KUNGL TEKNISKA HÖGSKOLAN

Numerisk analys och datalogi

RTP Security in 3G Networks

av

Karl Norrman

TRITA-NA-E00XY

NADA

Nada (Numerisk analys och datalogi)
SU
100 44   Stockholm

Deptartment of Numerical Analysis
and Computer Science
Stockholm University
SE-100 44 Stockholm, SWEDEN

RTP Security in 3G Networks

av
Karl Norrman

TRITA-NA-E00XY

Master´s Thesis in Computer Science (20 credits)
at the First Degree Programme in Mathematics and Computer Science,
Stockholm University year 2000
Supervisor at Nada was Mikael Goldmann
Examiner was Stefan Arnborg

Abstract

There is a need to protect the privacy of streams belonging to multimedia sessions. The construction of a protection mechanism should consider aspects such as confidentiality, integrity and authentication, among others. Since mobile terminals of the third generation will provide support for multimedia sessions, the protection mechanism also has to pay respect to the characteristics of wireless environments to be attractive.

Since third generation terminals should be compatible with Internet applications, it is an advantage to use protocols that are accepted by the Internet community. The protocols used on the Internet are standardized with in the Internet Engineering Task Force (IETF). Hence, the protection mechanism should be standardized there.

This Master's thesis presents an analysis of the requirements for a protection mechanism for multimedia sessions based on the Real-time Transport Protocol (RTP). It also presents a solution (including an implementation and an evaluation thereof) for the problem. Performed simulations of voice and video transmissions in a wireless environment suggests that the solution most probably will be adequate in practice as well.

Referat

RTP-säkerhet i 3G-nätverk

Det finns ett behov av att kunna skydda strömmar i multimediasessioner. Konstruktionen av en skyddsmekanism bör bland annat beakta aspekter som konfidentialitet, integritet och autentisiering. Eftersom mobila terminaler av den tredje generationen kommer att tillhandahålla stöd för multimediasessioner måste skyddsmekanismen även ta hänsyn till egenskaperna hos trådlösa omgivningar för att vara attraktiv.

Eftersom tredje generationens terminaler ska vara kompatibla med Internet-applikationer, är det en fördel att använda protokoll som redan är accepterade av Internetanvändarna. De protokoll som används på Internet är standardiserade av Internet Engineering Task Force (IETF). Därför bör skyddsmekanismen standardiseras där.

Den här magisteruppsatsen presenterar en analys av de krav som ställs på en skyddsmekanism för multimediasessioner baserade på Realtime Transport Protocol (RTP). Den presenterar också en lösning (samt en implementation och utvärdering därav) för problemet. Utförda simuleringar av ljud- och video-sändningar i en trådlös omgivning ger en fingervisning om att lösningen troligen fungerar tillfredställande även i praktiken.

Preface

This Master's thesis was written during the period from October 2000 to February 2001 at Ericsson Research, Communications Security Lab. I would like to thank the following people especially

- Rolf Blom, Elisabetta Carrara and Mats Näslund, whom I worked together with on the theoretical parts of the thesis,
- Vicknesan Ayadurai at SwitchLab for educating me on networking and introducing me to the wonderful world of the FreeBSD operating system,
- Fredrik Lindholm, with whom I co-wrote the chapter on key-management,

I am grateful to Göran Selander and Mats Näslund for happily answering all my questions on cryptography and other more or less unrelated questions regarding mathematics. I also would like to thank everyone else at the Communications Security Lab for making these months a pleasant time.

Contents

Chapter 1

Introduction

Today, telephone calls are not handled in the most efficient way by the phone networks. One reason for this is the way the calling parties are connected to each other. Traditional phone networks set up a connection by allocating a path for the call through the network and reserve this path during the entire call. This strategy is not optimal in the sense that it locks more resources than necessary. For example, a conversation consists of silence to a large extent. During these periods of silence there is really no need for the call to occupy the path, but the way the network locks the path at dial-up time, it does. Networks that operate like this are called circuit-switched networks.

A different approach to performing the call is to continuously take small samples of the spoken words and then transmit these samples piece by piece to the receiving side. The "telephone number" of the receiver is attached to each piece, and then the pieces are routed through the network to the receiving phone according to some rule. Note that there is no need for all the pieces to take the same route in this scenario, as long as they all end up at the correct destination. The pieces of speech are called packets. The advantage of the packet-based approach is that the network is only used during the time the packets are transmitted. In contrast to traditional phone networks, these kind of networks will not allocate a path in advance (and hence will not deny another call from using parts of it), resulting in a more efficient usage of the network. Networks based on this idea are said to be packet-switched. There are however drawbacks to this approach. Each packet has to contain additional transport-information, which means that more data has to be transmitted. Packets might also get lost during transmission, or might arrive out of order. There are efficient techniques for dealing with these problems; the most important ones will be presented later in this report.

1.1 VoIP and VoIPoW

As mentioned above, there is a way of utilizing the networks in a more efficient way. Freeing resources opens up the possibility of transmission of more data through the network.

Internet and many other networks use special "languages", called *protocols*, to administrate the traffic (see Section 2.2). One of these protocols is called IP (see Section 2.2.1) and the technique that provides voice communication, e.g., a phone call, over a network using IP are called Voice over IP (VoIP).

By taking this idea one step further and allowing terminals capable of handling IP to be wirelessly connected to the packet-switched network, we get Voice over IP over Wireless (VoIPoW). This introduces problems that are associated with wireless communication; transmission over air-links simply is not as reliable as over wired links. By reliable we mean that there will be more errors in the transmissions.

For VoIPoW to take over it has to show functionality and services that must be at least as good as todays circuit-switched systems. Also, the costs involved must be the same as today or lower. However, to reach these objectives, several problems relating to security, efficiency and reliability have to be faced (see Section 4.1).

1.2 Multimedia sessions

When two or more participants are connected in some sort of information-exchange in real-time, we say that they participate in a session. For instance, a regular phone call is regarded as a session with only one media type, audio that is.

There is nothing special about audio data, such as voice, once it has been transformed into a packet. Hence, there is nothing obstructing us from also sending video data and any other application-specific data over the same network. When coupling several different types of media in one and the same session, we get a multimedia session. The typical example being a phone also showing real-time pictures of the calling parties.

Since the nature of media data is stream-like, the data is said to be a flow or a stream.

1.3 3G networks and UMTS

Currently used systems for cellular phones are circuit-switched networks. This implies that multimedia sessions over these systems have to allocate large amounts of resources at dial-up time, blocking these from other users throughout the session. By introducing packet-switched networks the resource-usage will drop. Third generation (3G) networks are networks designed according to this strategy. Furthermore, what in particular distinguishes 3G networks from earlier generations is how the communication with the wireless terminals is handled, and the speed reached.

It is a great advantage if all systems use the same kind of packets. Since IP is already commonly in use in the computer industry, it is beneficial to use IP in this case as well. Following this idea we are going towards what is called "IP all the way", i.e., IP up to the terminal.

Universal Mobile Telecommunications System (UMTS) is a system of the third generation. It will be able to provide transmission speeds of 2 Mbit/s under ideal conditions. With a 2 Mbit/s transmission speed it is possible to stream video and audio simultaneously from one terminal to another in realtime. It is hoped that UMTS will be a world-wide standard, allowing compatibility between a large body of terminals and services. UMTS is standardized by the Third Generation Partnership Project (3GPP). 3GPP is a standardization body for specifications concerning the third generation mobile terminals. It consists of representatives from several large tele-communication companies.

Although UMTS specifies packet-switched networks, it will still be backward compatible with the older circuit-switched networks.

So, what are the benefits of switching to 3G networks? The reduced usage of resources allows users to run more resource demanding applications, for example multimedia applications. The high speed of the data transfer also favors for more complex and advanced applications. Also, using IP packets makes it possible to connect to the Internet and access services there.

1.4 Security

A person calling to find out whether to pick up any groceries on the way home from work might not be so concerned with the risk of someone eavesdropping on the conversation. On the other hand, for a lawyer speaking to a client, it might be devastating if someone picks up any information from the call. Therefore it should be possible to protect the session from interference and eavesdropping by unauthorized parties. It should be optional though, since the protection mechanism probably will be associated with a charge and it is not always needed as illustrated in the beginning of this section. The protection can be viewed as an additional service that can be bought when needed.

This report will discuss what is required for the protection of multimedia sessions, which are very demanding in terms of efficiency, describe a solution meeting the demands as closely as possible and evaluate an implementation of the solution.

Computer security is today extensively based on mathematics. Therefore the reader is assumed to have knowledge of basic concepts in discrete mathematics, probability theory, and complexity theory. We will not give detailed definitions and theorems, but refer the reader to texts devoted to the subject.

1.5 Objectives

The objectives of this report are described using some technical terms that are perhaps not known to the reader. These terms will be explained later on in the report. It is quite difficult to describe the objectives without these terms.

- Examine requirements for encrypted data that is transmitted in a wireless and packet-switched environment.
- Study the proposal for UMTS security (3GPP's f8-mode of operation[1]) and investigate if it fits multimedia sessions based on RTP[2].
- Study the Kasumi and AES/Rijndael block ciphers which may be used in f8-mode.

[1] f8 is a way of achieving encryption (see Section 3.7).

[2] RTP is a protocol for transmission of realtime streams (see Section 2.2.3)

- Implement and evaluate encryption of RTP-streams and integrate the implementation in Ericsson's testbed for VoIPoW simulation.
- Investigate the implications of using authentication and integrity checks of the media streams[3].
- Find a proposal for the authentication/integrity problem, implement it and do simulations.
- Discuss and search for problems and scenarios for the key-management problem for multimedia sessions on thin clients. Propose a basis for a discussion of a solution to the problem.

[3]Authentication and integrity checks are made to assure that the origin and contents of a message are correct (see Section 3.9)

Chapter 2

Data Transmission

This chapter describes some fundamental concepts of networks, protocols, data encoding, and transmission in general.

2.1 Networks

A loose definition of a *network* is two or more terminals connected to each other in some way. Seen this way, two computers connected by a person physically moving a floppy-disk between them, is a network. This kind of network is however not very interesting. We will in the remainder of this report only consider two kinds of networks, circuit-switched and packet-switched.

Switching points should in the following be interpreted as points where the network can chose to direct the incoming data to one or more outgoing lines. Machines, e.g., mobile phones and computers, are called *terminals* or *hosts*. The two terms will be used interchangeably throughout the thesis.

It might be clearer if the network is viewed as a graph $G = (V, E)$, where the vertices V are terminals and switching points, and the edges E are the connections between them.

2.1.1 The OSI model

Dealing with networks requires some sort of models to describe them. These models usually partition the network into *layers*, spanning from the actual hardware (bottom) to the applications (top) running on the hosts. A main feature of layered models is that each layer should be independent of the other layers.

Each layer should provide a specific and well defined service to the next layer above. The services are defined in protocols. One of the most commonly used models is the OSI model. OSI is an acronym for Open Systems Interconnection and was introduced by International Standards Organization (ISO). The OSI model consists of seven layers: Application-, presentation-, session-, transport-, network-, data link- and physical-layer. We will not go into the functionality of each layer, but refer to [33]. However, we will briefly describe the network and transport layers, since these are the most important ones for the remainder of the thesis.

On the network layer, a machine decides what to do with an incoming message. It is decided whether the message should be passed to a higher layer in the machine, or to which other machine the message should be passed. The decision is based on the address contained in the message.

The transport layer is responsible for dividing the data from higher layers into pieces and address them correctly. The address is usually accompanied by a *port* number. The reason for using ports is that a machine may want to run several applications, then each application is assigned a specific port through which it communicates. In this way, the received messages can be passed to the correct application in the higher layers.

When OSI was introduced, there were protocols associated with each layer. These protocols never made it big. They lost in competition to protocols from another model, the TCP/IP-model. On the other hand, the OSI model is better to use when discussing networks, so the status today is that the OSI model is used for theory and the TCP/IP protocols are used in practice.

2.1.2 Circuit-switched networks

Telephone systems of today are based on *circuit-switched* networks. Networks of this type have the distinguishing feature that when a terminal tries to connect (placing a call) to another terminal, a route between them through the network is determined and the rest of the communication will follow this predetermined route. Once the route is established there is no risk of delays caused by congestion at the switching points, data delivered out of order or non-delivered data (unless the connection is cut, of course). If there is not enough bandwidth to establish a connection between two terminals, it will be detected at setup-time and no connection will be made.

2.1.3 Packet-switched networks

As opposed to circuit-switched networks, *packet-switched* networks do not determine the route in advance. To transmit data over these networks, the data is chopped into pieces (called *packets*) which are then routed to the destination. The main difference here is that all the packets need not travel along the same route, which is the case for circuit-switched networks. Some implications of this strategy are listed below.

- There is no setup time before data can be transmitted.
- Bandwidth occupied by a transmission can change dynamically, which is not possible for circuit-switched networks.
- Switching points may be overloaded and hence packets might be lost if many packets try to pass the same point.
- Packets may arrive in a different order than in which they were sent, because they travel along different routes.

2.2 Protocols

When transmitting data over a network the use of standard protocols makes it easier for all participants to communicate.

This section will give a brief overview of the protocols used throughout the report. For an introductionary book on networks we refer to [33].

Some basic concepts that will be used are the following.

