any available CA. If the receiving SIP UA is able to verify the public key certificate, then the receiving SIP UA should compare the subject in the public key certificate with the From header field. If the receiving SIP UA is unable to verify the public key certificate, because it is self-signed, or signed by an unknown CA, then the user must be notified with the status of the public key certificate. And the receiving SIP UA must have the permission from the user to accept it.
7. If the public key certificate is accepted either by the user or directly by the receiving SIP UA, then the public key certificate should be added to the keyring, indexed with the SIP URI in the From header field.
8. When the signature of the public key certificate is validated and stored in the keyring, the receiving SIP UA validates the signature of the MIME body.
9. The receiving SIP UA generates and sends a response to the other SIP UA. If the response contains a message body and the receiving SIP UA supports S/MIME, then should the message body consist of a S/MIME entity.
10. If the SIP UA receives a SIP response with a message body that consists of a MIME entity, then it should notify its user that the session could not be secured.

An almost identical scheme, as described above, may be used if the SIP UA wants to encrypt a MIME body. However, some difference exists:

- The first part of the *multipart/signed* S/MIME entity does not contain the MIME body in clear text. Instead, the encrypted MIME entity is placed there, i.e. an *application/pkcs7-mime* S/MIME entity with a *smime-type* value of "enveloped-data".
- The SIP UA may access a public directory to obtain the public key certificate for the receiving SIP UA. The public key is then used for the encryption.
- If the receiving SIP UA is unable to decrypt the MIME body, then the receiving SIP UA must reject the request and send a 493 (Undecipherable) response. The 493 response should contain a public key certificate in a *application/pkcs7-mime* S/MIME entity with the *smime-type* parameters set to "certs-only". The chosen public key certificate should correspond to the To header field value in the request.

4.1.3.2 Tunneling integrity and authentication

Beside the typical use of S/MIME entities in SIP messages, it is possible to provide some degree of end-to-end authentication or integrity for SIP message headers by using message tunneling. This can be achieved by making a complete or partial copy of the message headers and place the copy together with the original message body in an "inner" message. The inner message is then

encapsulated in a *message/sip* MIME entity which represents the body of the message. This MIME entity in its turn may be clear-signed using S/MIME. The original headers, the “outer” message, are kept in clear text and may be modified during transmission.

The receiver of the message checks the included public key certificate as described in the previous section and verifies the detached signature. If the signature is valid, the receiver should compare the header fields in the inner message with the headers in the outer message, except to header fields that are allowed to be changed by intermediate SIP servers. If the SIP UA finds any differences between the compared header fields, then the user must be notified. If the user does not accept changes of the message, the SIP UA may send a 403 (Forbidden) response if it is a request or any existing session may be terminated.

If the To, From, Call-ID and CSeq header fields are present in the inner message, then the signed MIME entity can provide limited authentication. If the SIP UA trusts the public key certificate, i.e. the public key certificate is signed by a trusted CA and the From or To header field in the inner message corresponds to the subject of the public-certificate, it may be convinced about the identity of the sender. The SIP UA should not trust self-signed public key certificates, public key certificates issued by an unknown CA or public key certificates with subject fields that do not correspond to the From or the To header field.

An example of tunneling integrity and authentication is given in Figure 13.

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: multipart/signed;
protocol=”application/pkcs7-signature”;
micalg=sha1; boundary=boundary42

--boundary42
Content-Type: message/sip

INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <bob@biloxi.com>
From: Alice <alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 147

v=0
o=UserA 2890844526 2890844526 IN IP4 here.com
s=Session SDP
c=IN IP4 pc33.atlanta.com
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000

--boundary42
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s

ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756

--boundary42-
```

Figure 13 Example of a tunneled SIP message using the signing utility in S/MIME. Note, that the *application/sdp* entity is nestled inside the *message/sip* entity. The radix-64 encoding is unnecessary in SIP because it is an “8-bit” clean protocol, but is used in the example for clarification.

4.1.3.3 Tunneling encryption

Tunneling may also be used for providing limited end-to-end confidentiality for SIP message headers. Encryption with S/MIME, however, is most utilized when just the original SIP message body is encrypted. This tunneling method differs from the one described in the previous section, where the SIP message headers were copied to the inner message. Instead of making copies of the headers, this method moves or "hides" header fields to the inner message. In this case hiding means that the header field exists in both the inner and the outer message, but the header field in the outer message does not reveal all information from the header field in the inner message. For an example, the From header field in the outer message may be sip:anonymous@anonymizer.com and in the inner message it may be sip:alice@goverment.com. The receiving SIP UA ignores the header fields in the outer message if they exist in the inner message, except to header fields do not have an end-to-end semantic and these are listed in [59].

As mentioned in the section "Public key Certificates" on page 45, the encrypted entity should be encapsulated inside a signed entity.

An example of tunneling encryption may be seen in Figure 14. The signed data is surrounded by "#" characters, but in reality the signed data will be included in a radix-64 encoded CMS object. The encrypted data is surrounded by "*" characters, but in reality the encrypted data will also be included in a radix-64 encoded CMS object.

A SIP UA that receives the message in Figure 14, will process the message as follows (handling of public key certificates is not covered):

1. It verifies the signature that is included in the CMS object of the *application/pkcs7-mime* S/MIME entity with a *smime-type* parameter of "signed-data".
2. The content, which is an *application/pkcs7-mime* S/MIME entity with a *smime-type* parameter of "enveloped-data", is taken out from the CMS object.
3. This new S/MIME entity also contains a CMS object in which the encrypted data is.
4. The encrypted content is stripped off the CMS object and then decrypted.
5. The SIP UA replaces the From header field in the outer message with the From header field contained in the *message/sip* entity.
6. The SIP UA may now parse the message as it does for ordinary messages.

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <sip:bob@biloxi.com>
From: Anonymous <sip:988776a@ahhs.aa>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/pkcs7-mime; smime-type=signed-data;
name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m

###################################################################
#  Content-Type: application/pkcs7-mime; smime-type=enveloped-data;  #
#  name=smime.p7m                                                   #
#  Content-Transfer-Encoding: base64                                #
#  Content-Disposition: attachment; filename=smime.p7m              #
#                                                                   #
#  ************************************************************     #
#  *  Content-Type: message/sip                               *     #
#  *                                                          *     #
#  *  From: Alice <sip:alice@atlanta.com>;tag=1928301774      *     #
#  *  Subject: Secret information                             *     #
#  *  Content-Type: application/sdp                           *     #
#  *  Content-Length: 142                                     *     #
#  *                                                          *     #
#  *  v=0                                                     *     #
#  *  o=alice 53655765 2353687637 IN IP4 pc33.atlanta.com     *     #
#  *  s=-                                                     *     #
#  *  t=0 0                                                   *     #
#  *  c=IN IP4 pc33.atlanta.com                               *     #
#  *  m=audio 3456 RTP/AVP 0 1 3 99                           *     #
#  *  a=rtpmap:0 PCMU/8000                                    *     #
#  ************************************************************     #
###################################################################
```

Figure 14 Example of a tunneled SIP message using the encryption utility in S / MIME.The radix-64 encoding is unnecessary in SIP because it is a “8-bit” clean protocol, but is used in the example for clarification.

4.2 Authentication with EAP

The Point-to-Point Protocol (PPP [34]) Extensible Authentication Protocol (EAP [35]) is a general protocol for PPP authentication which supports multiple authentication schemes. EAP itself does not provide authentication but it has a defined binary packet format for carrying other authentication schemes. The general structure of the EAP packet is simple and can be seen in Figure 15.

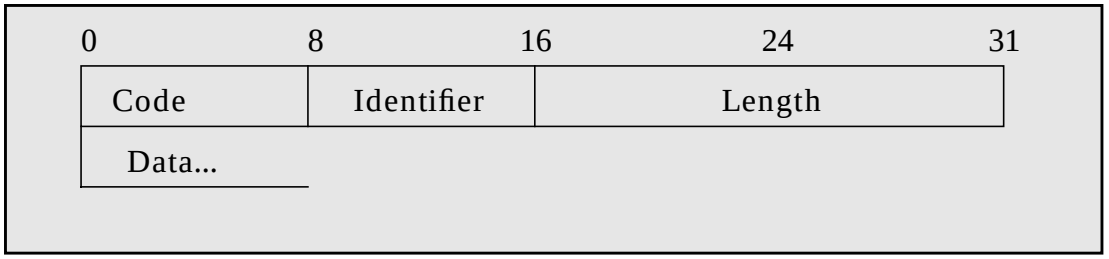

Figure 15 The generic format of EAP packets.

Each header field, shown in Figure 15, is described in Table 15.

Header field	Description
Code	Its value identifies the type of EAP packet, i.e. if it is a request, response, success or a failure packet.
Identifier	Used for matching responses with requests.
Length	Specifies the total length, in bytes, of the packet.
Data	May contain zero or more bytes and its format is determined by the value of the Code field.

Table 15 Header fields in EAP packets.

Success packets are sent by the authenticator for acknowledge after a successful authentication and failure packets are sent by the authenticator when the authentication failed. For success and failure packets the Data field is left empty and the Identifier field is identical to the Identifier field in the request that was sent by the authenticator.

For requests and responses the first byte of the Data field consists of a Type field, which describes which type of request or response the rest of the data contains. Common requests and response types have predefined Type field values, see [35] for more information about these types. Other Types field values may be defined for the particular authentication scheme that is used. The Identifier field of the response must match that of the request. The Identifier field must be changed when the authenticator sends a new request.

Figure 16 shows a common sequence of sent EAP packets during the authentication. The Identity request, seen in Figure 16, is one of the predefined Type field values and is used by the authenticator to identify the client. The authentication scheme used in Figure 16 is called MD5-Challenge and it is a predefined authentication scheme in EAP. MD5-Challenge is recommended to be supported by all EAP implementations. See [35] for more information about MD5-Challenge.

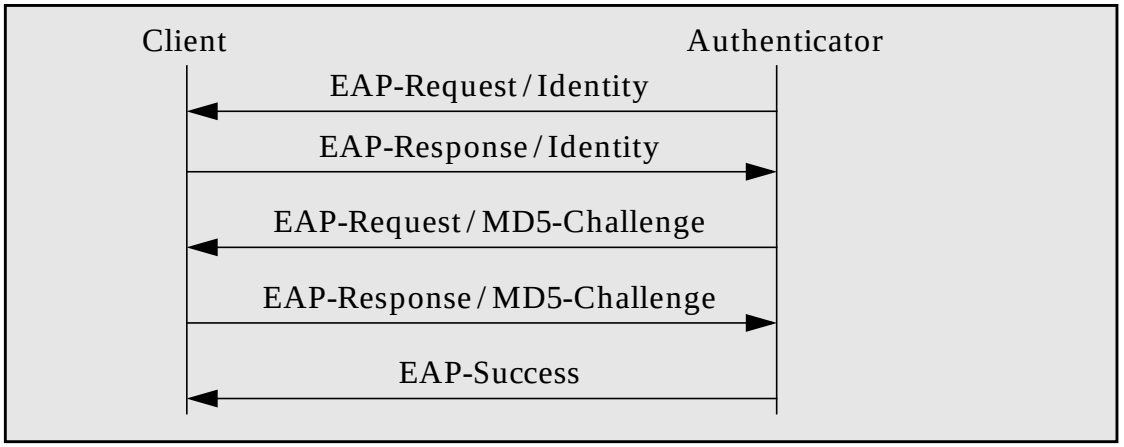

Figure 16 EAP Packet sequence during authentication.

4.2.1 EAP in SIP

EAP may be used in SIP for authentication, by extending the WWW-Authenticate, the Authorization and the Authorization-Info header field with a new authentication scheme, defined as “eap”. The challenge in the WWW-Authenticate header field consists of a realm value and an EAP request packet, the credentials in the Authorization consists of an EAP response packet and the EAP success and failure packets are sent in the Authorization-Info header field, see “HTTP Authentication” on page 21 for more information about these header fields.

Note, that some EAP responses, included in the Authorization header field, will always generate a 401 (Unauthorized) SIP response. For an example, when the SIP UA receives an EAP Identity request in a WWW-Authenticate header field, it will respond with its identity and not with its credentials. This means that the SIP authenticator is forced to respond with a 401 response, because the SIP request contains no authorized credentials.

Figure 17 shows the message flow which performs a similar authentication that were presented in Figure 16, by using EAP in SIP.

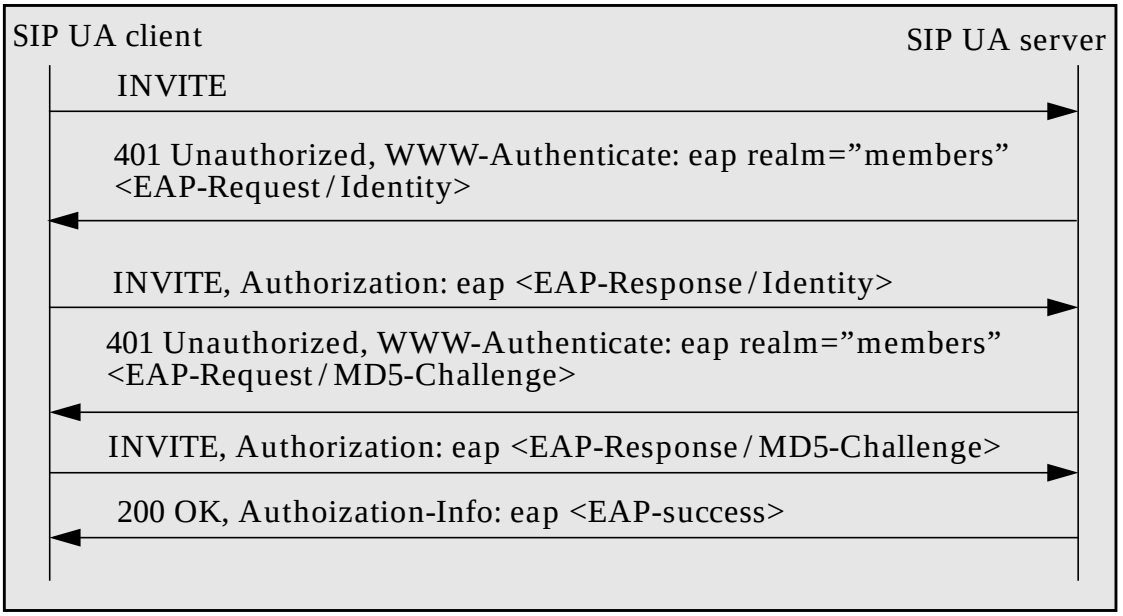

Figure 17 SIP message flow during authentication.

When the authenticator is a SIP proxy, then the SIP UA should use Proxy-Authorization instead of Authorization and the authenticator should use Proxy-Authentication instead of WWW-Authentication. EAP packets are inserted in these header fields in exactly the same way as described above.

4.2.2 UMTS AKA

The Universal Mobile Telecommunications System (UMTS) Authentication and Key Agreement (AKA [36]) procedure, defined by the 3rd Generation Partnership Project (3GPP), is intended to be used by the third generation mobile systems for mutual authentication between the User Equipment (UE) and the serving network (SN). A mechanism for exchange of symmetric session keys, used for encryption and integrity protection over the radio interface, is also provided by UMTS AKA. The documents released by 3GPP are quite extensive and hard to understand. An alternative is to study [54], which describes the UMTS AKA procedure with plenty of illustrations.

UMTS AKA resembles HTTP Digest Authentication, which is based on a challenge-response mechanism using cryptographic hashes of passwords, see "Digest Authentication" on page 23. The difference is that UMTS AKA provides mutual authentication and is based on a 128 bit shared secret, stored in the UE and in the Home Environment (HE). The HE, maintained by the UE user's operator, is responsible for enabling a user to obtain UMTS services in a consistent manner regardless of the user's location or UE used.

The SN provides the UE user with access to the services provided by the HE. Which SN the user authenticates itself to depends on the user's location, and the SN is in most cases not operated by the same operator that operates the HE. However, it is assumed that the HE trusts the SN to handle the authentication information securely. It is also assumed that the connection between the SN and the HE is adequately secure and that the UE user trusts the HE.

A detailed description how UMTS AKA is managed by different network nodes in UMTS will require a lot of background theory on how these nodes cooperates with each other. To avoid that, this section will just describe in principle how the authentication is performed, and a simplified scheme may be seen in Figure 18 on page 58.The authentication procedure is initialized by the SN, which makes an inquiry about getting one or several Authentication Vectors (AV), designated for a particular UE, from the HE. The UE is identified by its International Mobile Subscriber Identity (IMSI) which is included in the inquiry. The HE uses among others the secret key, K, and a sequence number, SQN_{HE}, to create the AVs. Each AV contains five parameters, which are listed in Table 16.

Authentication parameter	Description
RAND	Pseudo random number used to challenge the UE.
XRES	Expected response from the UE.
AUTN	Authentication token used to authenticate the SN to the UE.
CK	Symmetric key used for encryption.
IK	Symmetric key used for integrity protection.

Table 16 Defined authentication parameters in AKA.

The authentication token, AUTN, is constructed by concatenating three different values and each value is described in Table 17.