- A *packet* is a chunk of data transmitted between two hosts.
- A *header* is meta-data for a packet and can include things such as address of sending host, address of receiving host, and checksums.
- A *payload* is what remains of a packet after removing the header. The payload usually contains the application data, such as audio samples.

2.2.1 Internet Protocol

The *Internet Protocol* (IP) is one of the most well known and used protocols for packet delivery service and is defined in [30]. It is applied at the network layer. When writing IP in the rest of the report, we will mean the fourth version of the protocol, IPv4. In the future the successor IPv6, will most probably take over.

IP is connectionless, which means that it treats each packet independently of all other packets. IP does not know, nor care which packets that, in some sense, are associated in a higher layer. A host receiving an IP packet checks if the packet is addressed to the host itself. If that is the case the packet is passed to a higher layer (after some processing). If the host finds that the packet is addressed to someone else, the packet is just passed on to the part of the network where the other host resides.

IP is unreliable, that is, one cannot be sure that packets arrive in the same order as they were sent, or even arrive at all. Packets that disappear during transmission are said to be dropped.

Packets can be dropped due to several reasons. In the IP-header there is a field keeping track on how long a packet has been in transit, the TTL-field. TTL is an abbreviation for “time to live”. This value is decremented as the packet is routed between hosts. When it reaches zero the packet is discarded. This is to prevent packets from looping around forever. There is also a risk of bit-errors appearing in the packet during transmission, especially if it is routed over unreliable media such as a radio link. IP packets contain a checksum to protect the header. If a bit-error is introduced in the header during transmission this checksum will fail and the packet will be dropped.

The services provided by IP makes it suitable for transmission of audio and video, where the loss of some packets is not a major disaster. Modern codecs (see Section 2.3) can usually produce reasonable output from damaged data with the aid of good error concealment units. However, for transmission of executable binary files, cryptographic keys etc., where a single corrupted bit would be fatal, higher layers and protocols, such as TCP, must provide for retransmission of dropped packets, caring for error-robustness of the payload etc. The *Transmission Control Protocol* (TCP) is a protocol is a protocol that can be run on top of IP and makes sure that data does arrive at the destination and in the correct order.

2.2.2 User Datagram Protocol

The *User Datagram Protocol* (UDP) resides on the layer above IP, the transport layer. The design of UDP is made in the spirit of non layer violation, i.e., it does not rely on the structure of any other protocol in the protocol stack. The UDP header includes source and destination ports, the length of the UDP packet, and an optional checksum. The header is eight bytes long. The checksum is disabled by setting all the bits in the checksum field to zero. The checksum protects the entire header (including a pseudo header only visible to the protocol stack itself). The checksum field is set to zero during the checksum computation. UDP is defined in [29]. UDP adds the ability to demultiplex streams to different applications and the possibility to detect errors that might have appeared during transmission.

2.2.3 Real-time Transport (Control) Protocol

The *Real-Time Transport Protocol* (RTP) is especially designed for transmission of data with real-time characteristics. It is defined in [32] and fits in the session layer. RTP is independent of the underlying protocols, and adds no reliability properties. The aim is rather to provide for fast delivery of data and data description.

It is possible to define RTP profiles, which describe the data transmitted. The profile determines how packet data is encoded and interpreted. It may also specify other parameters that are needed by a particular application.

Media streams carried by RTP are called RTP-streams.

There is an accompanying protocol to RTP, the *Real-Time Transport Control Protocol* (RTCP). RTCP is used to monitor data and for minimal control and identification functionality.

An RTP-session is defined by the destination transport addresses and ports used for the RTP- and RTCP-streams. A multimedia-session is defined as a collection RTP-sessions.

`V`	`P`	`X`	`CC`	`M`	`PT`	`sequence number`
`timestamp`						
`SSRC identifier`						
`CSRC identifiers`						
`...`						

Figure 2.1. The RTP packet header.

The fields in the RTP header are shown in Figure 2.1 and will here be briefly described.

version (V) 2 bits Indicates the version number of RTP used; currently 2 is the latest version.

padding (P) 1 bit If the padding bit is set, there are additional padding octets in the end of the packet.

extension (X) 1 bit If the extension bit is set, the fixed header is followed by precisely one header extension.

CSRC count (CC) 4 bits Number of CSRC identifiers that follow the fixed header.

marker (M) 1 bit The marker bit is defined by the RTP profile in use.

payload type (PT) 7 bits This field determines the format of the payload and how it should be interpreted by the application.

sequence number 16 bits This field is simply the sequence number of the packet. The initial sequence number should be chosen randomly rather than start at zero.

timestamp 32 bits The timestamp indicates the time when the first octet of the payload was sampled.

SSRC 32 bits The Synchronization SouRCe is a randomly chosen number that is unique for each source within the RTP session. It is used to identify the streams.

CSRC list 0 to 15 items, 32 bits each Contributing SouRCes list. This list includes the sources contributing to the packet. The list is assembled by mixers that merge signals from several sources into one. Assume a mixer receives left and right audio streams from two microphones. It mixes these two streams into one, and transmits the mixed audio stream to a receiver. In this case the SSRC of the left audio source and the SSRC of the right audio source are listed in the contributing sources list.

2.2.4 IETF

The Internet Engineering Task Force (IETF) is a standardization organization, that standardizes protocols and other things related to the Internet. IETF is responsible for IP, UDP, RTP and many other protocols. They are also the inventors of several important security protocols, which we will encounter in Chapter 4 and Chapter 6.

Anyone is allowed to submit a draft to IETF, describing something that they want to become a standard. The draft must be presented at an IETF-meeting. These meetings occur three times a year, at different locations. If a draft is considered as reasonably good and describes something that is considered as important, it enters the state of an Request For Comments (RFC) proposed standard. After that it is further examined, and later (possibly after modifications) becomes a standard for the Internet.

2.3 Codecs

Before transmitting audio or video over a channel it has to be encoded in some way. Devices that provide the service of encoding and decoding are called *codecs* for short. They may be implemented in software as well as hardware.

The naive way of doing this (just using raw samples) may be sufficient if the medium over which the data is transmitted is reliable and there are no tight restrictions on bandwidth. This is unfortunately not the most common case. For instance, cellular phones and radios use highly error prone air-links. The data received is often not exactly the same as the data sent; errors have been introduced during transmission. Upon detection of errors, there could be a request for retransmission. In real-time applications there usually is no time for retransmissions.

To accommodate for this the codecs use different schemes to repair damaged data and produce output that resembles the original as closely as possible.

Error-correcting codes *Error-correcting codes* are used to insert as little redundancy as possible in the data, but enough to recover from a predetermined number of bit-errors. The mathematics involved is, for some types of codes (e.g., Goppa-codes), quite sophisticated.

Interpolation It is even possible to recover from packet-loss in a graceful way, if not too many packets are dropped. The idea is to use interpolation. The codec produces the output that in some sense is the most appropriate way to connect the samples at each side of the gap.

Unequal error protection Some portions of a sample may be more important than others. For instance, if the amplitude shifts of a sample-interval vary only slightly, not all samples in that interval are equally important to reconstruct an approximation of the original. Even though some of the bits are lost, it is still possible to hear what is said, although the sound may be somewhat "metallic". This is exploited in *unequal error protection* (UEP); some portions of the data might be protected by error-correcting codes and others are just thrown away when containing errors.

Comfort noise Codecs sometimes intentionally make the sound "worse" by inserting noise where there is none from the start. This is because people in general are used to some background noise when using telephones, especially during periods when no one is speaking. That absence of noise sounds strange to the human ear, it may sound as if the line is dropped. The artificial noise is known as *comfort noise*.

2.4 Data compression

To reduce bandwidth it is essential to minimize the size of the packets transmitted. Packet compression also has an impact on packet loss, since bit-errors are less likely strike small packets. Bit-errors will cause packets to be dropped if they are protected by a checksum.

Compression can be described as finding a way to describe something in a more compact way. For example, the sentence "the letter z one thousand times" is a far more compact description than the actual sequence of one thousand z.

One point that will be of importance later in the report is that a randomly generated string cannot be expressed in a shorter way than the string itself, i.e., random strings are not compressible.

When talking about compression in this section we mean lossless compression. Lossless compression schemes are able to reconstruct the data compressed to its original form exactly. This is in contrast to lossy compression schemes which do not reconstruct the original data exactly.

Lossy compression schemes usually give a better compression ratio than lossless ones, but are only suitable to pictures, sound and other media where exact decompression is not essential. When compressing packets, it is crucial to be able to decompress (at least) the header of the packet losslessly. For instance, if the field containing the IP-address was allowed to be decompressed in a lossy way, the packet might very well end up at the wrong destination.

Error propagating compression schemes Compression can be achieved using well known schemes such as LZ77 [35] and its clones. The basic idea of these schemes is to detect strings of data that appear more than once and replace all but one of these strings with a reference to that one. This type of compression schemes are however not very suitable for a wireless environment due to the error propagation they cause.

Other important compression techniques suffering from the same error propagating properties are Huffman coding and arithmetical coding. For a description of these, see any introductory text on coding theory.

Header compression Another approach is to remove the redundant information from the packet header, leaving the payload as it is. For audio/video data, the payload is usually compressed by the codec. This is known as *header compression* and pays off when the header is large compared to the payload. This is usually the case for real-time audio/video data. Typically the payload of an audio packet is 33 bytes and the IP/UDP/RTP header is typically $12 + 8 + 20 = 40$ bytes. Hence there is more meta-data used for administration than there is actual audio data. Fortunately, there are many fields in the header needed to be transmitted only once per session (such as the version number of RTP, ports etc.). These fields will be called static. Other fields only need to be transmitted once in a while (e.g., when changing audio encoding of a mediastream, the payload type field in the RTP header must be changed accordingly). This type of data will be referred to as semi-static data below. There are also fields that change in a predictable way, e.g., the RTP sequence number, and they do not need to be sent in the full every time.

Algorithms taking advantage of these properties can reduce the size of the average packet header to virtually nothing. For instance the ROHC algorithm [6] reduces the 40 byte header mentioned above to only 2 bytes on average. When there is a need for transmission of semi-static data the header will be less compressed of course, but since this only happens occasionally the overall gain is huge.

Delta-compression There is also the possibility to use delta-compression of the headers. Delta-compression schemes only transmits the difference between the previous header and the current one. This is only applicable to the header in the case of audio/video, since the payload contains samples which will vary a lot from one packet to the next. This variation between packets would force the delta-compressor to re-transmit almost the entire packet, resulting in negligible gain compression-wise and an over all loss in speed due to the overhead. Delta-compression of the headers should heuristically give approximately the same compression ratio as ROHC-like schemes, but recovery from packet loss becomes more cumbersome.

Conclusion The most profitable type of compression for RTP-streams is the header compression approach. In particular, the ROHC-scheme is a proposed standard in IETF and has shown to be working at an interoperability test performed by several companies.

Chapter 3

Cryptographic Background

We will in this chapter give an informal introduction to the basic building blocks of cryptography needed in the rest of the report. The purpose is not to supply rigorous mathematical definitions, but rather to explain informally the topics and make posed statements believable. Note that details are left out and sometimes we only give "half the truth". For a very good and precise introduction to cryptography we refer to [26]. Another readable book focusing on algebraically based cryptosystems is [19]. We assume the reader is familiar with elementary probability theory and discrete mathematics. The terms infeasible and difficult should be interpreted as "not polynomial-time computable" where the polynomial is a polynomial-function in the length of the input viewed as a binary string. Sometimes the polynomial-function will be defined in the input viewed as an integer written in base 2. For example, consider the ordinary multiplication algorithm for two integers with n decimal digits. Using this algorithm, the product is computed by n^2 multiplications and $n^2 - 1$ additions of integers in the interval $[0, 9]$. Consider simple opertations on integers in this interval, such as addition and multiplication, as operations that can be done in one time unit. Then the cost of a multiplication of two n-digit decimal integers can be expressed by the polynomial $p(n) = 2n^2 - 1$. Usually, only the term of highest degree is of importance in theoretical discussions, and we say that $p(n)$ is *ordo* n^2, denoted $p(n) \in O(n^2)$. If a problem is not polynomial-time computable, it means no algorithm exists that solves the problem using polynomially many simple operations. See [27] for more rigorous definitions and results in complexity theory.

3.1 Brief introduction to cryptography

Cryptography can naively be described as the science of keeping traffic over a public channel secret from others than the involved parties. The basic functionality is to provide an *encryption function* which scrambles the traffic in some way, and a *decryption function* which restores the traffic to its original form. The scrambled traffic should of course resemble random noise as closely as possible. There are also other properties needed to secure a channel, such as assuring that the ones involved in the communication really are who they claim to be (authentication), detection of tampered traffic (integrity) etc.

Nowadays these issues are resolved using techniques from various fields of mathematics, such as probability theory, computational theory, number theory, and many more.

When talking about cryptography the two communicating parties are historically called Alice and Bob, and a person trying to get access to their communication is usually called an attacker, enemy or adversary.

3.2 Keys

A *key* of a cryptosystem is something used to encrypt or decrypt (or both) a message in a way unique to that key. The main idea is that many users should employ the same cryptosystem, but all have their own set of keys. In the simplest case two people communicate with each other; sharing the same (secret) key makes it possible for them, but no one else, to encrypt and decrypt the messages sent between them.

One might suggest that it would be more secure to use a secret cryptosystem, only known to the ones using it rather than having a publicly known one. What speaks against this is that very few organizations have competence enough to develop a secure cryptosystem. Publicly known cryptosystems on the other hand, usually have undergone massive cryptanalysis by a large number of top researchers and any flaws are hence likely to have been detected. We must also be aware of that if we use a secret cryptosystem where the strength lies in that the system is secret, an adversary might get his hands on it and apply reverse-engineering. If this happens, the whole cryptosystem must be replaced, which is a much harder task than just replacing a set of keys. This suggests that a well known cryptosystem should be used, and that the security of that system should be based on the secrecy of keys and *not* on the cryptosystem being secret and obscure.

So, the strength of the cryptosystem should be based on the difficulty of determining the correct key. What properties should the keys have then? For starters there must be so many possible keys that trying every single one is an infeasible task. This type of attack is called a *brute-force attack*. The set of all possible keys for a cryptosystem is called its *key space*. The key space must be large enough to prevent a brute-force attack. What is meant by a "large enough" key space is not quite clear. In 1999 a message encrypted using DES (Data Encryption Standard) was recovered by brute-force in 22 hours and 15 minutes with dedicated hardware [10]. This hardware was built to a price affordable to a quite small company. DES has 2^{56} valid keys, which consequently is not enough for long time security if the attacker can be assumed to have access to such equipment. Today it is usually recommended to have at least 128-bit keys. That is, there are 2^{128} valid keys.

When selecting a key from the key space, it is important that each key is equally probable to be chosen. That is, the probability that we have chosen a specific key should be $2^{-\ell}$, where ℓ is the number of bits in the key. One way to do this is to select ℓ truly random bits. Without getting into a philosophical discussion about the existence of true randomness, we state that producing large amounts of truly random bits is expensive. Time between detections of particles during radioactive decay, measured by a Geiger counter and air turbulence measured with high precision are believed to be good sources of true randomness. Often one has to settle for pseudo random bits as key material. Pseudo random bits are bits generated by some function, where the output of the function is difficult to predict on a short random input. A function that, given some small truly random input (called the *seed*) produces output that looks random is called a *pseudo random number generator* or a *pseudo random function*. That the output looks random, means that it should not be possible to distinguish the output from truly random data using any polynomial-time algorithm. Note that the seed has to be kept secret.

Using a pseudo random number generator for key generation will, however lead to that all keys in the key space are not equally likely to be selected.

3.3 Stream ciphers

This section provides an introduction to the basic concepts of stream ciphers. For a more detailed and precise discussion of stream ciphers we refer to [26].

A stream cipher is an algorithm that given a key k individually encrypts each character in a stream of data (the *messagestream*). Usually the character set is $\{0, 1\}$; thus the messagestream may be considered as an infinite string of ones and zeros. Since any message may be encoded as a string of ones and zeros, the encoding of the message is then fed as the messagestream to a stream cipher. The stream cipher produces a stream, consisting of characters from from the character set, called the *keystream*. The messagestream and keystream are then combined according to some rule to produce the *cipherstream*. Variations of feedback shift registers, SEAL and RC4 are some popular stream ciphers.

Stream ciphers can be divided into two main categories, *synchronous* and *self-synchronizing* (or *asynchronous*) stream ciphers. In synchronous stream ciphers, the keystream is generated independently of the messagestream and the cipherstream. In asynchronous stream ciphers the keystream is dependent on a fixed number of previous characters in the cipherstream. We will concentrate on the synchronous variant in the following. The reason for this is that asynchronous stream ciphers propagate bit-errors in the ciphertext. This is a property that is not acceptable in wireless environments.

One common approach is to design an algorithm that, given a key, produces a pseudo random sequence of ones and zeros, i.e., a keystream. This keystream is then bitwise added modulo two (i.e., XORed) with the datastream to produce the cipherstream (see Figure 3.1). This type of cipher is known as a *binary additive stream cipher* (BASC). The beauty of this construction is that encryption and decryption of data is the same operation.

There is a particular type of BASC that is theoretically unbreakable: the *one time pad*. A one time pad is a truly random string of characters of length n. If we XOR this one time pad with a message-string of length $\leq n$, the resulting cipher-string has a very fine property: it is impossible to determine the message-string or the one time pad given the cipher-string. An adversary that only knows the cipher-string will not be able to recover the message even by trying all possible key-strings. The result of trying all possible key-strings is actually all possible message-strings, but the adversary cannot tell which is the correct one. Unfortunately one time pads have some serious drawbacks.

- It is a tedious task to produce truly random strings.
- The one time pads must be known by both the sender and receiver, so there is a distribution problem.
- The pad needs to be as long as the message it protects.
- As the name suggests, they can only be used once without compromising security.

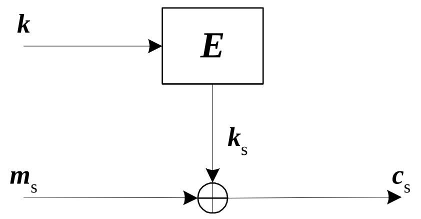


Figure 3.1. A typical stream cipher. E denotes the keystream generator and k the key. m_s, k_s and c_s denotes the messagestream, keystream and cipherstream respectively. Decrypt a cipherstream by interchanging the position of m_s and c_s.

There are properties of stream ciphers that make them especially suitable for encryption of wireless communications. For instance, if the keystream is generated independently of the message, bit errors that arise due to the transmission over unreliable media will only affect the erroneous bit. The ciphertext produced by a stream cipher is of the same length as the plaintext. The message-length is not expanded, which might be the case for block ciphers (see next section).

3.4 Block ciphers

This section provides an introduction to the basic concepts of block ciphers. For a more detailed and precise discussion of block ciphers we refer to [26].

By a block we will in the following mean a string of ones and zeros with a fixed length n, typically 64 or 128 bits. A block cipher is an algorithm that given a key k and a block p of plaintext returns a block c of cipher text. The blocks p and c are usually of the same length. Block ciphers in ECB-mode (see modes of operation below) can to some extent be viewed as stream ciphers using a very large character set. Some examples of block ciphers are DES, RC6 and the recently selected Advanced Encryption Standard (AES) named Rijndael [8].

When designing a block cipher there are several goals to aim for, apart from small code size and high processing speed. Some of them are

- Given a “large” amount of ciphertext blocks encrypted with the same key, it should be difficult to find the key or the plaintext. We leave the term “large” undefined, and refer to [26].
- The bits of the plaintext block and the ciphertext block should be uncorrelated to as large extent as possible (*confusion*).
- Even if two blocks of plaintext differ only by one bit, the two resulting ciphertext blocks should be almost uncorrelated (*diffusion*).

3.4.1 Modes of operation

If we want to encrypt a message consisting of more than one block using a block cipher there are several different methods, or *modes of operation*, to use. Some common modes of operation are explained below. We assume that the message is represented as a string of ones and zeros since any message may be encoded this way. In the figures describing the modes, the symbol $\oplus$ denotes the XOR-operation.

Even if a block cipher is ideal, in the sense that the only way to break it is a brute-force attack (see Section 3.8 below), using it in an insecure fashion will leak information. A secure way of using a cipher can be defined in several ways, one of which is *semantic security.* Semantic security is not only defined for modes of operation, but rather for any function. Informally it can be understood as follows. View the plaintext as the input to a function, and the ciphertext as the output from the same function. Anything that can be computed with knowledge of the ciphertext can also be computed without knowledge of the ciphertext. That is, knowing the ciphertext adds no information for an adversary. A formal definition of semantic, and other types of security can be found in [1].

ECB *(Electronic Code Book)* This is the most straight forward mode. Pad the message (see below) so that the length of it is a multiple of the block length n. Divide the padded message into pieces of length n and encrypt each piece using the block cipher. Finally concatenate the ciphertext blocks to form the encrypted message. Figure 3.2 shows the procedure.

OFB *(Output Feed Back)* When employing a block cipher as a generator of keystream data for a stream cipher, this mode and CFB mode (see below) are frequently used. It works as follows: chose an *initialization vector* (IV), which is a block. Then encrypt the IV and use it as the first n bits of the keystream. Now repeatedly encrypt the output from the block cipher and in this way receive more n bit chunks of keystream. Figure 3.4 shows the procedure.

CBC *(Cipher Block Chaining)* This mode works much like ECB. The difference is that before encrypting each piece, it performs a bitwise addition modulo two (i.e., taking XOR) with the previous ciphertext block produced. When encrypting the first block, the "previous" ciphertext block is chosen to be an IV. The mode is depicted in Figure 3.3.

CFB *(Cipher Feed Back)* This is a minor modification of OFB. Instead of feeding the output from the block cipher back the ciphertext block is feed back, as can be seen in Figure 3.5.

CTR *(Counter)* This mode takes an IV, encrypts it and uses the output as keystream, which is XORed with the plaintext. For the next block the IV is considered as an integer and is incremented by one and the procedure is repeated once again. Figure 3.6 shows the procedure.

Assuming that the underlying block cipher is a pseudo random function, CTR and CBC are provably secure in the semantic sense (see [1]). OFB can be shown to be semantically secure as well. On the other hand, ECB is not semantically secure. One way to see that is the following reasoning. Suppose

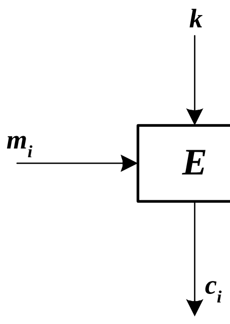


Figure 3.2. ECB mode, E denotes the encryption algorithm, k is the key, m is the message block and c is the cipher block. Decryption is performed the same way, just swap c and m, and replace the encryption algorithm with the inverse algorithm.

that a message consists of the block-sequence a, b, c, a. Then it can be noted by just inspecting the ciphertext that the first and the last block of the message are the same, since they both will encrypt to the same cipher-block when using ECB.

Different modes have different properties. In the following discussion on some properties of different modes, the block cipher in use is assumed to be a pseudo random function.

If ECB is used, the same plain text block will always encrypt to the same ciphertext block under the same key. This implies that patterns in the plaintext larger than the block length can be detected in the ciphertext, as seen above.

CBC can hide the patterns mentioned above, by changing the IV between encryptions. On the other hand, it introduces error propagation during decryption. As for all good block ciphers, a single bit-error in a cipher block causes the entire block to decrypt as garbage. Since the decryption is dependent on earlier blocks, this means that one bit-error causes two blocks to be corrupted. These two blocks are the only ones affected by the bit-error though. After the two erroneous decryptions the mode is decrypting correctly again.

OFB uses the block cipher as a key generator for a stream cipher. The way it is used, bit-errors in the ciphertext will only affect the corresponding bit in the plaintext (they will be inverted). Therefore, it has no error propagation. The loss of ciphertext will bring it out of synchronization though, and cause the rest of the message to decrypt incorrectly. Changing the IV between encryptions makes the same message encrypt to different ciphertexts. As will be described later, these properties can be used to get access to entry points to the cipher stream. Also notable is that both encryption and decryption use the block cipher in the same direction. This implies that only the encryption part of the block cipher needs to be implemented, and hence expensive chip-area can be saved in hardware implementations.

Like CBC mode, CFB creates dependencies between the ciphertext blocks, and thus letting a single bit-error in the ciphertext affect more than one block in the decrypted message. It is self-synchronizing, so that it can recover from bit-errors; but the error-propagating property makes CFB unsuitable for wireless environments.

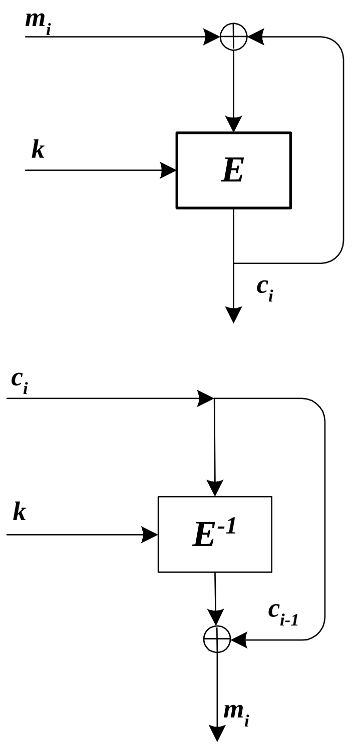


Figure 3.3. CBC mode, E denotes the encryption algorithm, k is the key, m is the message block and c is the cipher block. The picture on top shows the encryption procedure, where as the bottom picture shows the decryption procedure. The initial value of the cipher block c_0 must be agreed upon by the encryptor and decryptor.

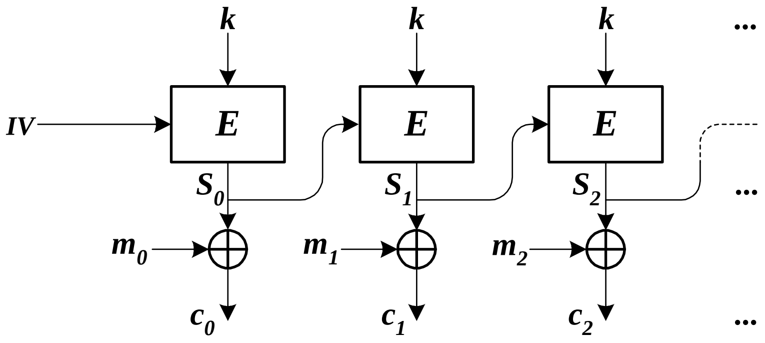


Figure 3.4. OFB mode, the IV, which does not need to be secret, is encrypted repeatedly using the key k. The cipherstream blocks c_i are constructed by XORing the keystream blocks S_i with the messagestream blocks m_i. The block cipher E can be any block cipher.