Values	Description
MAC	Message authentication code used by the UE to authenticate the SN.
AMF	Authentication and key management field. Used for disaster recovery.
$SQN_{HE} \oplus AK$	Used by the UE to reveal the sequence number used by the HE. The Anonymity Key (AK), also predictable by the UE, is exclusive or:ed with the sequence number. That operation makes it harder for an eavesdropper to reveal the sequence number.

Table 17 Concatenated values in the AUTN parameter.

The SN chooses one of the AVs, received from the HE, and sends the challenge parameter, RAND, and the authentication token, AUTN, to the UE. The UE extracts the MAC value from the AUTN parameter and calculates an own XMAC value, based on the secret key, K. If the MAC and XMAC values are equal then the SN is authenticated to the UE. Before the UE validates the AUTN parameter it compares the sequence number in it with its own sequence number, SQN_{UE}, to check if it is in a correct range. If the sequence number is valid, then the UE sends the response, RES, back to the SN to authenticate itself to the SN. The SN then compares the RES value with the stored XRES value. If both values are equal then the client device and the SN have been authenticated and they share two symmetric keys, CK and IK, that may be used for encryption and integrity protection of data over the radio interface.

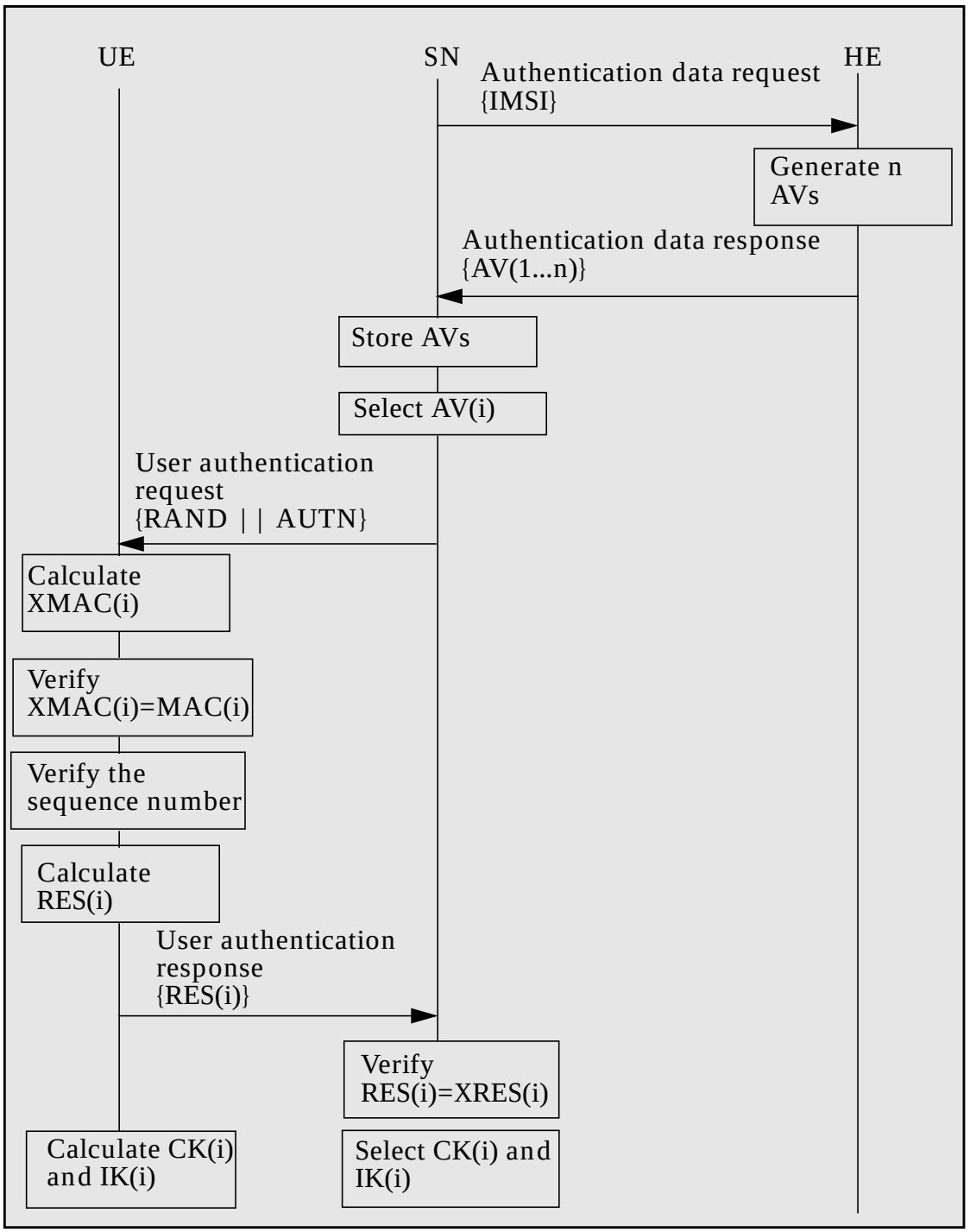

Figure 18 Simplified scheme of the UMTS AKA procedure, used for mutual authentication and key agreement.

If the UE receives a sequence number, that is not within a valid range compared to its own sequence number, it has to cancel the authentication procedure and resynchronize the sequence number stored at the HE. When the resynchronization is done, the authentication procedure is restarted and new AVs are generated by the HE. More information about the resynchronization procedure may be found in [37].

All authentication parameters, described above, are generated by the HE and the UE, which use eight different functions for that purpose, namely f0-f5, f1*, f5*. UMTS AKA also uses two other functions used for the encryption and the integrity protection of the data over the radio interface, which are called f8 and f9. The functions used for generating the authentication parameters are all operator specific, but 3GPP has stated some requirements that these functions must fulfil. This document will not cover these requirements nor how these functions are used for generating the authentication parameters. A detailed description may be found in [37]. The f8 and f9 functions are not operator specific and both these functions are specified in [38].

4.2.3 SIP/EAP/IMS AKA

IMS AKA[1] may be used in EAP, which in its turn may be used in SIP. 3GPP will use SIP as signaling protocol in the IP Multimedia Core Network Sub-System (IM CN SS [39]) part of UMTS. Therefore, 3GPP will benefit if IMS AKA will be used for authentication in SIP instead of the ordinary HTTP Authentication. The reasons are:

- IMS AKA is supported in the old GSM systems
- IMS AKA is developed and well-known by 3GPP
- IMS AKA solves many of the security flaws that HTTP Digest has
- IMS AKA is designed to be used by mobile equipment
- IMS AKA supports delegation of the authentication procedure to other entities

But some drawbacks exist:

- use of IMS AKA will increase the size of the SIP messages compared to the size of the SIP messages when HTTP Authentication is used.
- IMS AKA is not developed by IETF, i.e. it may be hard to convince IETF to use an authentication mechanism which they have not developed by themselves.

There is a pending draft on how to use IMS AKA in EAP [40], which also gives a brief overview of the AKA procedure. This document will only describe the new EAP Type fields values defined in the draft and not how the EAP packets are constructed for every possible scenario. The draft covers that information very well.

The new EAP Type field values defined are listed in Table 18[2].

1. To distinguish between the AKA procedure used in UMTS, 3GPP has chosen to call it IMS AKA.
2. The UMTS Subscriber Identity Module (USIM) acronym which is included in all Type field names is an application, which may be implemented on a smart card. This application is used for storing information about the subscriber, e.g. the secret key and the IMSI

Type field value	Description
USIM-Challenge	Requests of this type contain the authentication token, AUTN, and the challenge, RAND, parameter and responses of this type contain the response, RES, parameters.
USIM-Authentication-Reject	Only used for responses and is sent by the client when it has received an USIM-Challenge request, which contains an invalid AUTN parameter.
USIM-Synchronization-Failure	Only used for responses and is sent by the client when it has received an USIM-Challenge request, which contains an invalid sequence number.
USIM-IMSI	Requests of this type are sent when the authenticator does not know the International Mobile Subscriber Identity (IMSI) or the Temporary Mobile Subscriber Identity (TMSI) of the client. Responses of this type contain the IMSI in clear text. The IMSI should be considered as a secret, which means that responses and requests of this type should be avoided.

Table 18 New EAP requests and responses defined for IMS AKA.

As mentioned before in “UMTS AKA” on page 55, the IMSI which uniquely identifies the client should be kept secret to avoid revealing the true identity of the client. When the authenticator sends an EAP request with Identity as Type field value, the client should instead use the Temporary Mobile Subscriber Identity (TMSI) to identify itself. The TMSI is not generated by the client but is generated by the current authenticator or a previous authenticator and then distributed to the client. If the authenticator does not recognize the received TMSI, it has to request the client to send the IMSI in clear before it can generate a TMSI. Further information how the TMSI is generated and distributed may be found in [36].

The encryption and integrity keys generated during the IMS AKA procedure are intended to be used for providing confidentiality and integrity protection for messages sent after the authentication is completed. Unfortunately, the encryption and the integrity protection may only be provided between the first and the second SIP entity in the signaling path, because intermediate SIP entities must be able to read and modify SIP messages. The tunneling methods, described in “Tunneling integrity and authentication” on page 48 and

“Tunneling encryption” on page 51, may be used in a similar way to provide limited encryption and integrity protection end-to-end. Instead of using S/MIME, the algorithms described in the AKA documentation [38] should be used. It is important to mention, that there exists no documentation on how to use these keys in SIP.

Figure 19 shows an example how IMS AKA in EAP may be used in SIP. The SIP UA server in the example has never communicated with the SIP UA client. Therefore, it does not recognize the TMSI value, that the SIP UA client has used during a previous authentication, and has to request the SIP UA client to send the IMSI in clear. When the SIP UA server receives the IMSI, it creates the TMSI and includes it in the USIM-Challenge request, sent to the SIP UA client. The client removes the old TMSI and replaces it with the received one. The SIP UA server i Figure 19 is assumed to have one or more authentication vectors (AVs) for the particular SIP UA client. In reality, the SIP UA server has to ask some type of authentication server located in the SIP UA client’s Home Environment for the AVs.

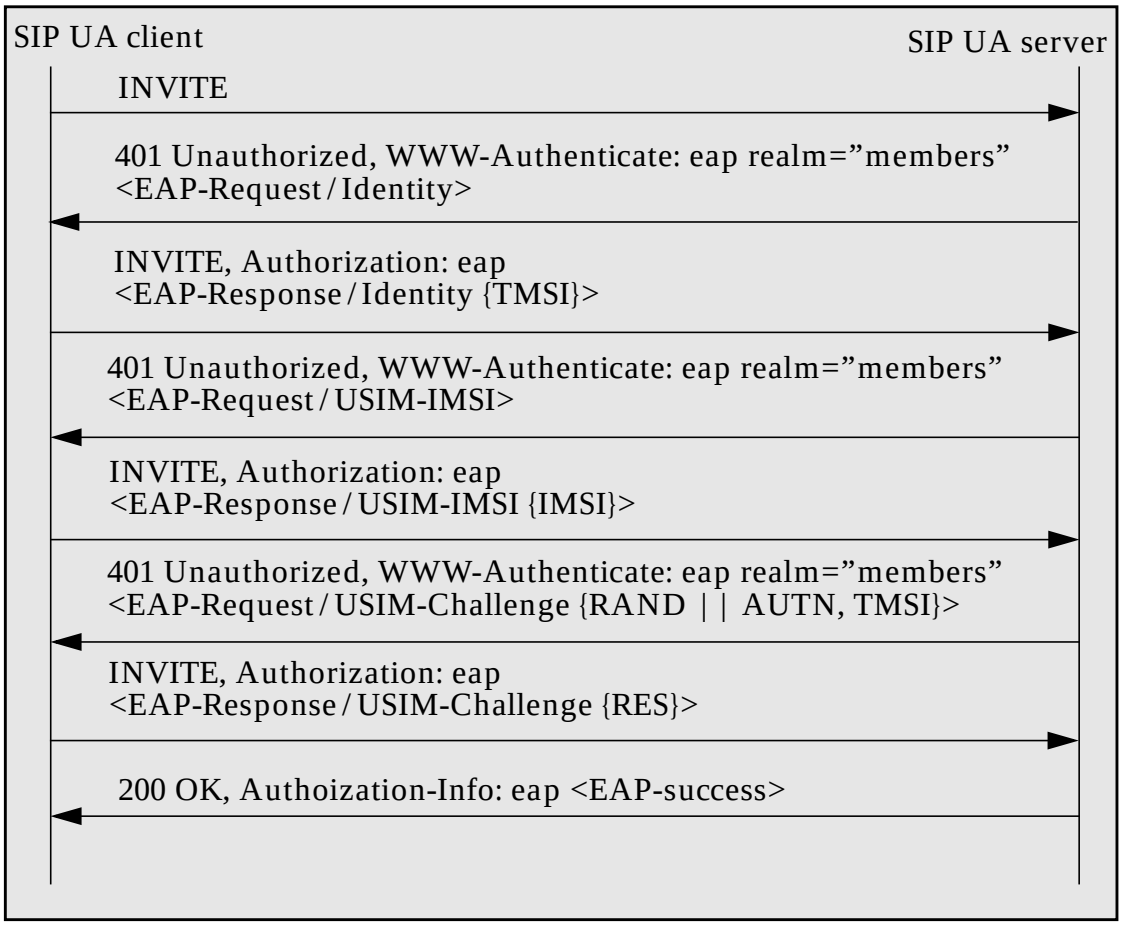

Figure 19 Example of the IMS AKA procedure used with EAP in SIP.

4.3 Transport and network layer security

This section covers security protocols that may be used to support the security mechanisms in SIP. These protocols are implemented in the transportation layer or in the network layer.

4.3.1 TLS

Transport Layer Security (TLS [5]) is implemented above the Transmission Control Protocol (TCP [56]) and is designed to provide reliable end-to-end security. The limitation of only using TLS above TCP is due to the reliability requirement, which the User Datagram Protocol (UDP [55]) does not offer.

TLS is based on the Secure Socket Layer (SSL [57]), which was developed by Netscape and integrated in their web browser, and is developed by IETF. Actually, TLS consists of four different sub-protocols to provide its security service and Figure 20 shows how these protocols are organized in the TLS protocol stack.

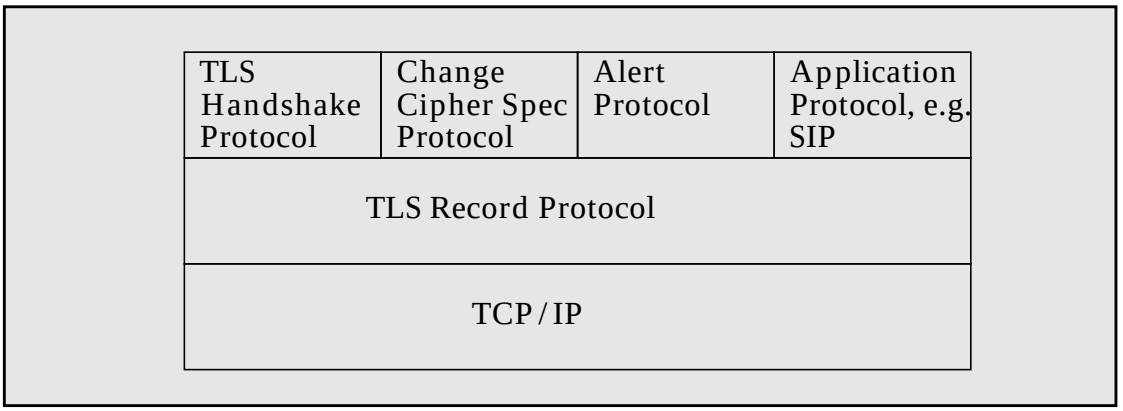

Figure 20 Illustration how the TLS Handshake, Change Cipher Spec, Alert and Record protocols are organized in the TLS protocol stack.

Each sub-protocol in Figure 20 has following tasks:

Sub-protocol	Tasks
TLS Record Protocol	Provides confidentiality and integrity protection services. It also handles the fragmentation and compression of the application data.
TLS Handshake Protocol	Defines how to negotiate an encryption and a secure hash function and cryptographic keys to be used to protect the data.

Sub-protocol	Tasks
Change Cipher Spec Protocol	Consist of a single message, which consists of a single byte. This message is always sent, by both entities, during TLS connection initiation to notify the other entity that the negotiated chipher suit should be used after this message is received.
Alert Protocol	Used to send TLS related alerts to the other entity. Each alert can be of fatal type or warning type. If a fatal alert is received, then should the TLS connection be terminated immediately.

Table 19 Description of each sub-protocol used in TLS.

This document will only give a detailed description of the TLS Handshake protocol, because a detailed description of the other sub-protocol would only be interesting for implementators.

4.3.1.1 TLS Handshake Protocol

The TLS Handshake Protocol is the most complex and interesting part of TLS and it defines ten messages used by the client and the server to authenticate each other, negotiate a secure hash function and an encryption algorithm and provide each other with secret data for generating cryptographic keys. The cryptographic keys are used with the negotiated algorithms by the TLS Record Protocol to provided encryption and integrity protection.

The message flow sent during the handshake is shown in Figure 21 [5] and the message flow can be divided in four phases[1]: Phase 1. Establish security capabilities, Phase 2. Server authentication and key exchange, Phase 3. Client authentication and key exchange and Phase 4. Finish.

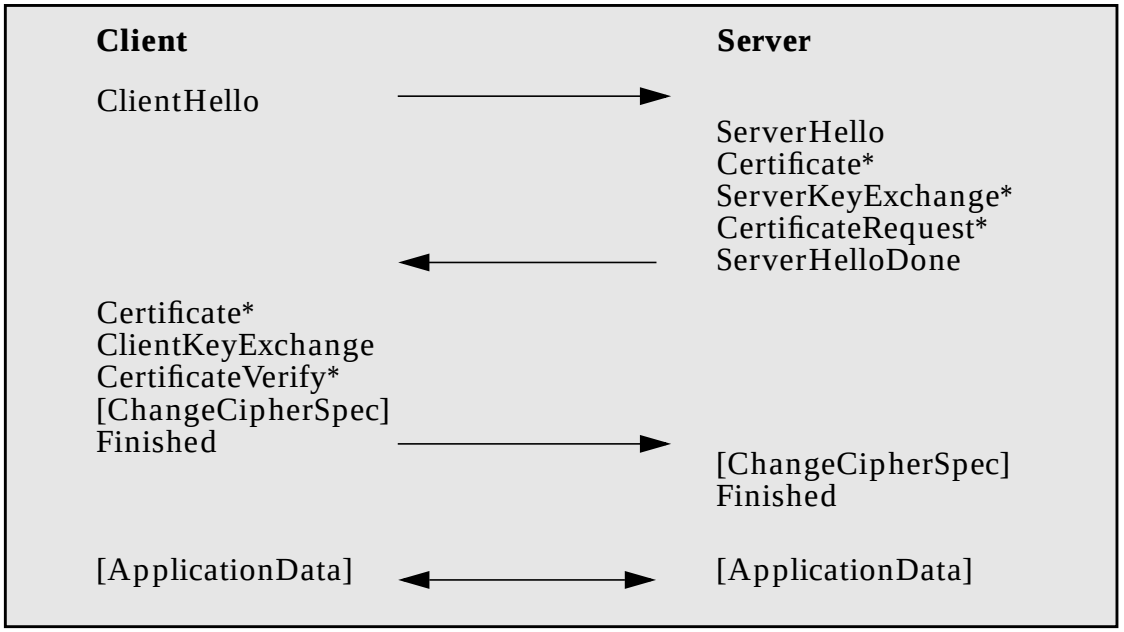

Figure 21 Message flow for a full handshake[1].

Phase 1. Establish security capabilities

The handshake is initiated by the client, who sends a ClientHello message. The ClientHello message consist of five parameters, which is described in Table 20.

Parameter name	Description
ProtocolVersion	The version of the TLS protocol that the client will use during the TLS session.
Random	A nonce generated by the client, which consist of a random part and a timestamp. The random part should be generated by a secure random generator.
SessionID	Variable-length session identifier, which is set to zero if the client wants to create a new connection on a new session. If the client wants to resume a session, it specifies a non-zero identifier for that session. This document will not cover how to resume a session, see [5] for that topic.

1. Messages marked with * are optional or sent depending on the situation. Messages marked between brackets are sent by other protocols.

Parameter name	Description
CipherSuit	This is a list of the cryptographic options supported by the client. The client specifies the preferred element first. Each element specifies a key exchange algorithm and a description of the algorithm used for encryption and integrity protection.
CompressionMethod	This is a list of the compression methods supported by the client. This document will not describe how the compression service is used.

Table 20 Parameters for the ClientHello message.

When the server receives the ClientHello message, it chooses one of the client's proposed cipher suit and compression method and inserts them into a ServerHello message. The ServerHello message has the same parameters names as the ClientHello message has. Beside the cipher suit and compression method, the server creates a session identifier, if the SessionID parameters in the ClientHello message was set to zero, and a unique Random value. The server then sends the ServerHello message to the client. Now, both sides has agreed to use a key exchange algorithm, symmetric encryption algorithm and a secure hash function.

Both the TLS Handshake Protocol and the TLS Record Protocol uses the negotiated secure hash function, which may be either MD5 or SHA-1, with the HMAC algorithm [58] to provide integrity protection for messages. The HMAC algorithm may be seen as a secure hash function that uses a secret key to authenticate the message. Figure 22 shows how the HMAC code, which is the output from the algorithm, is produced.

$$HMAC_{hash}(K, M) = hash(K^* \oplus opad \,\|\, hash(K^* \oplus ipad \,\|\, M))$$

where

K^*	=	Secret key padded with zeros so that its size is equal to the block length of the secure hash function.
hash	=	Secure hash function, MD5 or SHA-1.
opad	=	00110110 repeated 64 times.
ipad	=	01011100 repeated 64 times.
M	=	The message inputted to the HMAC algorithm.

Figure 22 Calculation of the HMAC code.

Table 21 lists symmetric encryption algorithms that may be used in TLS. The block ciphers are always running in Cipher Block Chaining mode, which means that before a block is encrypted it is exclusive or:ed with the encrypted

data from the previous block and the first block is exclusive or:ed with an initialization value. The CBC mode is used to hide certain patterns in messages.

Algorithm	Key size	Type
IDEA	128	Block cipher
RC2-40	40	Block cipher
DES-40	40	Block cipher
DES	56	Block cipher
triple DES	168	Block cipher
RC4-40	40	Stream cipher
RC4-128	128	Stream cipher

Table 21 Symmetric encryption algorithms used in TLS.

Phase 2. Server authentication and key exchange

If the server chose to authenticate itself to the client is sends a public key certificate in a Certificate message. Table 22 shows that the server is required to send the Certificate message in all cases except when Anonymous Diffie-Hellman is used as key exchange algorithm.

Before the client and the server will be able to use the negotiated symmetric encryption algorithm, they must exchange the secret key by using one of the key exchange algorithm described in Table 22. In reality, the entities exchanges a so-called premaster secret, which is used by both entities to create the secret keys for encryption and the HMAC calculations.

Key exchange algorithm	Description
RSA key exchange	• The server sends its public key certificate to the client in a Certificate message. The public key certificate contains the server's RSA public key and is signed by a CA. • The client verifies the public key certificate and uses the RSA public key to encrypt the premaster secret. The encrypted premaster secret is sent to the server in a ClientKeyExchange message.

Key exchange algorithm	Description
Ephemeral Diffie-Hellman	• The server sends its public key certificate to the client in a Certificate message. The public key certificate contains the server's RSA public key and is signed by a CA. • The server generates a Diffie-Hellman public key by using valid Diffie-Hellman parameters. The public key and the parameters is signed with the server's private key and sent to the client in a ServerKeyExchange message. • The client verifies the public key certificate, verifies the signature in the ServerKeyExchange message and generates its Diffie-Hellman public key, by using the parameters in the message. • The client sends its Diffie-Hellman public key to the server in a ClientKeyExchange message. • Both entities obtains the premaster secret by using the Diffie-Hellman algorithm.
Diffie-Hellman	• The server sends its public key certificate to the client in a Certificate message. The public key certificate contains the server's Diffie-Hellman public key and is signed by a CA. • The client verifies signature of the received public key certificate. • The client send its Diffie-Hellman public key in a Certificate message or in a ClientKeyExchange message. • Both entities obtains the premaster secret by using the Diffie-Hellman algorithm.
Anonymous Diffie-Hellman	• No public key certificates is used. • The server sends its Diffie-Hellman public key to the client in a ServerKeyExchange messgage. • The client sends its Diffie-Hellman public key to the server in a ClientKeyExchange message. • Both entities obtains the premaster secret by using the Diffie-Hellman algorithm.

Table 22 Description of key exchange algorithms used in TLS[1].

The CertificateRequest message that is optionally sent by the server to request a public key certificate from the client. The message contains two lists. The

1. There exists on more key exchange algorithm, which uses a temporary RSA key-pair to encrypt the premaster secret. More information about this algorithm may be found in [5].

first list specifies which types of public key certificates the server will accept and the second list specifies which CAs the server will accept.

Next message sent by the server is the ServerHelloDone message. Its function is only to tell the client that the server has sent all messages associated with the ServerHello message. The server will then wait for a client response.

Phase 3. Client authentication and key exchange

If the server has requested the client to authenticate itself to the server, the client is required to send a valid public key certificate in a Certificate message. If the client does not has a public key certificate that fulfils the requirements sent by the server in the CertificateRequest message, it sends an alert message to the server. The server may then decide if the handshake should continue or not.

The ClientKeyExchange message is always sent by the client and its purpose is to complete the exchange of the premaster secret. Table 22 on page 67 describes how this messages is used in different key exchange algorithms. Compared to the ServerKeyExchange message, this message is not signed.

The last message, CertificateVerify message, in this phase is optionally sent by the client to prove its ownership of the public key sent in a earlier Certificate message. The message consist of either a signed MD5 or SHA-1 hash of all messages sent or received during the handshake.

Phase 4. Finish

At this stage, both entities knows the premaster secret and is able to calculate the master secret. Following definitions is needed for the master secret calculation:

$$P_{hash}(secret, seed) = HMAC_{hash}(secret, A(1) \| seed) \| HMAC_{hash}(secret, A(2) \| seed) \| HMAC_{hash}(secret, A(3) \| seed) \ldots$$

where the recursive function, $A(i)$, is defined as

$$A(0) = seed$$
$$A(i) = HMAC_{hash}(secret, A(i-1))$$

and

hash	=	Secure hash function, MD5 or SHA-1.
seed	=	Initial value.

Figure 23 Definition of the expansion function.

TLS uses the expansion function, P_{hash}, defined above in its pseudorandom function, PRF. The PRF is defined in Figure 24.

$$PRF(secret, label, seed) = P_{MD5}\left(secret\left[0\ldots\left\lfloor\frac{n}{2}\right\rfloor\right], label \| seed\right) \oplus P_{SHA-1}\left(secret\left[\left\lceil\frac{n}{2}\right\rceil\ldots n-1\right], label \| seed\right)$$

where

n	=	The length of the secret in bits.
$secret[k_1\ldots k_2]$	=	The k_1 to k_2 bits of the secret.
$label$	=	An ASCII string.

Figure 24 Definition of the pseudorandom function, PRF.

The master secret is then calculated by using the PRF:

master secret =
PRF(premaster secret, ″master secret″, ClientHello.random ‖ ServerHello.Random)

Figure 25 Definition of the master secret.

When both the entities have calculated the master secret they use it to calculate the symmetric encryption key, the HMAC key and the initializing value for the encryption algorithm (if it runs in CBC mode). For an exact description how to calculate these keys and the initializing value, see [5].

When the client has calculated all keys, it sends the ChangeCipherSpec message to the server. This means that all messages, sent by the client, will now be encrypted and integrity protected by the TLS Record Protocol. The client then sends the Finish message, which is the last message sent by the client during the handshake. The Finish message verifies that the key exchange and the authentication processed were successful and it contains a hash value of all messages sent during the handshake, except the Finish message.

When the server receives the ChangeCipherSpec message from the client, it sends a ChangeCipherSpec message to the client. As for the client, the server now encrypt and integrity protect all messages. The server then ends the handshake by sending a Finish message to the client. As mentioned before, the Finish message contains a hash value of previous handshake messages and it is defined in Figure 26.

hash value = PRF(master secret, finish label, $MD5$(handshake messages) ‖ $SHA-1$(handshake messages))

where

finish label	=	For the client this ASCII string is set to “client finished” and the server sets it to “server finished”.
handshake messages	=	All messages sent during the handshake, except the Finish message.

Figure 26 Definition of the hash value in the Finish message.

4.3.1.2 TLS in SIP

IETF has proposed that TLS may be used in conjunction with SIP to provide security services on hop-to-hop basis, i.e. multiple TLS connections are established through the signaling path. Figure 27 shows an example of a signaling path between two SIP UAs, which is partially protected with TLS.

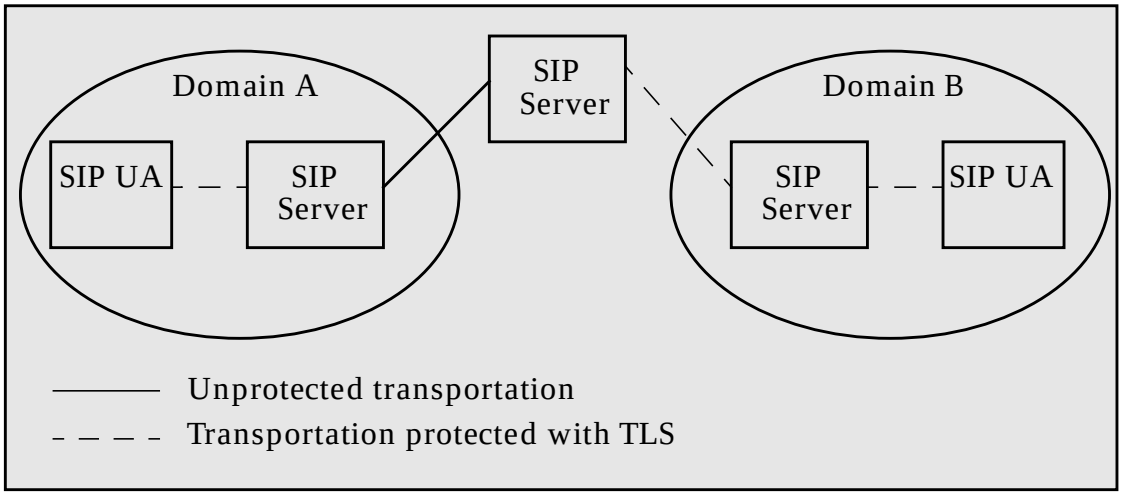

Figure 27 Example of where TLS is used between two SIP UAs in different domains.

IETF has stated following requirements on how TLS should be used in SIP:

- All SIP servers must implement TLS and is required to support mutual and one-way authentication.
- The SIP UA is recommended to be able to initiate a TLS connection and it may act as a TLS server.
- SIP servers should have a public key certificate, which subject correspond to their hostname.
- The SIP UAs may have a public key certificate. The public key certificate is used if the TLS server sends a CertificateRequest message to the SIP UA.
- Both SIP UAs and SIP severs should be able to verify the public key certificate, i.e. the signature is valid and the subject corresponds to a valid entity.

- When a SIP UA attempts to connect to a SIP server, it should establish a TLS connection over which SIP messages are sent.
- The SIP UA should be able to use same mechanism to validate public key certificate for S/MIME as it does for TLS.

The rest of this section will cover three different scenarios [59] how TLS may be used in SIP.

Registration

Figure 28 shows the message flow, between the SIP UA and the registrar, during a typical registration process.

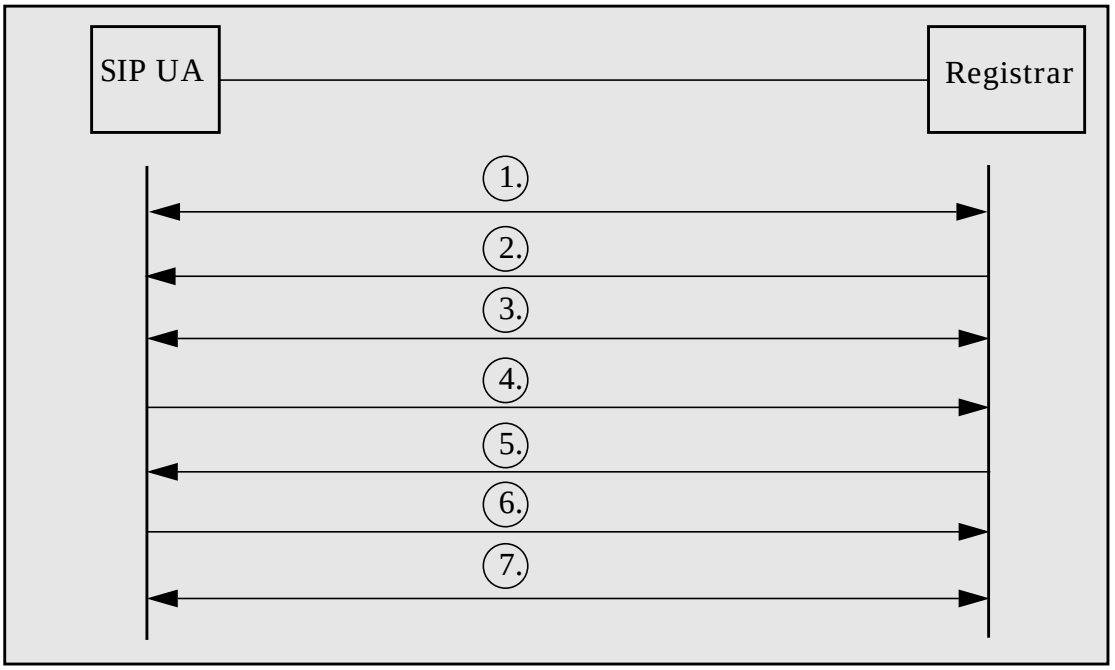

Figure 28 SIP UA registration scenario.