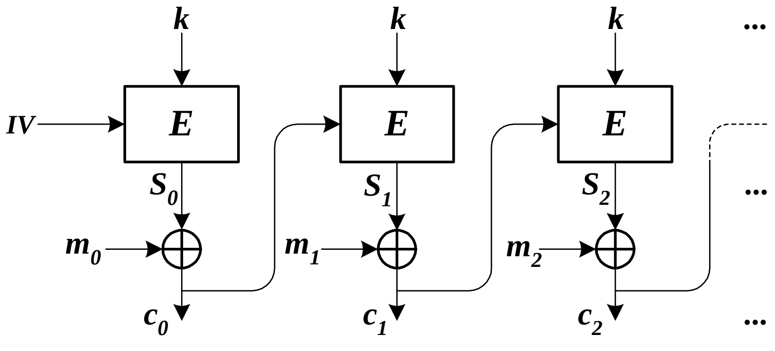


Figure 3.5. CFB mode, The mode works much like OFB (see Figure 3.4), but feeds the cipherstream blocks back instead of the encryption output-block.

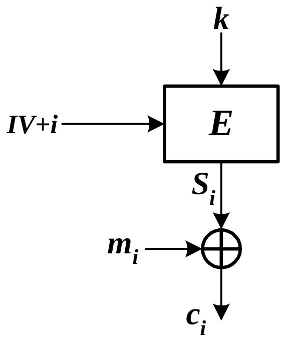


Figure 3.6. Counter mode, the IV added to the block-number in the progression of messagestream blocks is encrypted, using the key k. The cipherstream blocks c_i are constructed by XORing the keystream blocks S_i with the messagestream blocks m_i. The block cipher E can be any block cipher.

3.4.2 Message padding

If the length of a message is not a multiple of the block length used by the cipher the message must be padded for some modes of operation. One simple solution is to extend the message with zeros. This solution has the drawback that messages will not be unique. For example, the messages 101 and 1010 will be the same message when zero-extended. A common remedy for this problem is to store the length of the original unpadded message, in binary form, at the end of the padded message.

3.5 AES/Rijndael

As discussed in Section 3.2 the DES algorithm can no longer be considered secure. Therefore the National Institute of Standards and Technology (NIST) in January 1997 called for proposals for a new block cipher to replace it, the Advanced Encryption Standard (AES). From all submitted proposals 15 were selected for the first round. These were further analyzed, leaving five algorithms for the second and final round. Finally in October 2000, the winning algorithm was selected: Rijndael [8], invented by J. Daemen and V. Rijmen.

Rijndael supports key- and block-lengths of 128, 192 and 256 bits. The key-length and block-length can be specified independently. The structure of Rijndael makes it suitable for applications with tight memory constraints as well as applications requiring high speed, both in software and in hardware. With the use of the so called T-tables, encryption and decryption speeds of 80 Mbit/s were achieved by our C-implementation on a Pentium II with a clock frequency of 266 MHz; see Chapter 5 for a discussion of the implementation.

A common approach when designing block ciphers is to use a Feistel-structure. This includes using several rounds, where in each round the input block is divided in half, applying some function f to one of the halves and then concatenating them in reverse order, after applying an XOR, to produce the output block of the round (see Figure 3.7). The function f is usually dependent on a variable known as round key, which is different for each round. Examples of ciphers that use this structure are DES and several of the AES-candidates, e.g., MARS. Figure 3.7 shows a simple three-round Feistel-structure. Rijndael is based on totally different principles of more algebraic nature. The rounds in Rijndael are constructed from four basic operations: *ByteSub*, *MixColumn*, *ShiftRow* and *AddRoundKey* which are described below.

Rijndael has undergone thorough analysis by prominent cryptologists, including the National Security Agency (NSA), without revealing any weaknesses. Thus, Rijndael can be considered secure. However, it is noteworthy that the comments made by NSA during the standardization may have been kept secret, so we do not actually know if they discovered anything of importance.

The state Rijndael works with an internal state which can be viewed as a matrix with four rows and where the number of columns equals the block-length divided by 32. In this matrix every entry is one byte. The key can in a similar fashion be viewed as a matrix.

The bytes in the state are the polynomial-representations of elements in the Galois-field $GF(2^8)$ (using the standard polynomial-representation with $x^8 +$

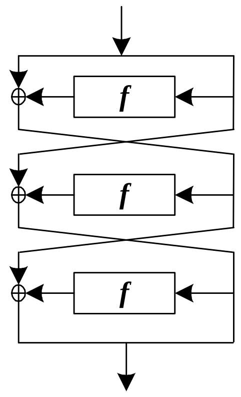


Figure 3.7. This is an example of a three-round Feistel-structure.

$x^4 + x^3 + x + 1$ as the reduction polynomial), where the bits represent the coefficients of the polynomial. For example, the polynomial $x^3 + x + 1$ will be represented by the number $(00001011)_2 = (0b)_{16}$.

The input block is, in the case of 128-bit blocks, viewed as a string of 16 bytes, $a_0, a_1, \ldots a_{15}$. These are mapped to the state as depicted in Figure 3.8. For the other block-lengths the principle is the same.

a_0	a_1	a_2	$\ldots$	a_{15}

$\downarrow$

a_0	a_4	a_8	a_{12}
a_1	a_5	a_9	a_{13}
a_2	a_6	a_{10}	a_{14}
a_3	a_7	a_{11}	a_{15}

Figure 3.8. The mapping from a 128-bit plaintext-block to the state matrix.

ByteSub The ByteSub operation is defined by the bijective mapping $s(x)$. The mapping $s(x)$ is constructed as follows: take the multiplicative inverse of $x \in GF(2^8)$ (zero is mapped onto itself), then apply a transformation (which is affine over $GF(2)$) to x^{-1}. The transformation can be found in [8]. The operation is most easily implemented by a 256-byte table containing the co-domain of the mapping. This kind of table is often called a substitution table (or S-box) with terminology borrowed from the specifications of DES.

ShiftRow The ShiftRow operation performs a circular shift of all rows in the state. Each row is shifted a different number of steps. One step is equal to one byte. The zero:th row is shifted zero steps, the first C_1 steps, the second C_2 steps and the third row is shifted C_3 steps, where C_1, C_2 and C_3 are constants that depend on the block-length.

MixColumn The MixColumn operation treats the column-vectors of the state as polynomials over $GF(2^8)$. The "column-polynomials" are multiplied modulo $01x^4 + 01$ with the polynomial $c(x) = 03x^3 + 01x^2 + 01x + 02$. The polynomials $c(x)$ and $01x^4 + 01$ are relatively prime (this can be checked by applying Euclid's algorithm), implying that $c(x)$ has a multiplicative inverse in the polynomial ring $GF(2^8)[x]/\langle x^4 + 1\rangle$

Multiplication by the inverse of $c(x)$ cannot be implemented as efficiently as multiplication by $c(x)$. Therefore decryption is not as fast as encryption.

AddRoundKey This operation is a simple XOR between the bytes of the state with the corresponding bytes of the expanded key (see key-schedule below).

The round One round of Rijndael is composed as follows:

```
ByteSub (state)
ShiftRow (state)
MixColumn (state)
AddRoundKey (state, roundKey)
```

The final round is slightly different in that the MixColumn operation is left out. The operations ByteSub and MixColumn can be combined into four tables, denoted T_0, T_1, T_2 and T_3 in [8]. Using these tables a round can be performed with only four table look-ups and four XORs per column of the state. During the table look-up, the ShiftRow operation shows up in the indexing.

Key-schedule The key supplied to Rijndael is 128, 192 or 256 bits long. Before encryption can take place this key needs to be "stretched" in order to cover a different round-key for each round. This will of course not add any extra entropy, but still makes it harder for an adversary to break the cipher. The stretching mechanism can be viewed as a pseudo random function taking the original key as input and producing more key material as output. The key is stretched using the key-schedule.

The key-schedule expands the key in the following way for 128-bit blocks. Each round-key is derived from the previous one. A round-key is viewed as a matrix, with the same dimensions as the state-matrix. Assume we want to derive the round-key $(\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$, where $\mathbf{b}_i$ is the i:th column-vector of the round-key-matrix. Furthermore, assume that $(\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3)$ is the previous round-key-matrix. Then $\mathbf{b}_0$ is derived by applying a function f to the XOR between $\mathbf{a}_0$ and $\mathbf{a}_3$, as shown in Figure 3.9. The function f is defined in terms of ByteSub, a rotation of the elements of the vector, and some constants. The vector $\mathbf{b}_1$ is derived as the XOR between $\mathbf{a}_1$ and $\mathbf{b}_0$, as depicted in Figure 3.10. The vectors $\mathbf{b}_2$ and $\mathbf{b}_3$ are derived as the XOR between $\mathbf{a}_2$ and $\mathbf{b}_1$, and $\mathbf{a}_3$ and $\mathbf{b}_2$, respectively. The operations are depicted in Figure 3.11 and Figure 3.12. As a general rule, the column vectors of a round-key are constructed as follows

$$\begin{array}{lcll} \mathbf{b}_0 & = & f(\mathbf{a}_0 \oplus \mathbf{a}_n) & \\ \mathbf{b}_i & = & \mathbf{a}_i \oplus \mathbf{b}_{i-1} & \text{, for } i > 0, \end{array}$$

where n is the number of columns in the state-matrix.

The first round-key is the original key, written into a matrix, in the same manner as when the plaintext block was mapped to the state (see Figure 3.8).

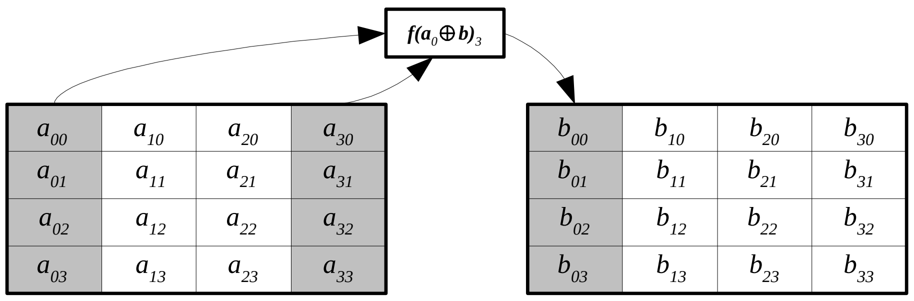

$f(a_0 \oplus b)_3$
a_{00} a_{10} a_{20} a_{30}
a_{01} a_{11} a_{21} a_{31}
a_{02} a_{12} a_{22} a_{32}
a_{03} a_{13} a_{23} a_{33}
b_{00} b_{10} b_{20} b_{30}
b_{01} b_{11} b_{21} b_{31}
b_{02} b_{12} b_{22} b_{32}
b_{03} b_{13} b_{23} b_{33}

Figure 3.9. Construction of the first column-vector of a round-key.

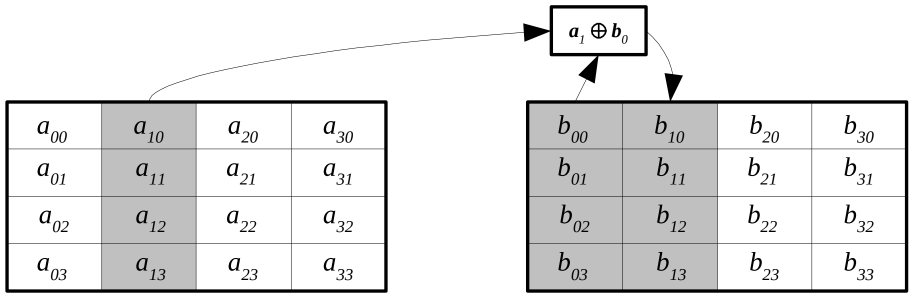

$a_1 \oplus b_0$
a_{00} a_{10} a_{20} a_{30}
a_{01} a_{11} a_{21} a_{31}
a_{02} a_{12} a_{22} a_{32}
a_{03} a_{13} a_{23} a_{33}
b_{00} b_{10} b_{20} b_{30}
b_{01} b_{11} b_{21} b_{31}
b_{02} b_{12} b_{22} b_{32}
b_{03} b_{13} b_{23} b_{33}

Figure 3.10. Construction of the second column-vector of a round-key.