Each step in Figure 28 is explained beneath:

1. The SIP UA initiates the TLS connection and the Hello messages are sent.
2. The registrar sends it public key certificate to the SIP UA, which verifies the signature and checks if the subject match the correct hostname. If the public key certificate verification fails, then the SIP UA must terminate the registration.
3. The SIP entities continue the TLS handshake and calculates the necessary keys.
4. The SIP UA send the REGISTER message over the secured link to the registrar.
5. The registrar requires that the SIP UA authenticates itself with Digest authentication. Thus, it sends a 401 (Unauthorized) response which contains the challenge.
6. The SIP UA inserts its credentials in a new REGISTER message and sends it to the registrar.
7. The registrar accepts the credentials as valid and both entities completes the registration. The TLS connection may remain open after the registration for future message exchanges between the entities, i.e. the registrar may act as a SIP proxy.

Interdomain invitation

Figure 29 shows a typical scenario where a SIP UA invites another SIP UA in a different domain by using TLS between each hop in the signaling path. The SIP proxy in domain A acts as the outbound proxy for the SIP UA in domain A and the SIP proxy in domain B acts as the outbound proxy for the SIP UA in domain B. To simplify the scenario, both proxies also handles inbound request, i.e. the SIP UAs will both send and receive SIP messages over the same TLS connection.

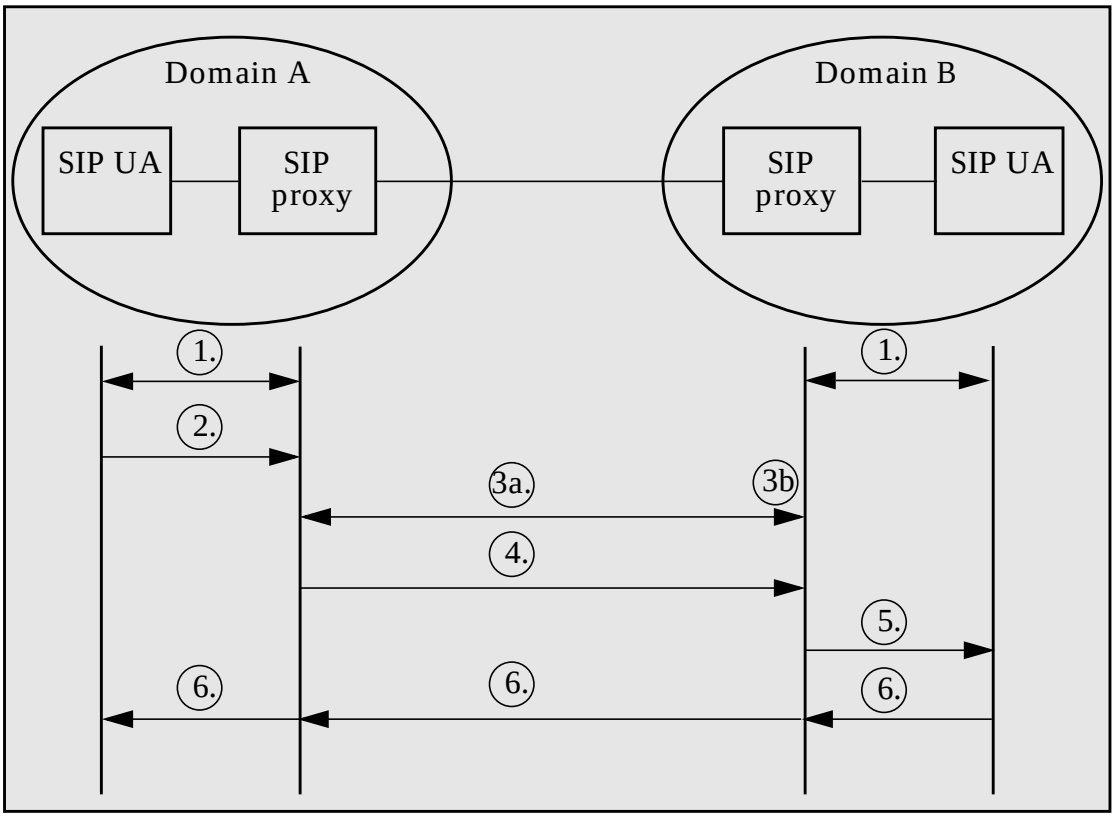

Figure 29 SIP UA invitation scenario between two SIP UAs in different domains.

Each step in Figure 29 is explained beneath:

1. Both SIP UAs establish a TLS connection with their outbound proxy and authenticate themselves with Digest authentication. If the outbound proxy has the same hostname as the registrar, then the SIP UA reuses the TLS connection used during the registration and the credentials used during the Digest authentication.
2. The SIP UA in domain A sends a INVITE request to its outbound proxy over the secured link. The INVITE requests contains proper credentials.

3a. The outbound proxy in domain A examines the INVITE and determines that the message should be forwarded to the outbound proxy in domain B. There exists no TLS connection between the two outbound proxies. Thus, the outbound proxies establish a TLS connection and both entities exchange their public key certificates for mutual authentication, both entities send the Certificate message during the handshake.

3b. The SIP proxy in domain B may have different policies how it handles inbound requests. An example of a policy is that the SIP proxy only allows requests with same domain name in the From header field as was specified in the received public key certificate from the outbound proxy in domain A. This means that the requests sent by the SIP UA in domain A in Figure 29 is allowed, but requests from the SIP UA in domain A in Figure 27 on page 70 is not.

4. If the validation of the public key certificates did not fail, then the outbound proxy in domain A forwards the INVITE request over the secured link.

5. The outbound proxy in domain B has already an open TLS connection with the SIP UA in domain B. Thus, it forwards the INVITE request to the SIP UA over the secured link.

6. Both outbound proxies has added a Record-Route header field in the INVITE message. Thus, the response from the SIP UA in domain B will pass through the same proxies.

Peer-to-Peer invitation

Figure 30 shows a typical scenario where an SIP UA, which does not use an outbound proxy, invites another SIP UA in domain B. The SIP proxy in domain B acts like a redirect server for inbound requests from non-authenticated SIP UAs, i.e. the two SIP UA will be connected peer-to-peer.

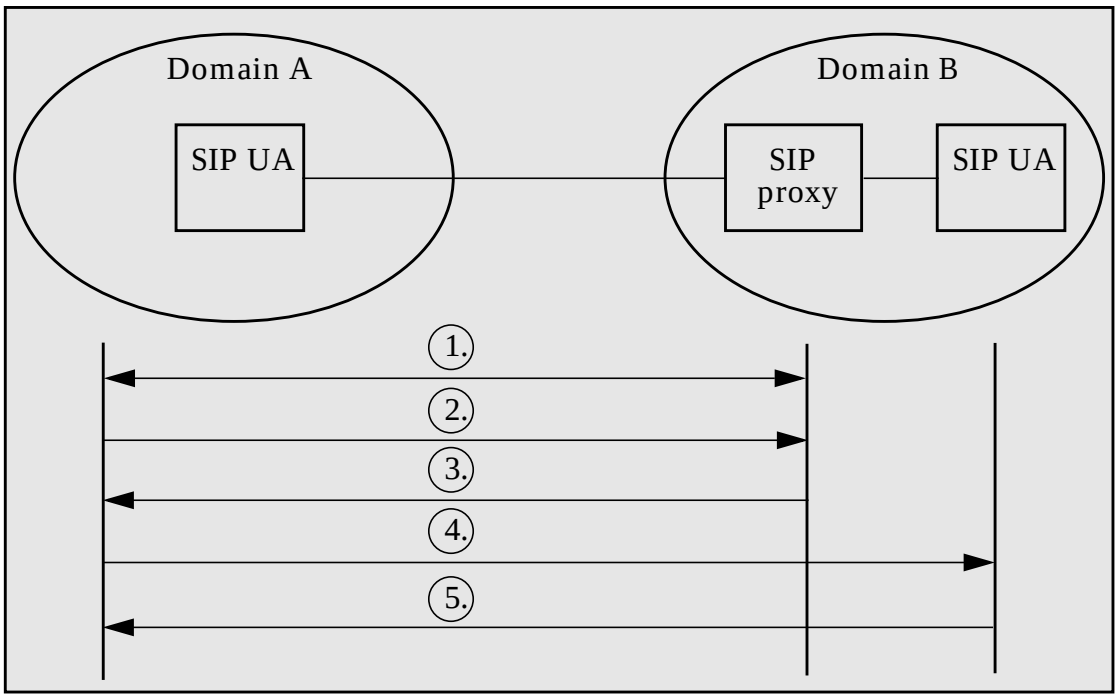

Figure 30 Scenario where an SIP UA invites another SIP UA peer-to-peer.

1. The SIP UA in domain A establish a TLS connection with the SIP proxy in domain B. The SIP proxy sends its public key certificate in a Certificate message, which the SIP UA validates.

2. The SIP UA sends an INVITE request addressed to the SIP UA in domain B. The INVITE request is clear signed, see “Clear signing” on page 43 and “Tunneling

integrity and authentication” on page 48, by the SIP UA to provide some limited authentication.

3. The SIP proxy receives the INVITE request over the TLS connection and treats the SIP UA as unauthorized. Therefore, it sends a redirect response back to the SIP UA in domain A, which contains a Contact header field that specifies an address directly to the SIP UA in domain B.
4. The SIP UA in domain A accepts the redirect response, because it was sent over a secure TLS connection. Then it calculates a new signature and sends the INVITE request to the address specified in the Contact header field of the redirect response. Note, that the INVITE requests is sent over an insecure connection to the SIP UA in domain B.
5. The SIP UA in domain B receives the INVITE request and examines the included S/MIME public key certificate. In this case, the SIP UA has a previously cached S/MIME public key certificate that match the received one. Thus, it authorizes the SIP UA in domain A. The SIP UA in domain B continues the invitation by sending a response back to the SIP UA in domain A. The SIP UA in domain B clear signs the response to authenticate itself to the SIP UA in domain A.

4.3.2 IPSec

IP Security (IPSec [6]) is an extension to both the version 4 and 6 of the IP protocol and is meant to provide security services at the network layer. This section will only give a brief description of IPSec due to the protocols complexity and that SIP is completely independent of it, i.e. a SIP UA will not notice if IPSec is used or not. Even if SIP implementations are independent of IPSec, IPSec may be a successful solution for protecting SIP messages. Especially, intermediate SIP servers may use IPSec for establishing secure long-lived connections between each other, e.g. between two outbound proxies in different domains.

Two protocols are used within IPSec to provide security services: the Authentication Header (AH) and the Encapsulating Security Payload (ESP). The ESP protocol may be configured to provide encryption or both encryption and authentication and the AH protocol only provides authentication. These protocols may be used to provide some or all of following services [1]:

- Access control
- Connectionless integrity
- Data origin authentication
- Rejection of replayed packets
- Confidentiality
- Limited traffic flow confidentiality

To be able to use IPSec, the sender is needed to have a Security Association (SA) with the receiver. The receiver must also have a SA with the sender, if IPSec should be used in both directions, i.e. the SA is a one-way relationship. A SA is defined by its parameters, see [6], and is uniquely identified by a Security Parameters Index (SPI) and the IP Destination Address. The SA parame-

ters give among other things whether ESP or AH should be used for the SA, because both protocols can not be used for same SA. The concept of using SAs is very important in IPSec and more information may be found in [6] and [1].

Both ESP and AH may be used in two different modes: transportation and tunnel mode. The transportation mode may be seen as an extension to the IP packet, where IPSec inserts either an AH or ESP header to the original IP packet and then applies the chosen security mechanism on certain parts of the original IP packet. The tunnel mode makes more drastic changes to the original IP packet, where IPSec encapsulates the original IP packet in the payload of a new IP packet, adds a new IP header, inserts an AH or ESP header to the new IP packet, and then applies the chosen security mechanism on the whole original IP packet. Which mode to use depends on the situation, but tunnel mode is most suitable to be used by intermediate network entities to protect the whole original IP packet and transport mode is most suitable to be used between end users. More information about how these different modes may be used and how the packets are defined and constructed may be found in [1], [60], [61].

The last fundamental part of IPSec is the key management, which is needed before confidentiality and integrity protection services can be provided. The key management may be handled manually or automatically. Manual key management is practical in small and static environments where a relative small number of SAs is used. Automated key management is required in large environments where a large number of SAs is used. IPSec is by default using the Internet Key Exchange (IKE [64]) for automated key management, which is a realization of the Internet Security Association and Key Management Protocol / Oakley Key Determination Protocol (ISAKMP [63] / Oakley [62]). This is the most complex part of IPSec and a description of these protocols in this document will be left out due to this.

4.4 3GPP security model for SIP

This section will give a summary of the requirements on SIP that 3GPP has specified in [51]. The requirements cover most part of the SIP standard, but this section will focus on the 3GPP security model for SIP. However, some background information on how 3GPP is intended to use SIP is also covered.

As mentioned in “SIP / EAP / IMS AKA” on page 59, 3GPP will use SIP for signaling in their IP Multimedia Core Network SubSystem (IM CN SS). Unlike IETF, 3GPP is more specific how different SIP entities should be used in their networks. IETF only lists different types of SIP entities and specifies what they are supposed to do and not how they are connected to each other.

For a particular SIP UA, the IM CN SS is decomposed in a Home Network (HN) and a Visited Network (VN). Parallels may be drawn to the Serving Network (SN) and the Home Environment (HE), described in “UMTS AKA” on page 55. The SIP UA uses the General Packet Radio Service (GPRS [49]) for transportation of IP packets and uses it to obtain an IP version 6 (IPv6 [50]) address. A broad overview how SIP entities are organized in the IM CN SS may be seen in Figure 31.

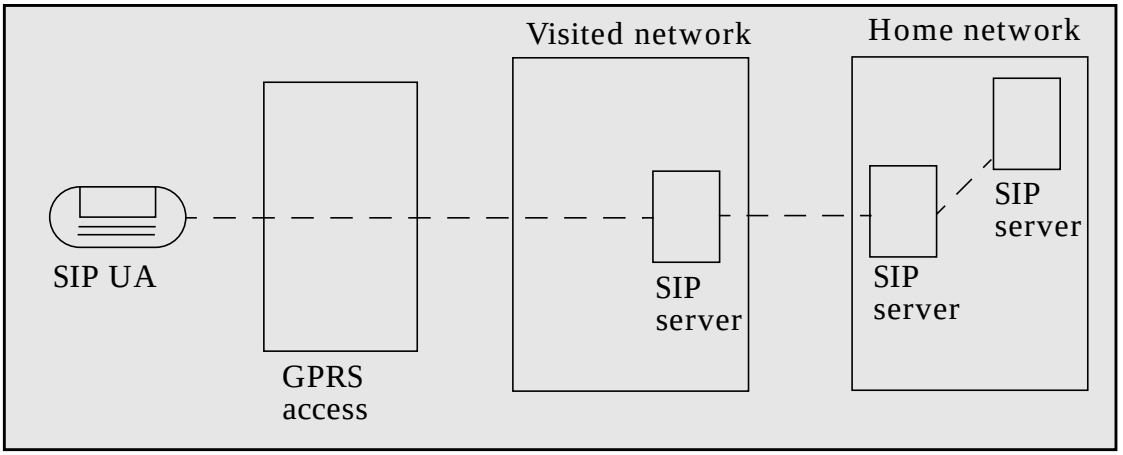

Figure 31 Overview of the IM CN SS architecture.

If a SIP UA is unable to connect directly to its HN it tries to find the outbound proxy of the VN. It is unlikely that the SIP UA knows the address of the outbound proxy for the VN in advance. To find the address it may use the GPRS access or some other service like the Dynamic Host Configuration Protocol (DHCP [52]).

When the SIP UA has found the address of the outbound proxy, it has to make a registration of its location by sending a REGISTER request to the HN. The request is sent through the outbound proxy and forwarded to a so-called inbound proxy in the HN. When the inbound proxy receives the request, it chooses a suitable registrar, in the HN, to forward the request to. The registrar will not accept the request before the SIP UA is authenticated. The registrar initiates the AKA procedure by requesting one or several authentication vectors (AVs) from a suitable entity in the HN. The registrar chooses one of the AVs and encapsulates the AUTN and RAND parameters within an EAP packet. The EAP packet is inserted in a 401 (Unauthorized) SIP response and then the registrar sends the response to the outbound proxy. Note, the EAP packet has the Type field set to USIM-Challenge and both the integrity key and the encryption key is appended as parameters to the WWW-Authentication header field. The outbound proxy removes the integrity key and the encryption key from the response and forwards it to the SIP UA. The SIP UA uses the information in the EAP packet to authenticate the HN and then it sends a new REGISTER request that contains the response to the challenge. See “UMTS AKA” on page 55 for more information on how the SIP UA validates the challenge from the HN, and how it generates the response to the challenge. The new request is forwarded to the HN which authenticates the SIP UA if the response to the challenge is valid. If the response is valid, then the registrar registers the SIP UA and sends a 200 (OK) response back to the SIP UA.

It is important to mention that the description of the authentication procedure above is quite broad and many details have been removed. But it should give enough knowledge on how the HN and the SIP UA performs the mutual authentication and how the outbound proxy receives the integrity key and the

encryption key. These keys may be used for protecting future messages between the SIP UA and the outbound proxy.

The HN itself and the link between the VN and the HN forms a zone, which is called the network domain. The zone between the outbound proxy and the SIP UA is called access domain. As mentioned before, messages sent over the access domain will be protected by using the integrity key and the encryption key with suitable algorithms. The outbound proxy and the SIP UA need to negotiate which algorithm they should use. No specific algorithm for encryption and integrity protection is mentioned in the 3GPP requirements.