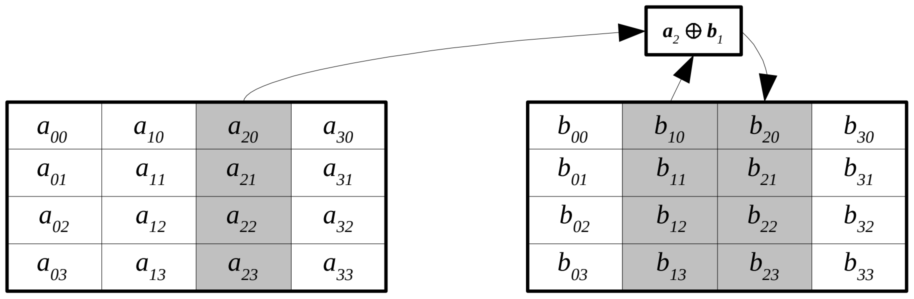
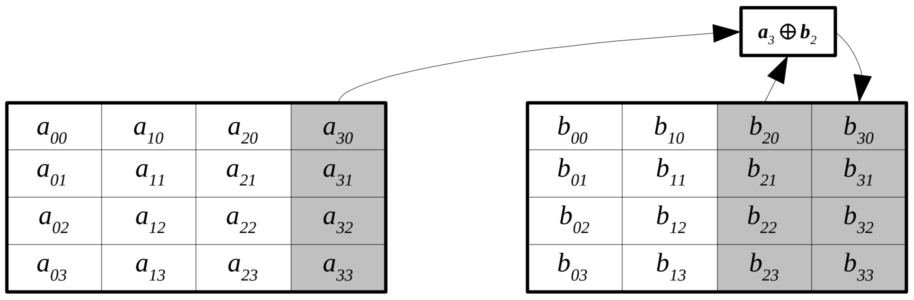

$a_2 \oplus b_1$
a_{00} a_{10} a_{20} a_{30}
a_{01} a_{11} a_{21} a_{31}
a_{02} a_{12} a_{22} a_{32}
a_{03} a_{13} a_{23} a_{33}
b_{00} b_{10} b_{20} b_{30}
b_{01} b_{11} b_{21} b_{31}
b_{02} b_{12} b_{22} b_{32}
b_{03} b_{13} b_{23} b_{33}

Figure 3.11. Construction of the third column-vector of a round-key.



$a_3 \oplus b_2$
a_{00} a_{10} a_{20} a_{30}
a_{01} a_{11} a_{21} a_{31}
a_{02} a_{12} a_{22} a_{32}
a_{03} a_{13} a_{23} a_{33}
b_{00} b_{10} b_{20} b_{30}
b_{01} b_{11} b_{21} b_{31}
b_{02} b_{12} b_{22} b_{32}
b_{03} b_{13} b_{23} b_{33}

Figure 3.12. Construction of the fourth column-vector of a round-key.

For key-sizes 192 and 256, this scheme is generalized. The key is written into a matrix that is large enough to contain the key. Then the key is stretched in almost the same manner as for 128-bit keys, until there is enough key-material to cover all rounds. There is a small modification of the above recursion formula for 256-bit keys, and that is that for the column-vector $\mathbf{b}_4$, an additional ByteSub operation is applied.

If the key-size differs from the block-size, the same recursion formula is used, but it starts to apply at the first empty column-vector. For example, assume we use 128-bit blocks and a 192-bit key. Then the first round-key consists of the first 128 bits of the key, and the first 64 bits (i.e., the first two column-vectors) of the second round-key consists of the remaining bits of the key. The last two column-vectors of the second round-key is then determined by the recursion formula above.

3.6 Kasumi

In 1997 Matsui proposed a new block-cipher, Misty [24]. Among the design criteria posed when creating Misty, Matsui put provable resistance against differential and linear cryptanalysis[2, 23] (actually linear cryptanalysis was invented by Matsui himself). The block-length is 64 bits and the key-length is 128 bits.

Misty uses a sort of recursive Feistel-structure; the function applied to one of the halves is a Feistel-structure in itself. This recursion is three levels deep and is very similar to the one in Figure 3.13. The recursive property of Misty makes it easy to analyze the cipher and to prove resistance against differential and linear cryptanalysis.

The European Telecommunications Standards Institute (ETSI)/Security Algorithms Group of Experts (SAGE) needed a block-cipher for the 3GPP confidentiality and integrity standard [11] to be used in UMTS, for circuit-switched voice. What they did was to modify Misty and rename it Kasumi (which means misty in Japanese). The main modifications were made to reduce the number of gates in hardware implementations. The overall high suitability for implementation in hardware was inherited from Misty. This is a very important property since the algorithm was issued for thin clients, such as cellular phones. There was also a modification of the key-schedule made, leaving it very simple but never the less seemingly robust.

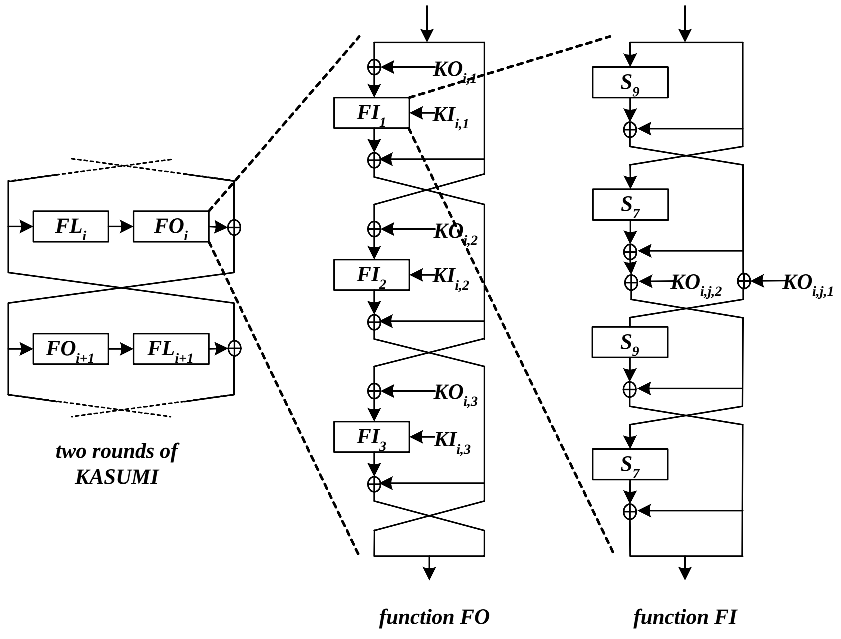


Figure 3.13. Simplified picture of the recursive Feistel-structure of Kasumi.

Figure 3.13 shows a simplified picture of the structure of one round of Kasumi. Symbols beginning with a K are different sub-keys produced by the key-schedule and symbols beginning with an F are functions. The function FL (not shown in the figure) is a very simple function composed of AND-gates, OR-gates and circular shifts. The boxes S_9 and S_7 are substitution boxes with input/output of 9 and 7 bits respectively. The symbol $\oplus$, is used for the XOR operation, that is, bitwise addition modulo 2.

As can be seen in Figure 3.13 there are (recursive) calls to smaller Feistel-structures, causing the effect of the substitution boxes to escalate. The recursive structure also makes the analysis of the cipher and proving resistance against attacks easier. The use of all key-bits in every round prevents meet-in-the-middle-attacks (see Section 3.8).

There are no currently known attacks against either of Misty or Kasumi.

3.7 f8 mode of operation

The f8 mode of operation is a variant of OFB developed by 3GPP and is defined in [11]. It uses a block cipher to produce a keystream for a stream cipher. The structure of f8 can be seen in Figure 3.14.

The most important difference between f8 and OFB is the way f8 hides the initialization vector (IV) from an adversary. The original IV is encrypted using a special key, and the result of that encryption called IV' (which is hard for an adversary to obtain without knowledge of the key) is used internally. The other property differentiating f8 and OFB is the counter present in f8.

The core of f8 is the block cipher Kasumi, with a 128-bit key and 64-bit block-length. There is however nothing special about Kasumi in this setting, so any block cipher E can be used in its place.

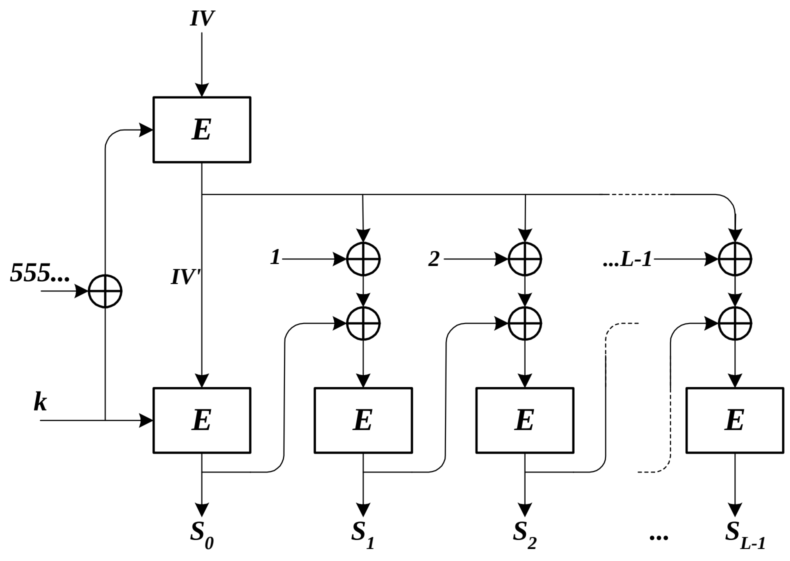


Figure 3.14. f8 mode of operation.

3.7.1 Structure

In the following || denotes concatenation and the i:th b-bit block of an object is denoted by subscript, e.g., bits $4b, 4b+1 \ldots 5b-1$ of the cipherstream C are denoted C_4.

Let E be a b-bit block cipher taking a key k, and let $E_k(x)$ denote the b-bit output block when encrypting a b-bit string x with the key k. The value m is the bitwise XOR of k and 0x555..., that is, as many fives as needed to match the key-length. Furthermore, let IV be the initialization vector. The output keystream is divided into chunks of b bits, where the i:th chunk is named S_i. Assuming we want to encrypt a n-bit plaintext P, let $L = n/b$ be the number of keystream chunks needed to produce enough keystream.

Initialization Before encryption can take place the keystream generator f8 has to be initialized. This is done by calculating $m = k \oplus$ 0x555... and putting appropriate values in IV. What to put in IV is defined in [11], but that particular choice of values is not applicable in all situations. Then IV' is calculated by $IV' = E_m(IV)$.

keystream generation The keystream S is defined as $S_0||S_1||S_2 \ldots ||S_{L-1}$ where

$$\begin{array}{rcl} S_0 & = & E_k(IV') \\ S_i & = & E_k(IV' \oplus i \oplus S_{i-1}) \quad , 1 \leq i \leq L-1 \end{array}$$

Encryption/decryption The cipherstream C is then defined as $C_i = S_i \oplus P_i$, the ordinary XOR-operation. Thus, f8 is an additive stream cipher. Since XOR, as noted, is an involution, decryption is accomplished in the same way, $P_i = S_i \oplus C_i$. If the length of the plaintext is not a multiple of the block-length, there will be more keystream bits than plan-text bits. These additional bits are discarded.

3.7.2 Properties

Obviously f8 mode of operation, along with OFB, does not have error-propagating properties. Inclusion of the block counter ensures that the keystream will not produce a cycle before at least 2^b blocks have been encrypted. The encryption of the IV makes it hard for an adversary to get known pairs of input/output in case he possesses knowledge of the IV and some plaintext. This reason for this scheme is that the constructors wanted “both belt and suspenders”, if the block cipher used turned out to have flaws.

3.8 Attacks

The most obvious attack against a block cipher is the brute-force attack. An adversary has access to some known plaintext, the corresponding cipher text and knowledge of which encryption algorithm was used. The adversary simply encrypts the plaintext with every possible key, comparing the output from the algorithm with the known ciphertext, until the correct key is found. This task is computationally infeasible if the key space is large enough.

There are several attacks possible against block ciphers other than the brute-force attack. Two of the most powerful ones applicable to general block ciphers are linear and differential cryptanalysis, which are based on statistical methods. For instance, with the use of linear cryptanalysis DES can be broken with 2^{47} known plaintext/ciphertext pairs [23]. DES is broken in the sense that the attack is faster than a brute-force attack.

Another attack that can be mounted, if the cipher is not carefully constructed, is the meet-in-the-middle attack. It requires a known plaintext block and its corresponding ciphertext block. If the cipher only uses certain bits of the key in early rounds, and others in later rounds, this attack can be performed. Assume that the cipher under attack has n rounds, a k-bit key and that both n and k are divisible by 2. Furthermore, the cipher uses the $k/2$ first bits of the key for the first $n/2$ rounds and the the rest of the key for the remaining rounds. In the first step the attacker (who knows one plaintext block and the corresponding ciphertext block) runs the cipher on the plaintext block for $n/2$ rounds under all the possible $2^{k/2}$ possible keys. The resulting blocks are stored in a table together with the key used. In the second step, the attacker runs the inverse of the cipher on the ciphertext block for $n/2$ rounds for each of the

$2^{k/2}$ possible keys. When the $n/2$ rounds are done for a key, the resulting block is looked for in the table created in the previous step. If the block is found, the key is retrieved by concatenating the key from the table with the one used in step two. Otherwise the next key is tried. The attack can be performed in only the square root of the time used by a brute-force attack, with additional storage. This example is simplified, but the principle should be clear.

3.9 Message authentication and integrity

In many cases it is important for a receiver of a message to be able to verify the origin of a message. This is in cryptography known as message authentication. Proving the origin of a message might, from a philosophical point of view, be impossible. Instead, the authentication problem is usually solved by proving beyond reasonable doubt that the message origin is the one claimed. This can be done by, for instance, providing the result of some computation that requires the knowledge of a secret that is supposedly only known by the sender and receiver. It is in this case important that the result of the computation is infeasible to obtain without knowledge of the secret. Message authentication codes (which will be discussed in Section 3.10) is a popular and commonly used method to provide message authentication.