Careful readers may have noticed that both keys sent during the authentication procedure were sent in clear in the SIP message through the network domain. As a consequence, SIP messages sent through the network domain need to be protected. 3GPP will use IPSec together with the Internet Key Exchange (IKE [53]) for that purpose. Further, 3GPP requires that only ESP in tunnel mode should be used and ESP is preferred to be used with encryption and authentication. IKE is used to negotiate, establish and maintain ESP security associations between entities. The link between the HN and the VN is the most vulnerable part of the network domain and 3GPP makes it mandatory to implement an ESP tunnel over this link. Between entities inside the HN, the operator has a choice of implementing ESP tunnels or not. This means that the integrity key and the encryption key may be revealed during the registration inside the HN if no further security mechanism is provided by underlying domains to the IM CN SS.

To strengthen the security in IM CN SS, operates may setup packet filtering entities at the border of their networks, e.g. a firewall. These entities should reject unsolicited traffic, but must at least allow IKE and IPSec traffic to pass through.

Operators may also implement hiding capabilities in their outbound proxies to prevent unauthorized entities to reveal information about their networks. The proxy may remove certain header fields in the SIP message when it forwards a request, and when it receives a response to the request it reinserts the header fields. Other entities in the signaling path will then be incapable of revealing any information of which proxies the SIP message has passed and will only be able to see the address of the outbound proxy.

4.5 Analysis

4.5.1 S/MIME vs. PGP

As mentioned in the chapter “Security mechanisms in SIP” on page 21, the Pretty Good Privacy (PGP [20]) mechanism for message authentication and encryption has been excluded in the latest draft for the next version of the SIP standard. Instead, IETF advocates usage of S/MIME for encryption and message authentication.

Both PGP and S/MIME were developed to solve the problem of interception and forgery of e-mail. Therefore, they are quite similar in their structure and they use almost the same algorithms for providing their security services. Both

use some sort of certificates for the public key management. However, the public key infrastructure (PKI) in S/MIME is considered to be more specified and centralized, with its use of Certificate Authorities (CA), while the PKI in PGP is more decentralized with each user acting as a small CA, trusting other small CAs.

The following questions may arise:

- Why did IETF change their minds to opt S/MIME instead of PGP?
- And if S/MIME, in some way, is better than PGP, why did not IETF choose S/MIME at the beginning?

I have no answer to the second question, because S/MIME and PGP have almost equal security services and SIP supports MIME. Thus, S/MIME had to be the most logical choice.

In the following paragraphs I will try to answer the first question. A summary of the discussion may also be seen in Table 23 on page 80.

The reason can not be that S/MIME supports stronger algorithms for encryption and digital signatures, because it actually supports some weaker algorithms than PGP. RC2/40 supported in S/MIME for encryption is considered to be weaker than Triple DES, International Data Encryption Algorithm (IDEA [47]) and CAST [46] which PGP supports and MD5 supported in S/MIME for calculating the message digest is considered to be weaker than SHA-1 which is the only secure hash algorithm supported by PGP. Therefore, S/MIME would be more vulnerable to a security downgrading attack, if this attack is viable. The security downgrading attack is a man-in-the-middle attack, where the attacker is able to chose which algorithms the two other parties should use for encryption and digital signatures.

One reason could be that the PKI in S/MIME is more centralized with users trusting a few large CAs, like companies as VeriSign. But this PKI removes much of the opportunity for the user to chose which CAs it trusts. The inventor of PGP, Phil Zimmermann, has written a short article [45], were he claims that the PKI used in PGP is actually a superset of the PKI used in S/MIME. This means that it could function in exactly the same way as the PKI used in S/MIME. On the other side is RSA Security inc., inventor of S/MIME, which claims that the PKI used in PGP will only be able to handle a small group of users. So, there is some hostility between the two parts.

Considering how the certificates are handled by different SIP entities, they could be divided into two major groups. The first group is the SIP end-users and the other is the intermediate SIP entities, like SIP proxies and registrars. The SIP end-users will probably store a small number of certificates for its trusted SIP entities and the PGP model would be preferred, because the SIP end-user has full control over the degree of trust for each certificate it receives. But the situation is different for intermediate SIP entities, because they will be forced to handle a large amount of certificates for unknown SIP entities. In this case the S/MIME model, with a small group of large trusted CAs, is preferred, because the intermediate SIP entity would be able to validate each received certificates by asking its trusted CAs. This does not mean that the PGP model would not work for this situation, but the S/MIME model is better designed to

handle this situation. Another benefit of trusting only a small number of CAs is that the number of public keys used for verifying signed certificates will be smaller. The latter case is of great importance and I think that the PKI used in S/MIME is more suitable for SIP.

Other benefits of using the PKI used in S/MIME is that the certificates are also supported by TLS. This means that users do not need to obtain a new certificate for their public keys when they use TLS together with S/MIME.

But is the benefit of using a more suitable PKI sufficient enough for excluding a relative large part of the security mechanisms in the SIP standard? Extensive changes of the standard will require that most implementations of the standard must be modified, which takes both time and effort. One of the authors of the SIP standard has another reason why PGP was excluded:

> *"PGP was removed since its specific application to sip was fragile. It was fragile because it required all elements to have a canonicalized serialization method for sip messages, which is unlikely to work. Rather, the better way is to include the thing that's being signed as a part of the message, as mime does. Thus, the preferred approach is to use PGP-MIME or S/MIME which provide those capabilities and are more robust and well thought out.*
>
> *-Jonathan R.*" (J. Rosenberg, personal communication, October 2001).

The key word in his reasoning seems to be the canonicalized serialization. What is meant with a canonicalized serialization method? A canonicalized serialization method used for SIP messages is a function that takes a SIP message or a part of a SIP message as parameter and outputs a transformed message. The transformed message has a structure that makes it possible to parse it, take it apart and reconstruct it, without any loss of information. The message is said to have a canonical form. This is crucial when the message is signed, because a slight change of the information, e.g. an extra white space, will cause a integrity failure at the receiving side. But if the message has a canonical form before it is signed and is modified by an intermediate SIP entity, e.g. a SIP proxy which capitalizes all characters in the message, then the receiver uses the canonicalized serialization method to bring the message back to a canonical form and will not get an integrity failure. As I understand, the problem with this is that all SIP standard implementations must implement this canonicalized serialization method in the exact same way, which may be a difficult task to accomplish, but not impossible. MIME and S/MIME already provide canonical form for their entities, so S/MIME seems to be a better choice when the problem with canonical form is considered.

Function	S/MIME	PGP	Preferred
Security algorithms -Message digest	MD5 and SHA-1	SHA-1	PGP
-Digital signature	DSS and RSA	DSS and RSA	Both
-Session key encryption	Diffie-Hellman and RSA	Diffie-Hellman and RSA	Both
-Message encryption	tripleDES and RC2/40	CAST, IDEA and tripleDES	PGP
Public key infrastructure	Centralized and large CAs. Uses X.509 certificates.	Decentralized and relative small CAs. Uses certificates defined in the PGP specification.	S/MIME
Integration with SIP	Transform messages to canonical form. Designed to be used with MIME.	Has no such features.	S/MIME

Table 23 Summary of the comparison between S/MIME and PGP.

4.5.1.1 Conclusions

S/MIME and PGP are quite similar in their structure and provide almost equal security services, which means that if only the security were considered, then both would be suitable to be used in SIP. But S/MIME is more flexible to integrate with SIP, because SIP already supports MIME and S/MIME is designed to be used with MIME. The public key infrastructure used in S/MIME is also preferred, because it is more suitable for intermediate SIP entities which will handle a large amount of certificates. Certificates used in S/MIME may also be used in TLS, which is an advantage.

4.5.2 IMS AKA vs. HTTP Digest Authentication

IMS AKA and HTTP Digest Authentication are both used for authentication and may be used in SIP, see “SIP/EAP/IMS AKA” on page 59 and “Digest Authentication” on page 23. IETF has specified that SIP UAs should use the Digest authentication scheme to perform the entity authentication. The reason is that Digest authentication is well known and used in HTTP. In the following paragraphs a comparison between these two schemes is made and I will give my view of which of them that is most suitable to be used in SIP. A summary of my discussion may be seen in Table 24 on page 83.

Mutual authentication may be used in Digest authentication but it is not mandatory, which may be the protocols largest security flaw. Mutual authentication is crucial for end-users if they are to trust the entity that they are connected to, because it is difficult to trust someone if you do not know its real identity. In IMS AKA, mutual authentication between the end-user and the Home Environment (HE) is mandatory. And if the HE trusts the identity of the Serving Network (SN), which it is supposed to do, then the end-user may also trust the identity of the SN.

Limited integrity protection of SIP messages may be used in Digest authentication, but it is neither sufficient enough nor mandatory. Most information in the messages is not integrity protected and may relativly easily be changed during transmission. The AKA procedure provides both entities with an integrity key, intended to be used with the algorithm described in [38] for providing integrity protection. As mentioned in the section “SIP / EAP / IMS AKA” on page 59 there exists no documentation on how to use this algorithm for SIP messages. And even if someone finds a smart way to use it, the SIP messages sent during the IMS AKA procedure can not be integrity protected with this algorithm, because the integrity key is unknown at that stage. This is different in Digest authentication, where the SIP messages are, at least, integrity protected from the first response sent by the authenticator.

Digest authentication uses nonces to prevent reply-attacks, see “Replay attacks” on page 16. The nonce should only be used for one message to provide full protection against this type of attack, but in most cases the authenticator reuses the nonce to save processor resources. If no encryption is applied to the message, then the nonce is sent in clear text, i.e. the authenticator does not hide its value. This does not affect the security if the nonce is only used once, but it is a serious security flaw if the nonce value is reused. IMS AKA uses sequence numbers to prevent against reply-attacks. Sequence numbers are relative easy to guess if you know a previous value. Therefore, IMS AKA is forced to hide the sequence number by using the secret key, i.e. it sends the exclusive or between the sequence number and the Anonymity Key (AK). My opinion is that this method is better than the one used in Digest authentication, because nonces require more processor resources to generate and is often misused. The disadvantage with sequence numbers is the requirement of having some type of resynchronization mechanism.

The secure hash function, MD5, used in Digest authentication has been analyzed and tested by many researches. And no results, which I have seen, have shown that it has any weakness in the one-way property, which the Digest authentication relies on. However, IMS AKA uses operator specific functions for the authentication procedure and these functions will probably be kept secret. And there is a risk that they contain hidden weaknesses that have not been found by the operator during testing and cryptologic analysis. This may be a major weakness in IMS AKA.

One of the most useful services in IMS AKA, is its capability of delegating the authentication procedure to an appropriate entity. Despite that only the HE knows the secret key, most of the authentication procedure is done between the SN and the end-user. This increases the scalability and the user mobility, e.g. an SIP UA may request an authorized service from a SIP proxy that is not

maintained by its operator. Digest authentication does not support delegation of the authentication procedure, which limits the end-user to only request services from entities that knows its secret password. A further disadvantage is that end-users will be required to remember secret passwords for each entity it requests services from, which probably will lead to passwords that are easily guessable by an attacker.

One advantage that both IMS AKA and Digest authentication has, is that they do not require any use of public keys, e.g. no public key infrastructure is needed and used algorithms will require less resources[1].

Digest authentication is integrated in the SIP standard and has its own headers for performing the authentication. Therefore, the authentication procedure adds a relatively small amount of extra bytes to the SIP messages. In the ideal case when the end-user knows all parameters, the authentication procedure does not require any extra SIP messages to be sent. But in ordinary cases, at least one extra SIP message is sent. IMS AKA is not integrated in the SIP standard and will therefore increase the size of the SIP messages quite much, e.g. the EAP packet must be encapsulated in the SIP header and the IMS AKA data in its turn must be encapsulated in the EAP packet. More SIP messages will also be sent when IMS AKA is used and the ideal case requires two extra SIP messages. But one should remember that IMS AKA always provides mutual authentication, which is hard to accomplish with less than two extra messages. In the ordinary case, at least four extra SIP messages will be sent, because the authenticator needs the identity of the SIP UA client.

One interesting point with IMS AKA is that the authenticator and the SIP UA client will end up with two keys, intended to be used for encryption and integrity protection. These keys seem to be useless, because the way SIP is used today prevents complete encryption and integrity protection of whole SIP messages end-to-end.

Function	Digest authentication	IMS AKA	Preferred
Mutual authentication	Optional	Mandatory	IMS AKA
Protection against reply-attacks	nonces	Hidden sequence numbers	IMS AKA

1. All algorithms used during the authentication part of AKA should be designed to be implemented on an integrated circuit with an 8-bit microprocessor running at 3.25 MHz with 8 kilobyte Read Only Memory (ROM) and 300 kilobyte Random Access Memory (RAM). And the execution time for producing an authentication vector should be less than 500 ms. Digest authentication does not have similar requirements.

Function	Digest authentication	IMS AKA	Preferred
Integrity protection	Optional and does not protect the whole message.	No integrity protection of the authentication messages. Calculated key may be used to protect future messages.	Digest authentication[a]
Used security algorithms	MD5 No algorithms that requires public keys are used.	Operator specific functions, f0-f5, f1* and f5*. No algorithms that requires public keys are used.	Digest authentication
Shared secret	Text based password.	A 128 bits secret key.	IMS AKA
Scalability	The authentication procedure is always made directly between the entities that knows the password.	The authentication procedure may be delegated to an entity that does not know the secret key.	IMS AKA
Integration with SIP	Full integration. Insignificant increase of the message size.	Significant increase of the message size. Require more messages to be sent.	Digest authentication

a. IMS AKA will certainly be the preferred scheme for this category if more effort is made to integrate it in the SIP standard.

Table 24 Summary of the comparison between Digest authentication and IMS AKA.

4.5.2.1 Conclusions

My own opinion is that IMS AKA is a stronger authentication method than Digest authentication. The main argument is that it provides mutual authentication and the authentication procedure may be delegated. The negative consequences of using IMS AKA in SIP is that the messages will be quite long and that many messages are required to be sent during the authentication procedure. But if IETF decides to integrate IMS AKA in the SIP standard, then the size of the message will certainly decrease to an acceptable level.

The operator specific functions used in IMS AKA may also be a negative property, because they may contain unexplored weaknesses due to insufficient cryptologic analysis and testing.

5 Security tests

This chapter describes and shows the result of several tests that have been performed on different implementations of a SIP stack. The document also contains a section describing a test on how to eavesdrop and modify packets on a network that uses switches to direct the traffic. Most tests described in this document have a logical connection to potential attacks on SIP mentioned in "Potential attacks on SIP", see page 30.

5.1 Test equipment

This section lists the equipment that was used for the tests.

5.1.1 Common equipment

The tests were performed on three personal computers equipped with 100 Mbit network adapters. All computers were connected through a switch. Two of the PC's are running Windows NT Server as Operating System and the third one is running Windows NT 4.0. The following software was used before or during the tests:

- Ethereal 0.8.20 [41]
- Microsoft Visual Studio 6.0
- WCI [42]

5.1.2 General network

During the test of the general network security the following software and libraries were used:

- WinPcap 2.2 (including libraries) [43]
- Vovida SIP User Agent 1.2.1

5.1.3 Vovida SIP stack

The Vovida SIP stack is a part of many applications in the VOCAL system developed by the Vovida.org community. To test the security of a protocol stack you need an application that uses the stack. For these tests the Vovida User SIP Agent, version 1.2.1, was used as the test application.

The test scripts were written in either a modified type of WCI or in an own developed SIP User Agent running above the Vovida SIP stack. The User Agent was developed in order to increase the control of which messages that should be sent and how they were constructed.

5.1.4 oSIP stack

The oSIP stack is a free implementation of the SIP standard. It is not as powerful as the Vovida SIP stack, but it is relative small in size and quite fast. The target for the tests were the oSIP User Agent, version 0.8.0.

The test scripts were written in either a modified type of WCI or in an own developed SIP User Agent running above the Vovida SIP stack. The User Agent was developed for increasing the control of which messages that should be sent and how they were constructed.

5.2 General Network security tests

These tests have been made to find out how vulnerable the SIP stack is when the underlaying network, over which the SIP stack is running, is insecure.

5.2.1 ARP spoofing

ARP spoofing [24] is a well known method to eavesdrop or modify messages on a switched network that uses the ARP protocol for mapping IP addresses to MAC addresses. A MAC address is a world unique address that is assigned to each produced Ethernet network adapter. The network adapter then uses this address, and not the IP address, for sending messages to other network adapters, on the same network.

The test script was written in Visual C++ and it is a modification of the WCI application. The modification was needed because the original source was trying to ARP spoof every computer it found, but for this test it was only interesting to ARP spoof a small known group of computers. Another reason for modifying the source was that WCI did not have any filtering mechanism for TCP and UDP packets, but instead it captures all incoming Ethernet frames. The source code was rewritten so that the application could recognize UDP and TCP packets. For UDP packets the ability to modify the content, compute the new checksum and forward the packet to the spoofed destination was added. The ability to recognize and decode SIP messages over UDP was also added.

The test involved all three computers with one of them acting as a malicious entity trying to eavesdrop and modify messages. Let the malicious computer be called C and the other two computers be called A and B respectivly. The configuration can be seen in Figure 32.