It is also important to be able to discover whether an adversary has tampered with a message while it was in transit. That is, it is important to protect the integrity of the message. The simplest example of an attack is the case when an adversary changes the answer to a question from yes to no. With the growth of wireless networks, interception and manipulation of messages becomes an increasing risk. The standard way of ensuring the integrity of a message is to append the result of some function that is dependent on the message itself and a secret, as in the case of message authentication.

In a peer-to-peer scenario, authentication is equivalent to integrity. But in a group scenario it may not be. Imagine a group-call, where every participant uses the same key to encrypt/decrypt the audio. Since everyone has access to the key, it is not possible to determine who in the group sent a particular message. In the peer-to-peer case there are only two involved parties, so if the message did not originate at your own end, it must have been the other person sending it.

3.10 Message authentication codes

In this section, the communication is assumed to be between two parties, i.e., we do not consider a group scenario.

As mentioned above, one solution to the authentication problem is to use a *message authentication code* (MAC). A MAC is a code derived from some secret shared by the communicating parties. The MAC is then typically appended to the message before transmission.

A nice way of solving both the authentication and integrity problem at the same time, in the peer-to-peer scenario, is to compute a *keyed hash-value* or *keyed message digest* of the message. This hash-value is then used as a MAC and is transmitted together with the message. The MAC can then be checked by the receiver.

A hash-function is a function h from the space M of all messages to some space H of all hash-values producible by h. For all practical purposes $|h(x)|$ is fixed, implying that h "compresses" all inputs with length larger than $|h(x)|$. Note that this implies that more than one message will have the same hash-value; h is not bijective.

The authentication and integrity is based on that the hash-value only can be computed by someone who is in possession of the secret key. That is, it should be difficult to forge a MAC. To achieve this, cryptographically secure hash-functions must be used.

There are several constraints that can be put on these functions. The properties we will need in this report is the two following.

- Given $h(k||m_1)$ for a random k it should not be feasible to find $m_2 \neq m_1$ such that $h(k||m_2) = h(k||m_1)$. That is, *collisions* should be hard to find.

- Computing $h(m)$ for random $m \in M$ is relatively easy, but computing $h^{-1}(h(m))$ should be computationally infeasible. This notation shall not be interpreted as h having a unique inverse in the normal sense, but rather as h having a set of elements in the domain which are mapped onto a particular element in the co-domain, and no member of that set should be possible to find efficiently. Functions having this property are called *one-way functions.* In spite of the fact that it still is an open problem whether one-way functions actually do exist [27], the cryptographic hash-functions used today have proven difficult to invert.

In other situations than the ones we will consider, there are other properties that are needed.

Todays most commonly used hash-functions are SHA-1 and MD5 [31, 12]. These functions produce hash-values of length 160 and 128 bits respectively. To produce a MAC using SHA-1 or MD5, a key need to be incorporated in the message that is hashed. This can be achieved by prepending the key to the message as above.

3.11 Public key cryptography

Public key systems What has been discussed so far is known as symmetric key cryptography. The reason for this that the same key is used both to encrypt and decrypt. This is however, not a necessity. In 1976 Diffie and Hellman wrote a paper [9] (today, probably one of the most influential papers on cryptography) about the use of asymmetric keys, initiating the era of public key cryptography.

The main idea with public key cryptography is that all users have their own public and private keys. If Alice wants to send a message to Bob, she encrypts it with his public key which can be looked up in some type of telephone book. Then using his private key, only known to Bob, he decrypts the message. One crucial thing to note here is that anyone can encrypt a message using Bobs public key, but only Bob can decrypt the message, using his private key. The advantage of public key cryptography is that a message can be securely transmitted without the parties sharing a secret.

One problem with this approach is to find encryption and decryption functions that are compatible in the sense just described. What is wanted is a one-way function that actually can be inverted with the knowledge of some information (the private key). This type of functions are sometimes called trapdoor functions.

There are two major mathematical problems that form a basis, on which most public key systems are based: factoring of large integers and finding discrete logarithms in finite groups.

Factoring is the problem of finding the unique prime factorization of a given integer. Mathematically we can write this: given an integer x, find $p_1, \alpha_1, p_2, \alpha_2, \ldots p_n, \alpha_n$, where p_i are prime and $\alpha_i \geq 0$ are integers for all i such that $p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdots p_n^{\alpha_n} = x$.

The discrete logarithm problem is, given a finite group G, a base $b \in G$ and a $y \in G$, to find a positive integer x such that $b^x = y$. The name "logarithm" comes from the similarities with the continuous counterpart of the problem.

These two problems have proven to be difficult, that is, although a large number of mathematicians have tried to find efficient algorithms for them for a very long time, no one has yet succeeded.

Examples of public key systems based on the difficulty of these problems are RSA (based on the factoring problem) and ElGamal (based on the discrete logarithm problem).

A drawback of existing public key systems, not shared with symmetric key systems, is that the operations involved are computationally expensive. Therefore, public key systems are mostly used to transfer keys, which are then used with traditional symmetric techniques.

Diffie-Hellman key-exchange One idea, described in [9], is particularly aimed at the exchange of keys. This idea is known as the Diffie-Hellman (DH) key-exchange. What the DH-exchange accomplishes is that two persons can agree on a key, only known to them, over an insecure communication channel.

The DH-exchange works as follows. Alice and Bob agree on a finite group G, and a generator g for that group (or in practice, an element of high order). Both the group and the generator can be publicly known. Then Alice picks an integer x at random, computes g^x and sends g^x to Bob over the insecure channel. Now Bob picks an integer y at random, computes g^y and sends g^y to Alice. Finally, Alice computes $(g^y)^x = g^{yx}$ and Bob computes $(g^x)^y = g^{xy}$. The secret key shared between the two is g^{xy}.

Assuming that discrete logarithms are difficult to find, an adversary only knowing g^x and g^y faces a seemingly infeasible task if trying to deduce x or y to compute g^{xy}. There might be other ways for the attacker to compute g^{xy}, but today none are known.

There are no requirements on which finite group to use (except to avoid a few "degenerate" cases). In addition to the obvious choice $\mathbb{Z}_p^*$ (the multiplicative group of integers greater than zero and less than p, where p is prime), groups such as elliptic curves over finite fields [19] and the recently discovered XTR-groups [22] can be used, although the latter have shown some weaknesses.

Public key infrastructure Although public key systems solve the key distribution problem, they suffer from another weakness relating to the keys. That is, how can Alice be sure that the key she thinks is Bob's really is his?

There are several different solutions to this problem, and it all boils down to who you can trust. For instance, assume we have a network connecting the population of a country. Then the government distributes one *certificate* to each individual. The certificate cryptographically ensures that the public key contained in it belongs to the particular individual, in a publicly verifiable way. The government is what we call a *certification authority*, or *CA* for short. Now, if Alice trusts the CA, she can take Bob's certificate and check Bob's key. The connections and trusts between entities in a network is called a *trust model*.

If there is a need to connect the networks of all the countries on a continent, there are now several CAs in the larger network. To remedy this, one possibility is to create a top CA, which certifies the CA of each country. Note that this is only one way to solve the problem. A structure of CAs is called a *public key infrastructure* (PKI).

The use of CAs, chained like this, is not flawless. An enlightening example is that Mother Theresa might issue a certificate to Charles Manson, but you would not normally trust someone showing a certificate signed by Mr. Manson.

Chapter 4

Security for Multimedia Sessions

This chapter discusses a proposal for encryption of end to end (e2e) multimedia communication over error prone networks described in [5]. The requirements for the proposal where investigated in [3].

4.1 Requirements

Encryption of multimedia sessions over wireless networks poses some special constraints on the mechanisms used. A thorough analysis of the requirements for a security protocol for data transmitted over an air-link using IP can be found in [3]. This section will summarize part of what is stated in that draft.

Bit error rate Due to the fact that air-links introduce bit errors at a higher rate than wire-links, the encryption algorithm used must not be error-propagating. For instance, for some modes of operation on block ciphers, a single bit error in the ciphertext results in that an entire block decrypts incorrectly.

Efficiency The terminals in an e2e solution may be computationally weak, e.g., cellular phones. Therefore, complicated computations should be avoided.

The nature of real-time sessions also puts efficiency constraints on the encryption scheme. Encoding and decoding of the mediastreams are time consuming as they are, and the encryption scheme should add as little as possible to that burden.

This implies that the encryption scheme must only involve simple and fast computational components.

Unreliable networks Networks based on IP do not guarantee that packets are delivered in order or that packets will arrive at all. Most applications can, however, handle these situations. Thus, the encryption scheme must cope with this as well. That is, there must be some sort of "fast-forward/rewind" mechanism present, to be able to decrypt packets that are delivered out of order.

In the case stream ciphers are used, it must be possible to retrieve a specific portion of the keystream quickly.

Bandwidth Cellular access links are expensive. Thus the size added to the packets by the encryption, in form of padding etc., should be minimized.

Header compression To reduce bandwidth consumption, header compression is used (see Section 2.4). The encryption scheme must not interfere with this compression.

Unequal error protection As mentioned in Section 2.3, a packet might be protected by unequal error protection to reduce bit errors. Hence, the bits of the payload are divided into classes which are protected differently depending on their importance.

The encryption scheme must make sure not to break the independence between these classes.

4.2 Authentication and integrity

Data integrity and authentication can be employed whether confidentiality is an issue or not. However, they possess particular features, which will be pointed out in the following, that make their use in wireless multimedia sessions inconvenient.

As mentioned in Section 3.10, keyed hash-functions are commonly used to achieve authentication and integrity of messages (or packets in this case). These cryptographically secure hash-functions produce typically hash-values of 128 or 160 bits. For a typical audio packet of length 320 bits (using header compression) this implies that bandwidth consumption will increase by a factor 3/2. This will reduce the number of possible users per cell by the same factor and is unacceptable for a cellular network service-provider. The cost for additional cells to achieve the same capacity as without the use of MACs will be too large to be attractive. Reducing the MACs to bandwidth compliant length, say 16 bits, gives little or no security, since that short MACs are easy targets even for brute-force-attacks.

The use of keyed hash-functions or checksums in general make the packets sensitive to bit-errors. It is impossible to distinguish between bit-errors as a result of transmission over an unreliable radio link or bit-errors inflicted by an adversary. Therefore the action usually taken when a checksum or MAC fails is to discard the packet to be on the safe side; the result being a lower sound quality.

Again, the length of the MACs relative to the length of the packets leads to trouble. A packet containing a perfectly intact sample may be discarded due to the MAC itself being struck by a bit-error and thus will fail.

A bit-error will, as stated in [34], occur with a probability of order 10^{-3} on a radio-link. Assume that 128-bit MACs are used and that 320-bit audio packets are transmitted. In this scenario about 45% of the packets will be dropped due to failing MACs; the sound quality will be unacceptably low. Without the use of MACs, packets containing bit-errors would be fed to the codec, which would be able to produce a reasonable output from the damaged data. It should be noted however, that the bit-errors tend to come in bursts rather than having a uniform distribution. This is in favor of the use of MACs, but still the packet-drop rate will be much too high.

Although data integrity and authentication are important concepts and might be desirable features, they pose difficult problems and are too expensive to be practical. Under these considerations it is probably best to leave the implementation of MACs to the application and not force it at lower network layers. In this way applications that really need authentication and integrity and can deal with the high packet-loss rate may implement MACs at will.

4.3 Applicability of other protocols

There are standardized protocols that potentially could be used for securing IP/UDP/RTP-streams. However, they suffer from deficiencies that make them unsuitable in the scenarios we are considering. This section will discuss the two most obvious choices, and why they are not applicable.

4.3.1 IPsec

There exists a solution that provides security for IP, called IPsec [18, 17]. The security offered by IPsec is good, but does however have some properties not suitable for multimedia over wireless links.

A major problem with using IPsec in a wireless environment using UDP/RTP is that it encrypts also the UDP and RTP headers. Since encrypted data can be considered as random, the header compression scheme will fail to work (see Section 2.4). This results in poor utilization of the bandwidth. As stated in [3], this will lead to higher costs, and hence be less attractive. The headers cannot be compressed before encryption, since the compressor resides on a lower layer in the stack than the other protocols.

Another issue is the usage of authentication with the aid of MACs in IPsec. The MACs are 96 bits long, and would increase the size of a typical audio packet by roughly one half. Again, the bandwidth consumption goes up, and furthermore the packets are easier targets for bit-errors (see Section 4.2).

IPsec also adds additional bits for administration (in the form of headers), which consumes bandwidth as well.

Even though IPsec provides good security, it is too expensive to be used in all applications we foresee. Thus, there is a need for a security mechanism that is cheaper. The next sections discuss work on such security mechanisms.

4.3.2 (W)TLS

The WTLS (Wireless Transport Layer Security) protocol is a protocol designed for securing transmission over wireless networks [36]. It is the wireless version of the TLS (Transport Layer Security) protocol. (W)TLS is, just like IPsec, a protocol well suited for the purpose for which it was designed. There are, however, several points that make (W)TLS inappropriate or impossible for encryption of wirelessly transmitted RTP-streams. Today, TLS implementations are commonly targeted at securing TCP. Even though the standard allows other protocols than TCP, it can only be used to protect reliable protocols. Since UDP is unreliable (see Section 2.2.2), TLS cannot be used to secure IP/UDP/RTP streams. WTLS on the other hand, can handle being run over unreliable protocols. Unfortunately WTLS comes with a built in key-management scheme. It is a desirable feature of an encryption scheme to be detached from the key-management, since different types of key-management schemes are suitable for different situations.

4.4 Ericsson's proposal

At the IETF (Internet Engineering Task Force) meeting in November 2000 Ericsson participated with the drafts [3, 5]. Given the requirements above, [5] proposes the use of either Kasumi or Rijndael in f8-mode for the encryption of the RTP-packets. Preferably Rijndael should be used, for the sake of the length of the IV (recall that Kasumi has 64-bit blocks, and Rijndael has 128-bit blocks). There are also more "political" reasons for choosing Rijndael, for instance it is believed to take over from DES as *the* standard block cipher. Furthermore, since Kasumi is standardized in 3GPP it might not be well received by IETF.

4.4.1 Meeting the requirements

The following assumptions are made in [5]

- Key negotiation has successfully been accomplished with the aid of some protocol.
- The RTP-headers are received without any errors or lost data.
- RTP is run over UDP/IP.
- The session is unicast. That is, the transmission goes from one source to one receiver.
- The crypto application has knowledge of the UDP-ports used.

The f8-mode meets the requirements of no error propagation, since one error in the ciphertext only implies that the bit in error will be affected during decryption. Since f8-mode acts like a stream cipher, the packets will not be enlarged, and hence the bandwidth usage is preserved. The way the IV is formed (see below), f8-mode may enter the cipherstream at the starting points of the packets in time $O(1)$, taking care of dropped packets and out of order delivery.

The choice of Rijndael and Kasumi is motivated by the fact that they with high probability will be present in the hardware of cellular phones in a not to distant future. Also, f8-mode will be implemented in future mobile terminals. This results in availability and high-speed implementations.

4.4.2 Stream encryption

Since we are dealing with packet-switched networks, the streams will be broken into packets. To encrypt the stream, each RTP-packet must be encrypted individually. The headers are left in the clear. That is, they are not encrypted. Leaving them in the clear makes header-compression possible. Recall from

Chapter 3 that encrypted data should be as close to random data as possible and from Section 2.4 that random data is not compressible. Although leaving the headers in the clear might open up for attacks, [5] can only see the possibility of denial of service (DoS)-attacks. It is also noted that an adversary by inspecting the header, can determine the encoding-scheme of the data and then flip individual bits. At the moment this will only have DoS-effects, but in future RTP-profiles this can lead to more serious attacks.

The scheme is dependent on the uniqueness of the tuple (SSRC, port). If this pair is not unique, it is possible for several datastreams to have the same keystream. This property of the (SSRC, port) tuple is left to the implementors to obtain. RTP guarantees that the SSRC is unique inside a RTP-session by the collision detection built into RTP. The use of the port number will ensure that each stream originating from the same source (e.g., left and right channel audio) will obtain a unique keystream. However, the tuple is not unique inside a multimedia session, since the multimedia session may consist of more than one RTP-session. Figure 4.1 shows a situation where two RTP-streams could end up with the same keystream. Therefore it is important that implementors ensure that the tuple is unique in the multimedia session. One way to achieve the uniqueness is to let the key-management assign SSRCs to each participant joining the session. The SSRC is the unique number that identifies each source in an RTP-session.

Alice		Bob
SSRC1	$\longrightarrow$	PORT1
PORT2	$\longleftarrow$	SSRC2
PORT1	$\longleftarrow$	SSRC1

Figure 4.1. A situation where the tuple (SSRC, port) is not unique. The two streams at the top belong to one RTP-session, and the single stream at the bottom belongs to another.

4.4.3 Construction of the IV

The IV for f8 is formed by concatenating certain fields from the RTP-header (concatenation is denoted by $||$). Most importantly, by using the SSRC and port number, the stream is guaranteed to be unique for each stream belonging to the same multimedia session. The other fields from the RTP-header are entered to achieve an implicit authentication. That is, an adversary cannot change these fields, since the packet will then decrypt incorrectly. When Rijndael is used, the IV is constructed as follows

```
IV = port-number || SSRC || SEQ || TS || M || PT || 00...0
```

where the port number is a 16-bit destination UDP-port, SSRC is the 32-bit synchronization source, SEQ is the 16-bit sequence number, TS is the 32-bit time stamp, M is the marker bit, PT is the 7-bit payload type and the remaining 24 bits are zero. If Kasumi is chosen as block cipher, then the IV only has 64 bits and is defined in the following way

```
IV = port_number || SSRC || SEQ
```

4.4.4 Conclusions

Ericsson's proposal for protecting the multimedia sessions is targeted at confidentiality of each RTP-stream individually. This is accomplished under the stated assumptions.

There is no authentication of the streams, since this turned out to lower the quality of the data to an unacceptable level for voice and video.

The requirement that the session is unicast is not necessary, since it is never used. Therefore, the proposal scales to multicast sessions as well.

4.5 CISCO's proposal

At the IETF meeting in November 2000, also CISCO submitted a draft [25], concerning the encryption of multimedia-sessions. The ideas are very similar to [5].

CISCO propose that Rijndael is put in a slightly modified counter mode to produce the keystream. The difference between CISCO's counter mode (CCM) and ordinary counter mode is that CCM uses fields from the RTP-packet to reinitialize the counter when faced with a new packet, whereas ordinary counter mode would keep on adding one to the IV.

By not hiding the IV by an extra encryption, CCM perhaps trades security for speed, compared to Ericsson's proposal. On the other hand, under the assumption that Rijndael is secure, counter mode is provably secure (see Section 3.4.1). It has not been rigorously proven that CCM is secure, but the difference between CCM and counter mode is very small.

Notable is also that [25] proposes that part of the header (the CSRC-list, if present) should be encrypted, and that the SSRC should be used to uniquely define the keystream. Using only the SSRC as a unique identifier, can lead to two datastreams ending up with the same keystream. For instance, imagine a stereo CD-player with left and right outputs going on different streams A_l and A_r. If the SSRC identifies the CD-player, then A_l and A_r will use the same keystream.

The proposal uses what is known as a sliding window. The window consists of a lower and upper bound on the sequence numbers. The bounds slide in synchrony during the encryption, hence the name sliding window. This technique is used to prevent replay attacks. That is, if an adversary records some packets and tries to replay them some time later in the same session, they will be discarded because their sequence number will be outside the window.

Included in the proposal is also optional short MACs for authentication. As discussed in Section 4.2, this is not very suitable for wireless environments.

Furthermore, the proposal also covers RTCP, but Ericsson's proposal can easily be extended to cover RTCP as well.

4.6 A merged proposal

Since [25] and [5] were targeted at the same problem and showed resemblances, CISCO and Ericsson decided to cooperate and merge the two ideas into one. The name of CISCO's proposal (Secure Real-time Transportation Protocol abbreviated SRTP) was inherited by the merged one. It was decided that Rijndael run in CCM mode should be mandatory and that f8-mode should be optional.

Furthermore, it was decided that the IV should contain a 32-bit roll-over counter, which keeps track of how many times the RTP-sequence number has wrapped around. This counter makes it possible to encrypt 2^{48} packets, without reusing the exact same keystream. The IV changed, and currently has the following format:

```
IV = ROC || FLAG || M || PT || SEQ || TS || SSRC
```

The `FLAG` is an 8-bit field used to differentiate between RTP and RTCP. The value of `FLAG` is 0 for RTP and 1 for RTCP. It can also be used for future expansions of the protocol.

Kasumi was dropped, because an IV of 64 bits is not large enough to accommodate the roll-over counter.

To be able to quickly retrieve different security parameters for each stream, an optional *Security Parameters Index* (SPI) was discussed. The SPI is a 32-bit value, that is used to locate the the crypto-context in a database. The crypto-context used to process each RTP-stream is associated with the SPI carried in the packets belonging to that stream. However, the people working with header compression claimed that there would not be support for compression of this new field. Furthermore, the benefits of using the SPI did not match the complexity it added to the protocol, so it was left out.

In a group scenario, where all members of the group use the same key, authentication becomes troublesome. It is not enough to use MACs, since anyone in the group has access to the key. If it is necessary to be able to determine who in the group sent a particular message, a stronger form of authentication is needed, *source origin authentication.* There have been voices in IETF asking for such authentication, and at the IETF meeting in August, a revision of SRTP was presented with such facilities. The scheme used (see TESLA [28]) is not yet a standard, and hence the specifications might still change. Some people in IETF seem to be in a hurry to get SRTP standardized. Because of that, the source origin authentication will be left out in the next version.

SRTP will go to last call in November 2001. That means that the draft is announced as final. If no one objects, it will become a Request For Comments (RFC), which is one step closer to being a standard.

The functionalities and properties of SRTP are still under discussion, but the list below shows a snap-shot of the status at the time of writing.

- **ciphers** AES/Rijndael (mandatory to implement)
- **authentication algorithm** HMAC (SHA1) (optional)
- **modes of operation** CCM (default), f8 mode
- **authentication** optional
- **replay protection** yes (if authentication is used)
- **error-robust** yes (using stream ciphers)

- **robust re-keying** no (if more than one stream uses the same key, their sequence numbers may be out of synchronization. Since the new key is supposed to be used from a certain sequence number on the stream defined as the master stream, there might be some packets that use the wrong key at the time of change.

Chapter 5

Implementation Results

This chapter describes the results from the implementation phase of the project. The Implementation was carried out in two steps. First the algorithms were implemented on a PC workstation. They were then integrated in a VoIPoW test-bed.

5.1 Algorithms implemented

We implemented two block-encryption algorithms and put them into f8 mode of operation. The block-encryption algorithms were Kasumi and Rijndael.

5.2 Language

There were several reasons for selecting C as the implementation language.

- Since the intent of the code was to supply encryption in real-time environments, speed was of uttermost importance.
- We had access to a C-compiler with one of the most efficient optimizers on the market (Microsoft Visual C++ v6.0).
- C was the language best known to us, and is widely used in the Ericsson organization.
- The ciphers were to be incorporated with ECSeL (The Ericsson Crypto Software Library), which is written in C.

We also implemented Kasumi in x86-assembler, but portability issues were stronger than the gain of increased speed. Also, a trick due to Matsui using a large table, made the assembler implementation obsolete. The large table-driven version is clearly not suitable for a thin client, but works well on an ordinary PC.

5.3 Platforms

The code was originally written on a PC running Windows NT. Some speed optimizations were dependent on the processor endianess, but both the Kasumi and Rijndael ciphers included a macro which allowed for deciding endianess at compile time.

When moving the code to different machines (SPARC) and different operating systems (Solaris, RedHat Linux, FreeBSD UNIX, Windows 2000, etc.) it turned out that the endianess-switch caused a problem. The problem was that the initial version assumed that the macros `BIG_ENDIAN` and `LITTLE_ENDIAN` were not both defined at the same time. That was however, the case on some systems.

Otherwise, since the code was written as closely to ANSI C as possible, there were no major difficulties porting it.

5.4 Performance figures

The reference platform, on which the figures in this section were produced, was a Pentium II, 266 MHz running Windows NT. The speed fluctuated slightly when measured, so the numbers in the tables are averages. The fluctuations were caused by properties of the processor, such as caching effects for code and data, adaptive branch-prediction, etc. The caching effects were reduced using cache-warming though.

Both Kasumi and Rijndael were implemented in several versions with trade offs between speed, size and functionality.

Kasumi Table 5.1 shows the speed, of the Kasumi implementation in ECB mode. For Kasumi the speed is measured in Mbit/s, where $M = 2^{20}$. It is a bit surprising that the encryption and decryption speeds differ when look-up tables are used in place of the FI function. The reason for this is still unclear. One explanation may be that there is some difference in speed between encryption and decryption, and that difference is small enough not to show when the throughput is only around 22 Mbit/s.

Table 5.1. Encryption/decryption speeds of Kasumi in Mbit/s.

version	Encryption	Decryption
no tables	22.3	21.9
tables	42.5	38.5

Rijndael The Rijndael cipher was implemented in four different versions. Two of them use the T-tables, whereas the other two do not. In two of them, it is possible to switch key- and block-length in run-time, whereas in the other two, those lengths are determined once and for all at compile-time. Versions using the T-tables are called "large" and are denoted in the performance tables Table 5.2, Table 5.3 and Table 5.4 with an l. The other version are called "compact" and are denoted by a c. The versions that are able to switch key- and block-lengths in run-time are denoted f for "flexible".

The asymmetric structure of Rijndael explains why the versions that are not table-driven show such a big difference between encryption and decryption.

Flexibility turned out to slow down the natural extension of the table-driven version. There is room for optimizations of this, but we did not have time to polish the code within the time frame of the thesis project. One point that severely affects the speed is the modulo calculations required for the ShiftRow operation. Our compiler fails to find the common sub-expressions that appear in this particular case, and hence much of the work is duplicated. These modulo computations should be pre-calculated to gain speed. Furthermore, when the block-length is 128 bits, the modulus is 4 and can be computed as a bitwise AND with 3 instead of performing a division.