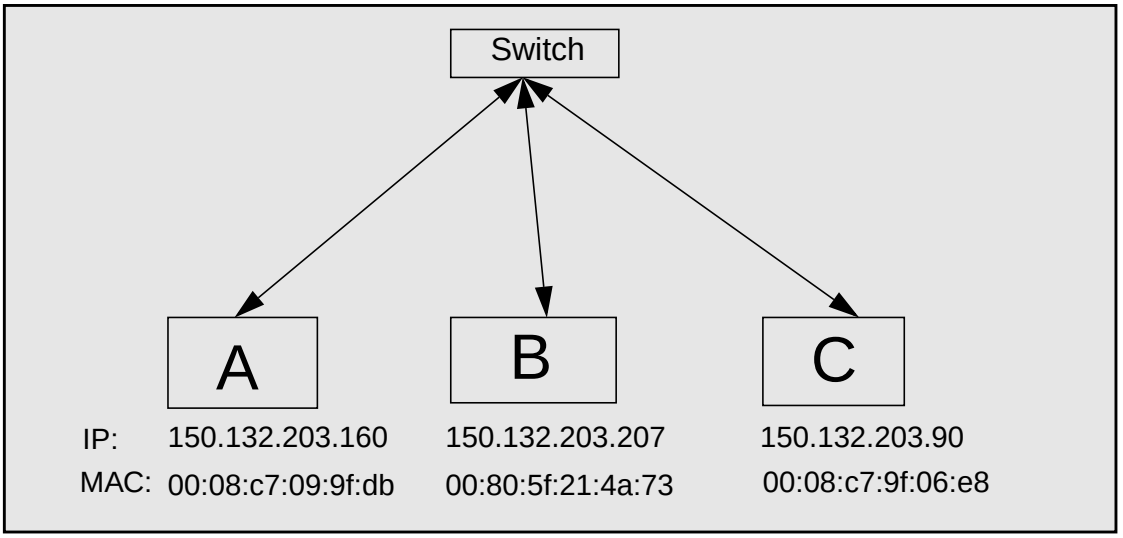

Figure 32 The computer configuration for running the test.

The messages that were sent during the test can be seen in Figure 33 and a description of the test sequence is given beneath:

1. A and B starts Ethereal, which traces all incoming and outgoing packets during the test.
2. C has started the test script and waits for broadcasted ARP requests from A and B.
3. A and B have no entries in their cached ARP table, which forces them to send ARP requests to find the MAC address for the given IP address.
4. A pings B to see if it is alive, and as a consequence user A needs to broadcast an ARP request over the network.
5. C captures the broadcasted ARP request from A to B and sends a ARP respond with its own MAC address to A.
6. B receives A's ARP request and responds with its own MAC address.
7. B clears A's entry in the cached ARP table by typing:
   arp -d <A's IP address>
8. B pings A, and as a consequence user B needs to broadcast an ARP request.
9. C captures the broadcasted ARP request from B to A and sends an ARP respond with its own MAC address to B.
10. A receives B's ARP request and responds with its own MAC address.
11. C is now able to see and modify all incoming packets from A and B.
12. A and B start their Vovida SIP User Agents, using UDP as transport protocol.
13. A invites B.
14. C receives the packet and recognizes it as a SIP message and it also recognizes that it is an **INVITE** request, but it just forwards the message.
15. B receives the **INVITE** request and the phone starts ringing.
16. B respond with a **180 Ringing** to A.

17. C receives the response from B but does not forward the message.
18. B answers the phone and respond with **200 OK** to A. B's phone stops ringing.
19. C receives the response and modifies it to a **486 Busy Here** response and forwards it to A.
20. A receives the **486 Busy Here** response and sends a **ACK** back to B. A's phone starts a busy tone.
21. B receives the ACK and thinks it is for the **200 OK** and tries to send RTP packets to A.
22. A hangs up the phone because B's phone is busy and has no open connection for receiving RTP packets.
23. B hangs up the phone because it does not receive any media.
24. C sends ARP responses to A and B with correct MAC addresses, which means that A and B now have proper ARP tables
25. End of test.

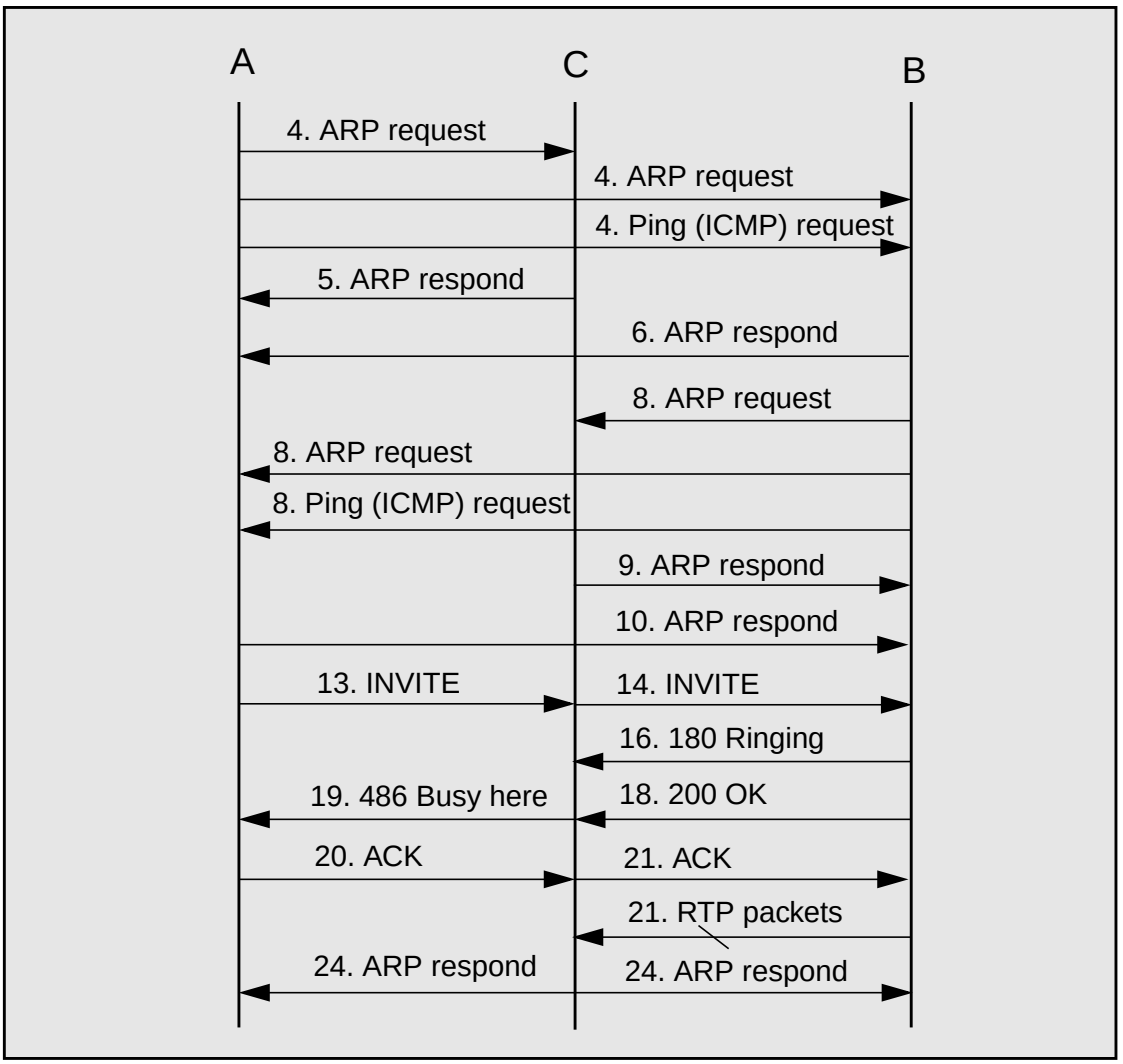

Figure 33 Illustration of the ARP spoofing test.

5.2.2 Comments on the results

The result of the failed SIP session itself is not interesting, there exists almost infinite ways to destroy the session, but the interesting part is how easy it is to affect a SIP session if you are able to eavesdrop and modifies packets over a switched network.

5.3 Security tests on the Vovida SIP stack

The Vovida SIP stack is one of the main modules included in the VOCAL system developed by the Vovida.org community. One of the applications that uses the SIP stack is their User Agent, version 1.2.1, that may be compiled on both Unix and Windows. All tests that are described in this specification are using this application to estimate the security of the actual stack. There is always a possibility that a security vulnerability does not originate from the actual stack, but originates from the application that uses the stack.

It should be mentioned that the construction of the test cases that may lead to a security vulnerability are hard and time consuming to build. Most of the following tests are focusing on either crashing or hanging the stack.

5.3.1 Test cases

This section gives the reader a detailed description of each test case that was performed on the Vovida SIP stack. It would be meaningless to present the actual source code for each test. Instead, each test case is described in a more informal way, which can be seen in Table 25.

Test description	Result
Send SIP messages larger than 3600 bytes to the User Agent.	The User Agent generates an error messages when it receives the message but it does not handle the error. This means that it only receives 3600 bytes of the message and for those header fields that do not fit inside this buffer, the User Agent assigns default values. Even worse, the User Agent overwrites the stack when it receives messages that are larger than 3600 bytes. Fortunately, it only overwrites one byte of the stack with the NULL character.
Send SIP message with random ASCII characters (48 -255) in the From header field with variable length. The total length of the message is kept below 3600 bytes.	The User Agent handles the message as it does for ordinary messages.

Test description	Result
Send SIP message with random formatting strings with variable length in the From header field, i.e. "%s%n%x". The formatting characters are bound between values of 97 and 123. The total length of the message is kept below 3600 bytes.	The User Agent handles the message as it does for ordinary messages.
Send SIP message with random ASCII characters (97-123) in the Request-URI and in the To header field with variable length. The total length of the message is kept below 3600 bytes.	The User Agent handles the message as it does for ordinary messages.
Send SIP message with negative, too large or too small Content-Length value.	The User Agent does not read this value when it receives the SIP message. Instead it calculates the length by itself and overwrites the specified value.
Send SIP message with a Content-Length value that contains a lot of characters.	Same as previous test case.
Send SIP message with a formatting string as Content-Length value.	Same result as for the two previously test cases.
Generate 100,000 new INVITE messages and extract the random part of the Call-ID value. Save all values to a file and plot the power spectrum of the sequence.	Figure 34 shows that the spectrum is almost constant. Which indicates that the generated Call-IDs may be samples of a random process. This makes it harder to predict the next Call-ID.

Table 25 Description of performed tests on the Vovida SIP stack.

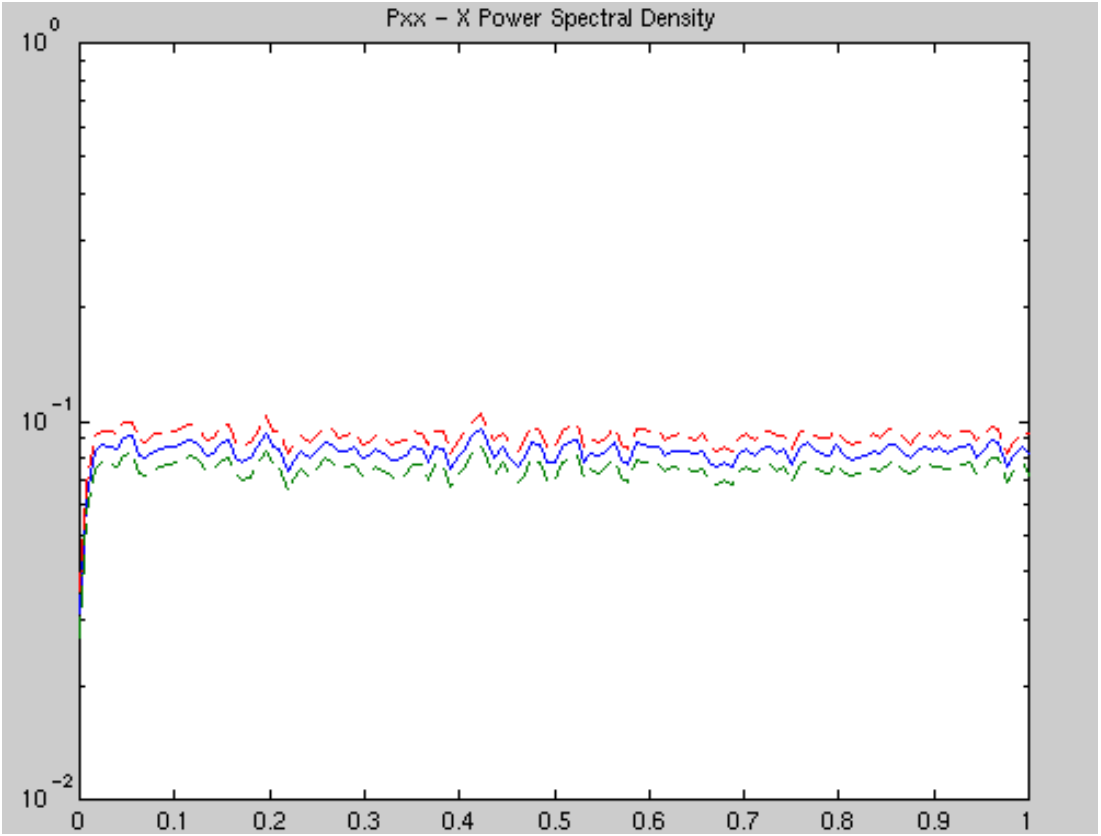

Figure 34 Power spectral density for the random part of the Call-ID header field value. The almost constant value shows that the values are random.

Test description	Result
Generate two series of 100,000 INVITE requests and extract their Call-ID values. Save all values in two separate files and calculate the coherence between the series. The coherence, that can have values between 0 and 1, shows the degree of dependence between the series for different frequencies.	Figure 35 shows that the coherence between the two series is relatively small for all frequencies. This means that it is a hard task to predict a series of Call-ID's, even if you have managed to eavesdrop recent values. As a comparison[a], Figure 36 shows the coherence between two series that have been generated by the following recursive sequence, but with different starting values: $x_n = 7 \cdot x_{n-1} \bmod 4231$ If you plot this sequence it looks random and it has an almost constant spectrum, but the coherence shows its weakness and it should not bee used as a random generator. The reason for the high coherence between the two series arises from the fact that the recursive series is cyclic and the cycle length is relatively short.

a. There are a significant difference between the scales of the y-axis in the figures.

Table 26 Continuation of Table 25.

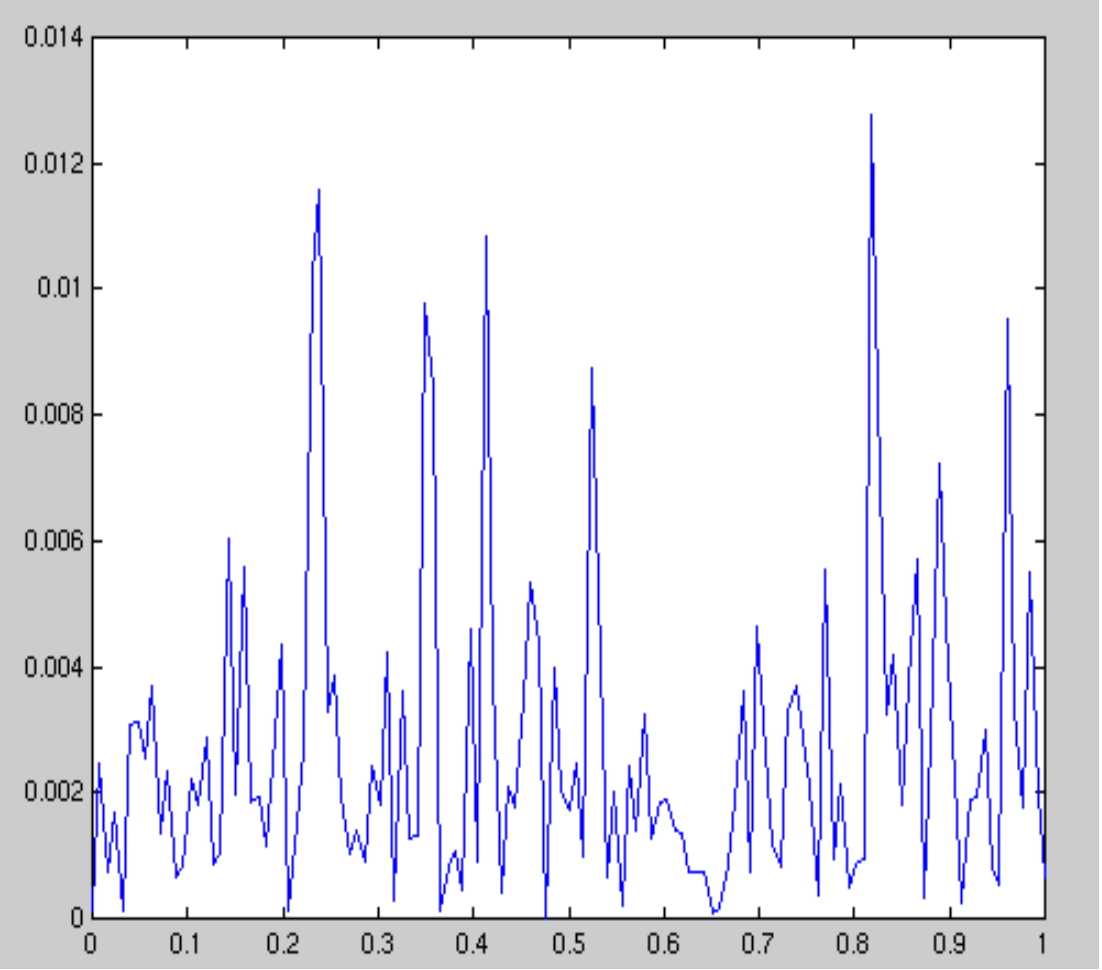

Figure 35 Plot of the coherence between two series of generated Call-ID values.

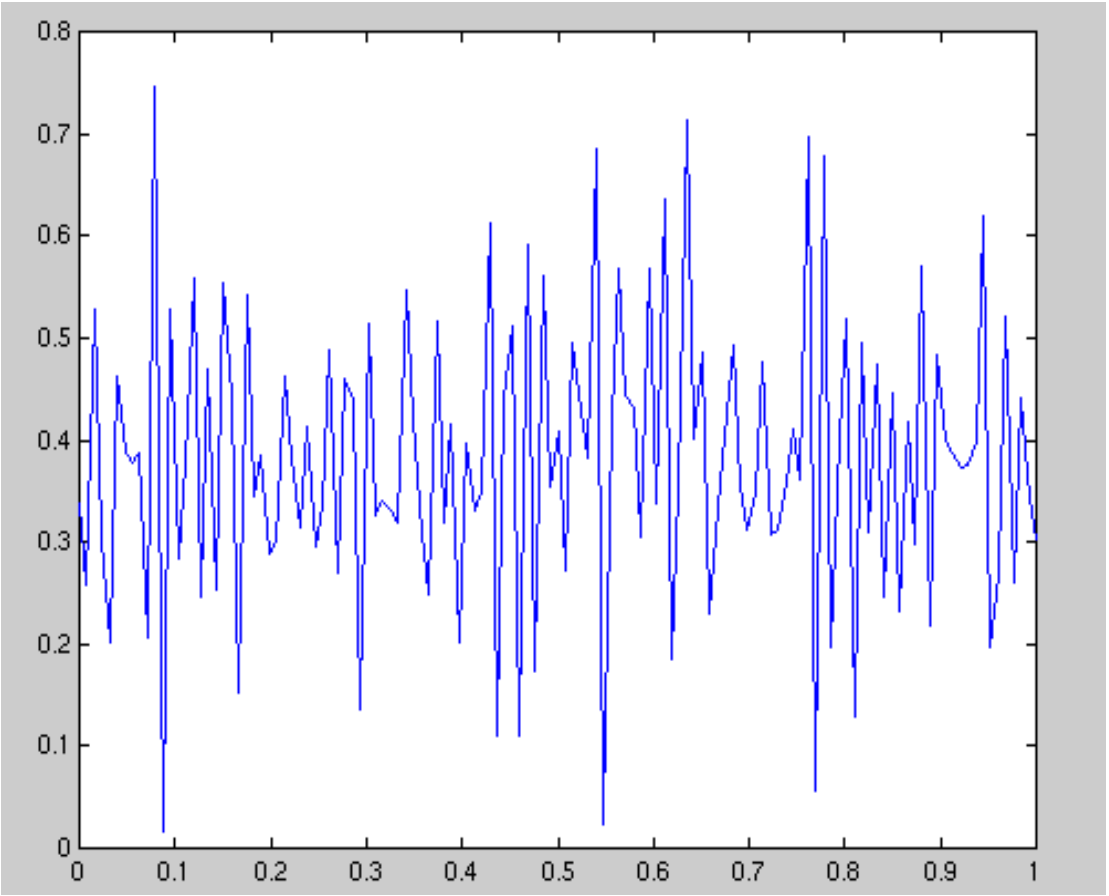

Figure 36 Plot of the coherence between two series with different starting values ($x_{0,0} = 1, x_{0,1} = 23$) of the recursive series: $x_n = 7 \cdot x_{n-1} \bmod 4231$.

Test description	Result
Send an INVITE to a User Agent and change the Call-ID in the ACK request sent after receiving 200 OK.	The User Agent ignores the ACK because it has not received any request with the new Call-ID.
A User Agent, A, invites another User Agent, B. B replies to the invite with 180 Ringing and 200 OK. A sends an ACK to B that is captured by a malicious person, who changes the IP address in the SDP message to its own and then forwards the message to B.	B notices the change of IP address but ignores it and uses the SDP message that was sent in the INVITE request. A sends its RTP packets to B and B sends them to A.

Test description	Result
A User Agent, A, invites another User Agent, B. B replies to the invite with 180 Ringing and 200 OK. A malicious person captures the 200 OK and changes the IP address in the SDP message to its own. A sends an ACK to B.	User A uses the IP address in the changed SDP message and starts to send RTP packets to the malicious person and B sends its RTP packets to A.
A User Agent, A, invites another User Agent, B. B replies to the invite with 180 Ringing and 200 OK. A sends an ACK to B. Both User Agents start sending RTP packets to each other. A malicious person has been able to capture an earlier message from the dialog between A and B and tries to construct a BYE messages from the header fields in the captured message. The malicious person sends the BYE request to B.	User B receives the BYE request and terminates the session. User A continues to send RTP packets to B.

Table 27 Continuation of Table 26.

5.3.2 Comments on the results

The results show that the Vovida SIP stack is relatively resistant against various types of stack and formatted string attacks. One explanation may be that the stack is written in C++ with stable libraries for manipulating strings instead of using ANSI C, which has a couple of string manipulation functions that are considered to be unsafe.

The tests also show that the pseudorandom number generator that is used by the Vovida SIP stack is good enough to be used for producing unique Call-ID's which is critical for protection against malicious persons that try to send spoofed messages without the ability to eavesdrop on the session. It should be mentioned that the Vovida SIP stack has two ways to generate its pseudorandom numbers and the tested one is the weakest of them, but it is the default pseudorandom number generator. However, it should not be considered as a cryptographic pseudorandom number generator. The other pseudorandom number generator that the Vovida SIP stack supports is called Yarrow [4] and it is considered to be a cryptographic pseudorandom number generator and it is used in many security applications, e.g. SSH. Unfortunately, no tests have been made to verify its strength.

The last tests described in the previous sections shows how easy it is to make destructive actions when it is possible to eavesdrop on the session and modify the content without detection. In this case it does not matter how strong the pseudorandom number generator is, because the malicious person just copies these values when it constructs new messages or modifies previous messages. The only way to protect against this is to apply data integrity protection and

encryption. But it is wrong to blame Vovida for this because the SIP standard does not state the tools for accomplishing data integrity protection or encryption.

The most frightening result is that the stack can not handle received messages larger than 3600 bytes. And as a consequence of this it overwrites the memory on the stack with one byte. Fortunately, this is not a catastrophe because you only have one byte to mess with and that is not enough for a malicious person to launch a buffer overflow attack. After the tests were performed Vovida released a new version, version 1.3.0, of the stack which fixed some parts of this bug. Sadly, the fix gives a malicious person the possibility to launch a Denial of Service attack against the receiving User Agent. This is possible because the SIP stack terminates the thread that handles incoming requests when it receives messages larger than 3600 bytes, which leads to a User Agent that is unable to receive any request. The only solution is to restart the application that is built above the SIP stack.

5.4 Security tests on the oSIP stack

The oSIP stack is an open source project and anyone is welcome to contribute with new features and other improvements. The project is still in an early stage and many features are still missing. One important feature that is missing is the session management, i.e. it does not handle any media. In the current version, this feature must be implemented by the application that uses the oSIP stack.

The oSIP stack is implemented in ANSI C and may be compiled on both Unix and Windows environments and it includes a User Agent, version 0.8.0, that was the target for these tests. The User Agent is included in the implementation to give developers ideas on how to implement an application above the stack. Therefore is it quite primitive and does not handle any media.

The intention was to perform the same tests on the oSIP stack as those that were performed on the Vovida SIP stack. This makes it easy to compare the difference between these implementations. Test cases which requires session management have not been performed, because the oSIP User Agent does not support that feature.

5.4.1 Test cases

This section gives the reader a detailed description of each test case that was performed on the oSIP stack. It would be meaningless to present the actual source code for each test. Instead, each test case is described in a more informal way, which may be seen in Table 28.

Test description	Result
Send SIP messages larger than 4000 bytes to the User Agent.	The User Agent crashes. The reason is that it does not expect messages larger that 4000 bytes and therefore overflows an allocated buffer. No error messages are output and the UA has to be restarted. The buffer is allocated dynamically, which prevents an attacker from launching a buffer overflow attack.
Send SIP messages with at least one SIP URI that is larger than 200 bytes, e.g. in the From header field.	The User Agent crashes. The reason is that it does not expect URIs larger that 200 bytes and therefore overflows an allocated buffer. No error messages are output and the UA has to be restarted. The buffer is allocated dynamically, which prevents an attacker from launching a buffer overflow attack.
Send SIP message with random ASCII characters (48 -255) in the From header field with variable length. The total length of the message is kept below 4000 bytes and the From header field is not longer than 200 bytes.	The User Agent handles the message as it does for ordinary messages.
Send SIP message with random formatting strings with variable length in the From header field, i.e. "%s%n%x". The formatting characters are bound between values of 97 and 123. The total length of the message is kept below 4000 bytes and the From header field is not longer than 200 bytes.	The User Agent handles the message as it does for ordinary messages.
Send SIP message with random ASCII characters (97-123) in the Request-URI and in the To header field with variable length. The total length of the message is kept below 4000 bytes and the From header field is not longer than 200 bytes.	The User Agent handles the message as it does for ordinary messages.

Test description	Result
Send SIP message with negative, too large or too small Content-Length value.	The User Agent outputs error messages for values that are too large or negative and then it discards them. For values that are positive but smaller than the expected value, the User Agent removes the message body and continues the message exchange as if nothing has happened.
Send SIP message with a Content-Length value that contains a lot of characters.	The User Agent outputs an error message and discards the message.
Send SIP message with a formatting string as Content-Length value.	The User Agent removes the message body and continues the message exchange as if nothing has happened.
Generate 100,000 new INVITE messages and extract the random part of the Call-ID value. Save all values to a file and plot the power spectrum of the sequence.	Figure 37 shows that the spectrum is almost constant. Which indicates that the generated Call-IDs may be samples of a random process. This makes it harder to predict the next Call-ID.

Table 28 Description of performed tests on the oSIP stack.

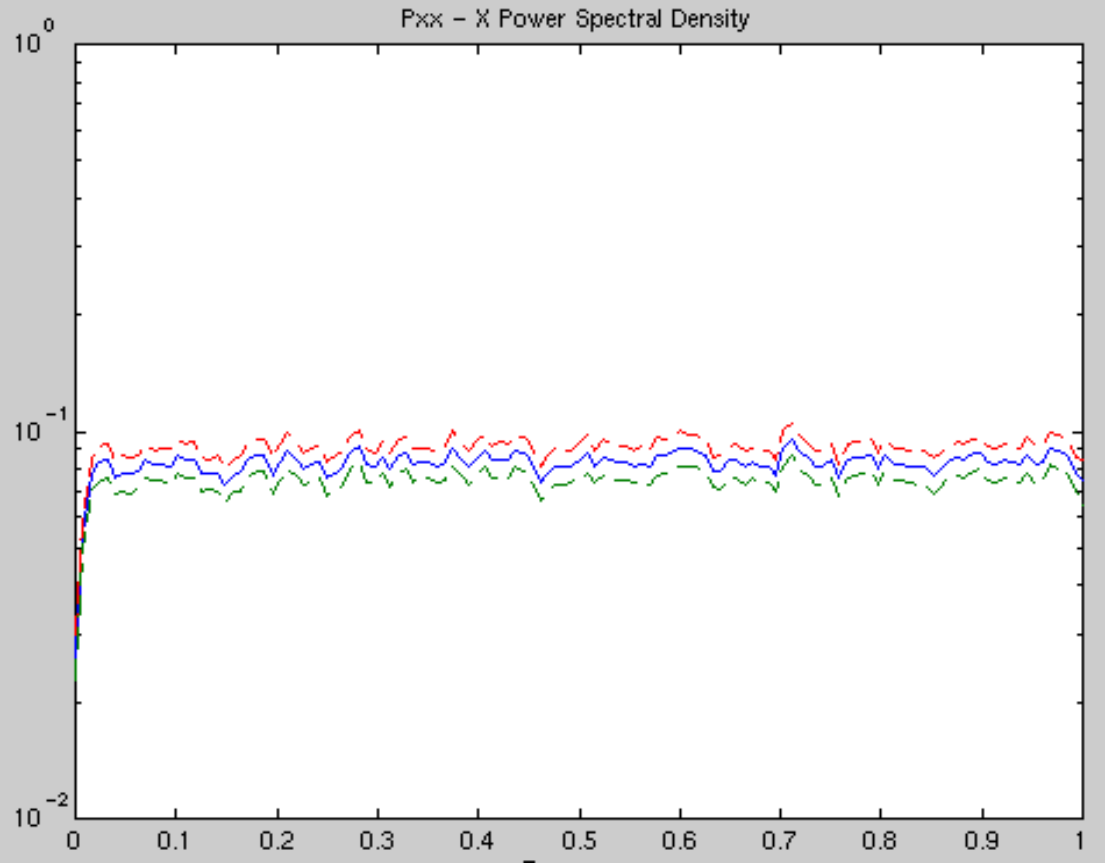

Figure 37 Power spectral density for the random part of the Call-ID header field value. The almost constant value shows that the values are random.

Test description	Result
Generate two series of 100000 INVITE requests and extract their Call-ID values. Save all values in two separate files and calculate the coherence between the series. The coherence, that can have values between 0 and 1, shows the degree of dependence between the series for different frequencies.	Figure 38 shows that the coherence between the two series is relatively small for all frequencies. This means that it is a hard task to predict a series of Call-ID's, even if you have a managed to eavesdrop on recent values. As a comparison[a], Figure 36 on page 94 shows the coherence between two series that are easy to predict if previous values are known.

a. There are a significant difference between the scales of the y-axis in the figures.

Table 29 Continuation of Table 28.

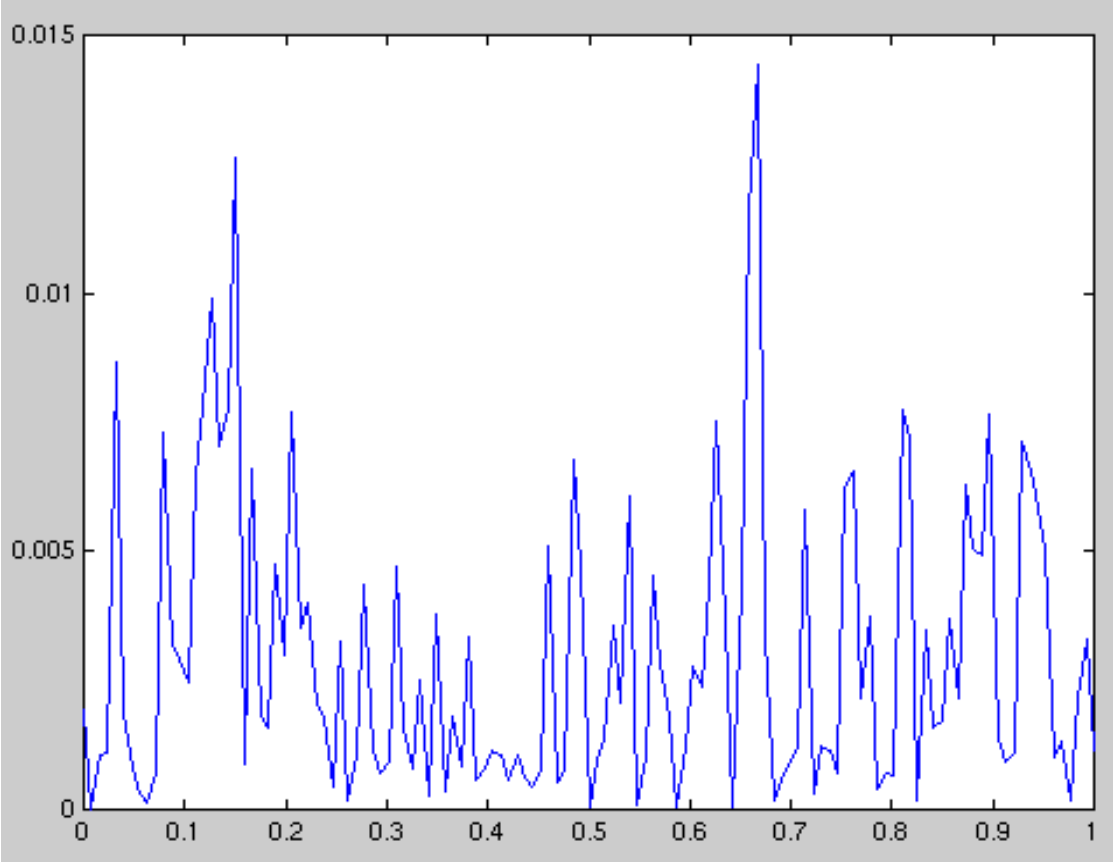

Figure 38 Plot of the coherence between two series of generated Call-ID values.

Test description	Result
Send an INVITE to a User Agent and change the Call-ID in the ACK request sent after receiving 200 OK.	The User Agent does not recognize the new Call-ID and generates an error message. It retransmits the 200 OK response, with the Call-ID set to the old value, to the other entity.

Table 30 Continuation of Table 28.

5.4.2 Comments on the results

The results show that the oSIP stack is relatively easy to crash and fool remotely. Although it is written in ANSI C, no signs of Buffer Overflow or Formatting Strings vulnerabilities are shown. This arises from the fact that oSIP uses dynamic memory allocation for its buffers instead of using static allocated buffers. Which means that it is impossible for an attacker to manipulate the stack pointer when he or she overflows a buffer.

oSIP uses the standard C functions for generating the Call-ID values and this function should not be considered as a cryptographic pseudorandom genera-

tor. Although oSIP does not use a cryptographic pseudorandom generator, the strength of the Call-ID values is good enough.

The most frightening result is that the stack crashes when received SIP messages contain at least one SIP URI that is larger than 200 bytes. It will not be a hard task for an attacker to use this weakness for launching a Denial Of Service attack.

Another weakness is that the oSIP stack relies on the fact that the Content-Length header field value should contain a correct numeric value. The removal of the message body, when it receives a Content-Length header field value that is either non-numeric or too small, is not a proper action to solve the problem.

5.5 oSIP stack vs. Vovida SIP stack

Neither of the tested SIP stacks has prioritized security, which the test results confirm. Both SIP stacks suffer from overflow vulnerabilities when they receive large messages or messages with long header field values. The Vovida SIP stack does not crash when it receives large messages, which the oSIP stack does, but it overwrites memory.

Both SIP stacks are resident to Format Strings attacks, i.e. an attack where control characters are inserted in some header fields to crash the SIP stack or to execute arbitrary code.

It is possible to fool the oSIP stack by including false information in the Content-Length header field, which is not possible when the Vovida SIP stack is used.

Both SIP stacks use, by default, a standard C function as random generator. Vovida has gone one step further and has implemented a cryptographic pseudorandom generator to provide stronger Call-ID values. In oSIP, there is no option to use another random generator.

One feature, provided by the SIP standard, that has not been tested is the HTTP Authentication. The oSIP stack does not support this feature at all, which is a serious weakness. The Vovida SIP stack supports HTTP Authentication, by the means of generating a response to an authentication request from another entity. It has no functionality to generate authentication requests, i.e. the logic for generating parameters to the authentication procedure is not implemented. This is the main reason why this feature has not been tested on the Vovida stack, because the parameter generation is the most interesting part to test, e.g. tests on how it generates and changes the nonce values had been interesting to perform.

At the moment, the Vovida SIP stack is more secure than the oSIP stack and it provides more functionality. Thus, if the focus is on security then the best choice is the Vovida SIP stack. But it is important to mention that no one of the tested stacks is not even close to provide full security. The main reason is that the SIP standard does not state which security mechanisms to use to be able to achieve this.

6 Conclusions

6.1 Summary

According to my opinion the security mechanisms provided by the SIP standard and those that are proposed to be used are sufficient to provide a secure signalling session if they are used together in a well-specified way. The only restriction is that TLS and/or IPSec must be used in conjunction with SIP to achieve this. Both these protocols require the use of public key certificates or security associations. The idea of using public key certificates for distributing public keys solves several problems. However, public key ceritficates are not suitable to be used by equipments that have limited storage resources and are used in environments with limited bandwidth, due to their size.

IMS AKA may be considered as the preferred mechanism to provide entity authentication in SIP. Even if it is more suitable to be used in the mobile telecommunication system than in the Internet systems, it may be a better solution than the Digest authentication. If someone finds a constructive way to use the produced symmetric keys for encryption and integrity protection of the SIP messages, the IMS AKA should definitively replace Digest authentication as the default mechanism for entity authentication.

This thesis is focused on the security mechanisms in SIP that have been developed by IETF. Most of the information about proposed security mechanisms has been obtained from drafts published by IETF. This may have influenced the information in this thesis and the results from the analysis in a way that does not give the full picture of how different security mechanisms may be used in future SIP applications, i.e. IETF tends to give the designers too many choices how to implement these security mechanisms. It does not matter if these security mechanisms are well done if no one implements them or makes an adequate implementation of recommended security mechanisms. Even if they are implemented adequately, the default option is not to use them in the application, i.e. the user is often responsible of turning them on. A classical example of this is E-mails which in most cases are sent unprotected by the average user even if most E-mail applications provide adequate security mechanisms. Despite that E-mails have been one of the largest source for distributing Trojan Horses and other types of viruses.

The tested SIP stacks also confirms the fact that security mechanisms are implemented at the latest stage and that they are not properly tested.

6.2 Summary of the results

- Basic and Digest authentication are integrated in the SIP standard and they provide entity authentication and limited message authentication.
- Basic authentication should not be used, because it sends the credentials in clear text.

- Several successfull attacks are possible against SIP if only Basic or Digest authentication is used. Replay attacks are a major threat to the Digest authentication if the authentication method is used improperly.
- S/MIME may provide end-to-end security services for the SIP message body and for some header fields. The disadvantage of using S/MIME to provide end-to-end security services for the header fields is an increased message size. S/MIME requires public key certificates which may be self-signed.
- PGP has been excluded from the SIP standard due to problems with the integration in SIP and its public key infrastructure is not suitable for intermediate SIP servers.
- A variant of UMTS AKA called IMS AKA is a scalable and secure procedure to perform entity authentication. The current specification how to use it in SIP is insufficient and needs to be more specified regarding the use of the encryption key and the integrity key. If this is done, then encryption and integrity protection may be provided without public key certificates.
- TLS may be used in conjunction with SIP, to provide hop-to-hop security between trusted entities. The trust is based on public key certificates. A long-lived TLS connection between the SIP User Agent and the outbound proxy is a preferred way to protect the SIP messages inside the home network.
- IPSec may be used in conjunction with SIP and is most suitable to be used between intermediate SIP servers. IPSec is not suitable to be implemented in the SIP User Agent.
- Two SIP stacks has been tested and the test results show that if only limited security mechanisms is provided , then it is realtive easy to crash or hang the SIP stack.
- No single procedure or algorithm that is included in the SIP standard or has been proposed to be used in the SIP standard can provides end-to-end security services for the complete SIP message.

7 References

[1] William Stallings. “Cryptography and Network Security : Principles and Practice, Second Edition”. Prentice-Hall, June 1998.

[2] Rosenberg, Schulzrinne, Camarillo, Johnston, Peterson, Sparks, Handley, Schooler. “SIP : Session Initiation Protocol”. draft-ietf-sip-rfc2543bis-05. Internet Engineering Task Force, October 2001.

[3] M. Handley, H. Schulzrinne, E. Schooler, J. Rosenberg. “SIP : Session Initiation Protocol”. Request For Comments 2543. Internet Engineering Task Force, March 1999.

[4] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart. “HTTP Authentication : Basic and Digest Access Authentication”. Request For Comments 2617. Internet Engineering Task Force, June 1999.

[5] T. Dierks, C. Allen. “The TLS protocol version 1.0”. Request For Comments 2246. Internet Engineering Task Force, January 1999.

[6] S. Kent, R. Atkinson, “Security architecture fot the internet protocol”. Request For Comments 2401. Internet Engineering Task Force, November 1998.

[7] Raymond G. Kammer. “Data Encryption Standard”, Federal Information Processing Standards Publication 46-3. National Institute of Standards and Technology, October 1999.

[8] H. Feistel. "Cryptography and Computer Privacy" Scientific American, v. 228, n. 5, May 73, pp. 15-23

[9] R.L. Rivest, A. Shamir, and L.M. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM (2) 21 (1978), 120-126.

[10] Diffie, W., and Hellman, M., “New Directions in Cryptography.”, IEEE Transactions on Information Theory, November 1976.

[11] R. Rivest. “The MD5 Message-Digest Algorithm”. Request For Comments 1321. Internet Engineering Task Force, April 1992.

[12] Arati Prabhakar, “Secure Hash Standard”, Federal Information Processing Standards Publication 180-1. National Institute of Standards and Technology, May 1993.

[13] T. Berners-Lee, R. Fielding, H. Frystyk. “Hypertext Transfer Protocol -- HTTP / 1.0”. Request For Comments 1945. Internet Engineering Task Force, May 1996.

[14] “ITU-T Recommendation H.323”. Draft v4. ITU-T, November 2000.

[15] M. Handley, V. Jacobson. “SDP : Session Description Protocol”, Request For Comments 2327. Internet Engineering Task Force, April 1998.

[16] Jonathan B. Postel. “Simple Mail Transfer Protocol”. Request For Comments 788. Internet Engineering Task Force, November 1981.

[17] S. Bradner, “The Internet Standard Process -- Revision 3”. Request For Comments 2026. Internet Engineering Task Force, October 1996.

[18] N. Borenstein, N. Freed. “MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies “. Request For Comments 1341. Internet Engineering Task Force, June 1992.

[19] B. Ramsdell. “S/MIME Version 3 Message Specification”. Request For Comments 2633. Internet Engineering Task Force, June 1999.

[20] J. Callas, L. Donnerhacke, H. Finney, R. Thayer, “Open PGP Message Format”. Request For Comments 2440. Internet Engineering Task Force, November 1998.

[21] Christopher Benson. “Security Threats”. Best Practices for Enterprise Security. Microsoft TechNet, 2000.

[22] John Postel. “Internet Protocol”. Request For Commens 791. Defense Advanced Research Project Agency, September 1981.

[23] David C. Plummer. “An Ethernet Address Resolution Protocol”. Request For Comments. Network Working Group, November 1982.

[24] Sean Whalen, “An Introduction to Arp Spoofing”. Packset Storm, April 2001. On the web 2002-03-21 at http://packetstormsecurity.org/papers/protocols/intro_to_arp_spoofing.pdf.

[25] Daemon9, Route, Infinity (Nicknames). “[IP-spoofing Demystified]: (Trust-Relationship Exploitation)”. Phrack Magazine, June 1996. On the web 2002-03-21 at http://www.fc.net/phrack/files/p48/p48-14.html.

[26] J. Postel. “Internet Control Message Protocol”. Request For Comments 792. Network Working Group, September 1981.

[27] Yuri Volobuev. “Playing redir games with ARP and ICMP”. Insecure.Org, September 1997. On the web 2002-03-21 at http://www.insecure.org/sploits/arp.games.html.

[28] Aleph One. “Smashing The Stack For Fun And Profit”. Phrack Magazine, November 1996. On the web 2002-03-21 at http://www.phrack.org/show.php?p=49&a=14.

[29] R. Housley. “Cryptographic Message Syntax”. Request For Comments 2630. Internet Engineering Task Force, June 1999.

[30] E. Rescorla, “Diffie-Hellman Key Agreement Method”, Request For Comments 2631. Internet Engineering Task Force, June 1999.

[31] Arati Prabhakar, “DIGITAL SIGNATURE STANDARD (DSS)”, Federal Information Processing Standards Publication 186. National Institute of Standards and Technology, May 1994.

[32] R. Housley, W. Ford, W. Polk, D. Solo, “Internet X.509 Public Key Infrastructure Certificate and CRL Profile”, Request For Comments 2459. Internet Engineering Task Force, January 1999.

[33] ITU-T Recommendation X.509 (1997) | ISO/IEC 9594-8:1997, Information technology - Open Systems Interconnection- The Directory: Authentication framework.

[34] Simpson, W., "The Point-to-Point Protocol (PPP)", Request For Comments 1661. Internet Engineering Task Forrce, July 1994.

[35] L. Blunk, J. Vollbrecht, “PPP Extensible Authentication Protocol (EAP)”, Request For Comments 2284. Internet Engineering Task Force, March 1998.

[36] 3GPP Technical Specification 3GPP TS 33.102 V3.6.0: "Technical Specification Group Services and System Aspects; 3G Security; Security Architecture (Release 1999)", 3rd Generation Partnership Project, November 2000.

[37] 3GPP Technical Specification 3GPP TS 33.105 V4.1.0, "Technical Specification Group Services and System Aspects; 3G Security; Cryptographic Algorithm Requirements (Release 4)". 3rd Generation Partnership Project, June 2001

[38] 3GPP Technical Specification 3GPP TS 35.201 V4.1.0, “Technical Specification Group Services and System Aspects; 3G Security; Specification of the 3GPP Confidentiality and Integrity Algorithms; Document 1: f8 and f9 Specification (Release 4)”. 3rd Generation Partnership Project, December 2001

[39] 3GPP Technical Specification 3GPP TS 23.228 V5.2.0, “Technical Specification Group Services and System Aspects; IP Multimedia Subsystem (IMS); Stage 2 (Release 5)”. 3rd Generation Partnership Project, October 2001

[40] J. Arkko, H. Haverinen, “EAP AKA Authentication”, Internet Draft. November 2001.

[41] Ethereal. On the web 2002-03-21 at http://www.ethereal.com/download.html.

[42] ARP0c connection interceptor. On the web 2002-03-21 at http://www.phenoelit.de/arpoc/index.html.

[43] WinPcap: the Free Packet Capture Architecture for Windows. On the web 2002-03-21 at http://netgroup-serv.polito.it/winpcap/install/default.htm.

[44] “A secure pseudorandom number generator”. Counterpane Internet Security. On the web 2002-03-21 at http://www.counterpane.com/yarrow.html.

[45] Phil Zimmermann, “Why OpenPGP’s PKI is better than an X.509 PKI”. Open PGP Alliance, Februari 2001. On the web 2002-03-21 at http://www.openpgp.org/technical/whybetter.shtml.

[46] C. Adams, “The CAST-128 Encryption Algorithm”, Request For Comments 2144. Internet Engineering Task Force, May 1997.

[47] X. Lai, J. Massey, “A Proposal for a New Block Encryption Standard.” EUROCRYPT '90, 1990.

[48] R. Rivest, “A description of the RC2(r) Encryption Algorithm”, Internet Draft draft-rivest-rc2desc-00.txt, Internet Engineering Task Force, June 1997.

[49] 3GPP Technical Specification 3GPP TS 23.060 V4.1.0, “Technical Specification Group Services and System Aspects; General Packet Radio Service (GPRS); Service description; Stage 2 (Release 4), June 2001.

[50] S. Deering, R. Hinden, “Internet Protocol, Version 6 (IPv6) Specification”, Request For Comments 2460. Internet Engineering Task Force, December 1998.

[51] M. Garcia, D. Mills, G. Bajko, G. Mayer, F. Derome, H. Shieh, A. Allen, S. Chotai, K. Drage, J. Bharatia, K. Hobbis , D. Willis, “3GPP requirements on SIP”, Internet Draft. Internet Engineering Task Force, November 2001.

[52] R. Droms, “Dynamic Host Configuration Protocol”, Request For Comments 1541. Internet Engineering Task Force, October 1993.

[53] D. Harkins, D. Carrel, “The Internet Key Exchange (IKE)”, Request for Comments 2409. Internet Engineering Task Force, November 1998.

[54] J. Dohmen, L. Olaussen, “UMTS Authentication and Key Agreement - A compehensive illustration of AKA procedures within the UMTS system”, Gradutate Thesis. Agder University College, May 2001.

[55] J. Postel, “User Datagram Protocol”, Request For Comments 768. Internet Engineering Task Force, August 1980.

[56] J. Postel, “TRANSMISSION CONTROL PROTOCOL; DARPA INTERNET PROGRAM; PROTOCOL SPECIFICATION”, Request For Comments 793. Internet Engineering Task Force, September 1981.

[57] Alan O. Freier, Philip Karlton, Paul C. Kocher, “The SSL Protocol; Version 3.0”, Internet Draft. Internet Engineering Task Force, March 1996.

[58] H. Krawczyk, M. Bellare, R. Canetti, “HMAC: Keyed-Hashing for Message Authentication”, Request For Comments 2104. Internet Engineering Task Force, February 1997.

[59] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, “SIP: Session Initiation Protocol”. draft-ietf-sip-rfc2543bis-09. Internet Engineering Task Force, February 2002.

[60] R. Atkinson, “IP Encapsulating Security Payload (ESP)”, Request For Comments 1827. Internet Engineering Task Force, August 1995.

[61] R. Atkinson, “IP Authentication Header”, Request For Comments 1826. Internet Engineering Task Force, August 1995.

[62] H. Orman, "The Oakley Key Determination Protocol", Request For Comments 2412. Internet Engineering Task Force, November 1998.

[63] D. Maughan, M. Schertler, M. Schneider, J. Turner, “Internet Security Association and Key Management Protocol (ISAKMP)”, Request For Comments 2408. Internet Engineering Task Force, November 1998.

[64] D. Harkins, D. Carrel, “The Internet Key Exchange (IKE)”, Request For Comments 2409. Internet Engineering Task Force, November 1998.

[65] SANS Institute, “NSA Glossary of Terms Used in Security and Intrusion Detection”. On the web 2002-03-21 at http://www.sans.org/newlook/resources/glossary.htm.