The speed is measured in Mbit/s, where M $= 10^6$ rather than 2^{20}. This choice of M was made to achieve performance figures comparable to the ones in [8] by Gladman. A comparison, made on our reference platform, between our implementation and Gladman's shows that his is slightly faster; encryption by his implementation is done at a rate of 86.2 Mbit/s and decryption at 86.0 Mbit/s. These figures should be compared to the ones in Table 5.2, using the large version and a 128-bit key.

Table 5.2. Encryption/decryption speeds in Mbit/s of Rijndael, using 128-bit blocks.

version	key-length = 128		key-length = 192		key-length = 256	
	enc.	dec.	enc.	dec.	enc.	dec.
l	80.3	83.0	68.1	70.1	59.0	60.5
lf	8.3	9.0	7.0	7.5	6.0	6.5
c	21.6	15.5	18.2	13.0	15.7	11.1
cf	18.5	14.0	15.5	11.7	13.4	10.1

Table 5.3. Encryption/decryption speeds in Mbit/s of Rijndael, using 192-bit blocks.

version	key-length = 128		key-length = 192		key-length = 256	
	enc.	dec.	enc.	dec.	enc.	dec.
l	75.9	69.2	76.0	68.6	65.6	59.7
lf	5.3	6.1	5.4	6.2	4.5	5.0
c	17.4	11.9	17.4	11.9	15.1	10.2
cf	16.2	10.6	16.2	10.6	13.9	9.10

f8-mode of operation We implemented the f8 mode of operation, using the AES/Rijndael as underlying block cipher with 128-bit block length and 128-bit key length. The output keystream was used to encrypt RTP-packets. Encryption-speed was measured for both the compact and large version of Rijndael, denoted Rijndael_c and Rijndael_l respectively and are shown in Table 5.5. In that

Table 5.4. Encryption/decryption speeds in Mbit/s of Rijndael, using 256-bit blocks.

version	key-length = 128		key-length = 192		key-length = 256	
	enc.	dec.	enc.	dec.	enc.	dec.
l	66.8	66.9	66.6	66.1	66.2	67.0
lf	4.4	5.0	4.4	5.0	4.5	5.0
c	15.6	10.0	15.6	10.0	15.6	10.0
cf	13.7	9.5	13.7	9.5	13.7	9.5

table are also the corresponding figures for the non-table based version of Kasumi for comparison.

When measuring the encryption-speed of the packet encryption we used simulated RTP-packets with a typical 96 bit header (which was not encrypted) and a 256, 512 and 1024 bit random payload. With this scenario we need one block encryption to set the IV, and two, four and eight block encryptions respectively, to produce the keystream for each packet. This heuristically implies that the f8 algorithm should be able to encrypt somewhere around 16.5 Mbit/s for 256 bit payloads using the compact version of Rijndael. The actual throughput will be a bit less because of some overhead though. Since encryption and decryption is the same operation, the same figures hold for decryption as well.

As the packet-length increases the initial encryption of the IV accounts for a smaller portion of the total number of encryptions. Therefore, the encryption-speed will increase as well.

Kasumi outperforms the compact version of Rijndael due to the shorter block-length. The shorter block-length implies that less overhead is introduced by construction and hiding of the IV.

Estimation of speeds when using CISCO's counter mode, yields somewhere around 30-50% better figures than f8-mode.

Table 5.5. Encryption/decryption speed in Mbit/s for packets with different payload-lengths, using Rijndael and Kasumi in f8-mode.

block-crypto	length = 256	length = 512	length = 1024
Kasumi	17.7	19.7	20.7
Rijndael_c	14.1	17.0	18.9
Rijndael_l	50.2	61.0	68.2

Calculation of the internal IV (copying of some data from the packet header and one block encryption) takes around 0.006 milliseconds. Key-refresh and initialization of the keystream generator (we view initialization as a key-refresh from a non-existent key) takes approximately 0.012 milliseconds. These figures are valid for the compact version. Due to a bit more complex key-schedule in the large version the times will be a bit larger. The key-schedule for the large version includes some pre-computation to increase encryption speed.

5.4.1 Conclusions

The measurements of the implementations suggest that the speed of the proposal will not cause any problems. Remember that UMTS will provide speeds of approximately 2 Mbit/s, and these results indicates that the encryption will not a bottleneck.

5.5 Test-bed integration and simulations

The implementation for encryption of RTP-packets using Ericsson's proposal was integrated in Ericsson's test-bed, which is used to simulate VoIPoW. The testbed is a network of computers, where each computer simulates a certain component in a VoIPoW scenario. Several VoIP applications can be run in the testbed and the produced data is subject to bit-errors, packet-drop, packet-delay etc., to simulate the air-link. The parameters determining characteristics of the air-link can be altered in real-time. The testbed makes it possible to monitor and measure performance of these applications and of the network as a whole.

To measure the performance of the encryption scheme and compare it to the de facto-standard IPsec, a VoIPoW session was simulated. The basic scenario was that a mobile terminal, e.g., a cellular phone, sends an audio RTP-stream over a wireless link to a base-station. The base-station is connected to a fixed network, e.g., the Internet. The RTP-stream is forwarded to a terminal residing in the fixed network, where the sound is played on the terminals speaker. Figure 5.1 shows a picture of the simulated network.

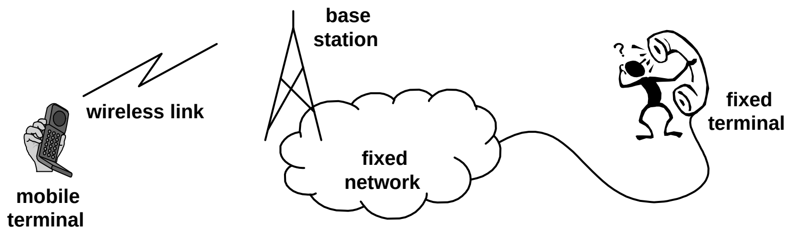


Figure 5.1. This is the simulated scenario, where a mobile terminal places a voice-call to a fixed terminal.

The RTP-stream was then encrypted, using the IPsec implementation which comes with the FreeBSD distribution, and Ericsson's proposal. The packet-loss rate was measured for both encryption schemes.

Measurements were made with different characteristics of the wireless link, ranging from an ideal channel with no bit-errors, to channels resembling real-world characteristics. The performance of both schemes is depicted in Figure 5.2. It shows that IPsec is inferior even with small bit-error rates.

Since Ericsson's proposal was designed to be interoperable with ROHC, we also added ROHC-compression over the wireless link. That is, the mobile terminal transmitted a compressed RTP-stream to the base-station. The stream was then decompressed and forwarded over the fixed network. Due to the fact that IPsec encrypts the IP/UDP/RTP headers, they cannot be compressed, and hence there is not much of a point showing the performance of compressed IPsec packets. Ericsson's proposal in combination with ROHC showed the best results (see Figure 5.2).

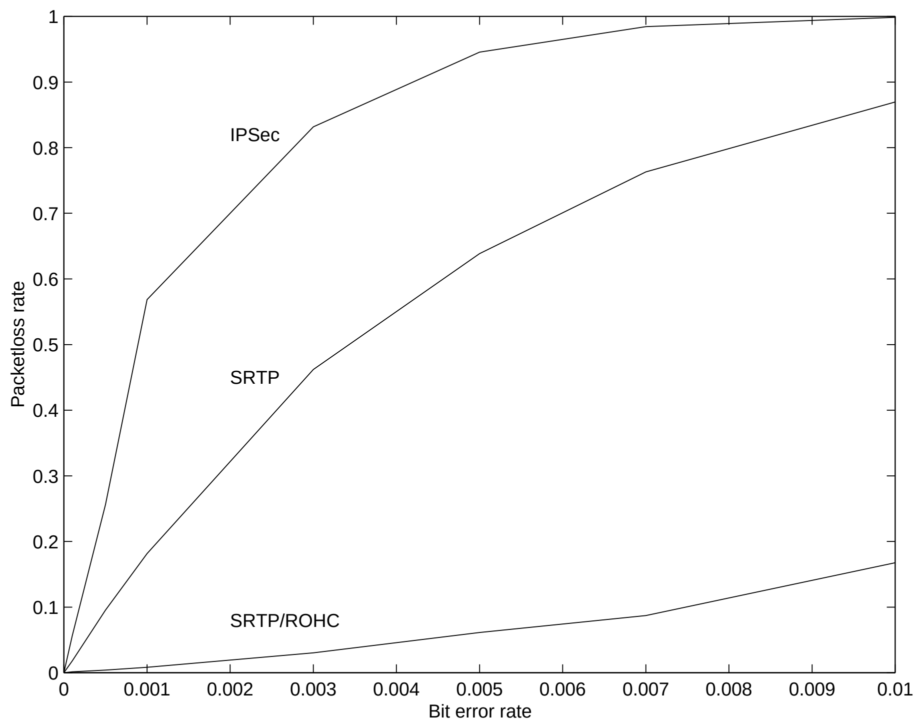


Figure 5.2. These curves show the increase in packet-loss rate corresponding to BER for different solutions.

The measurements show that Ericsson's proposal is well suited for wireless environments, and that the standard solution IPsec does not perform well in this setting.

Chapter 6

Key-management

6.1 Introduction

The security solution presented in this report is based on the fact that the participants in the multimedia session have exchanged keys prior to starting the session. This has to be taken care of by a key-management scheme. The key-management scheme needs to distribute a key to the participants of the session.

This chapter discusses various trust-models and possible solutions for the key-management issue with multimedia sessions in focus. The purpose is to find and discuss alternatives to currently existing key-exchange protocols, emphaseing the sharing of the key and not so much negotiation of particular algorithms used. If such a protocol is found, is it possible to integrate it in SIP/SDP [15, 14]? SIP (Session Initiation Protocol) and SDP (Session Description Protocol) are protocols, commonly used to set up a multimedia session (see Section 6.6.1 below). They are currently undergoing standardization in IETF.

The aim of the protocol suggestions is, unlike the standard key-management protocol IKE [16], not to achieve perfect forward secrecy (PFS, see below), but rather to obtain efficiency. Though it would be possible to achieve PFS even in this case with minor extensions to the protocol at the cost of additional hard computations.

The reason for the study is to obtain a basis for a discussion of the key-exchange needed for multimedia sessions as mentioned in Section 4.4.1.

6.2 Requirements and previous work

Two of the more commonly known protocols for key-exchange are IKE [16] and Kerberos [20]. IKE is mainly used for e2e key-exchange and has the disadvantage that it involves resource demanding computational tasks, e.g., the Diffie-Hellman key-exchange. A Diffie-Hellman key-exchange is about 1000 times as time-consuming as AES. Kerberos is a centralized system, which also comes with disadvantages. Kerberos requires a fully working infrastructure of Kerberos realms, similar to a PKI (see Section 3.11), which will be very complex as the number of realms increases.

The (W)TLS protocol [36] should also be mentioned. It is especially designed with wireless, and to some extent thin, clients in mind.

In most cases all features of IKE might not be necessary. Instead of considering a specific key-exchange, we will try to define specific needs.

- How much "negotiation" is needed?
- What is the minimum number of parameters that can be used?
- What actions can be considered resource demanding (with respect to both computation and communication)?
- What information can be assumed to be known by the negotiators? Will we need a PKI solution or other similar solution to support the system?
- Is it possible to make some general system which works for all different trust-models?
- Is it possible to integrate the key-exchange in SIP/SDP?
- Is perfect forward secrecy needed and what is the computational cost if it is?

6.3 Notation

Note that in this chapter we will, in addition to the demands posed on cryptographically secure hash-functions in Section 3.10, assume that they are pseudo random functions in the following sense. For a fixed hash-function h, the output $h(k||m)$ should be difficult to differentiate from a random string without the knowledge of k. The hash-function can in this sense be thought of as a family of hash-functions that are indexed by k. Furthermore, the length of the hash-value produced must be at least as large as the length of the key. The word key refers to the key we want to exchange. The following notation will be used in this chapter:

m_x	message from x
N_x	nonce, random bit string of fixed size $\geq$ key length, from x
$CERT_x$	certificate for x
E_x	public encryption function for x
E_x^{-1}	inverse encryption function for x (i.e., private decryption function for x)
h_x	public Diffie-Hellman key for x
g	generator for group used in Diffie-Hellman exchange
$HASH$	publicly known and cryptographically secure hash function
ID_x	identity for x
$x\|\|y$	concatenation of two bit strings x and y
$[param]$	optional parameter *param*

6.4 Trust-models

There is a need for a basis, or starting point for the key-exchange. A decision on who can be trusted has to be made. Three models/levels of trust have been identified: end to end trust, centralized trust and mixed trust.

End to end The most restrictive case of trust is the end to end (e2e) trust, where the participants only trust each other. The responsibility of the key-exchange will in this case lie in the hands of the participants. There exist several problems with this approach, both legal and practical. One of the practical problems is the resource demanding protocols needed for the key-exchange, which is a big issue for thin clients. An example of a legal problem is that the authorities might want to be able to intercept and view the communication. This is known as *legal interception*.

Centralized In the centralized scenario, the policy for the home network is that the key negotiation is handled by a centralized authority in the network. An advantage of this is that it can be less resource demanding for the terminals. This, in contrast to the e2e-model, gives the centralized authority the ability to monitor the communications.

Mixed trust With mixed trust we mean networks with conflicting policies, i.e., a terminal in a network with a e2e policy contacting a terminal in a network with centralized policy or vice versa.

6.5 Key-exchange issues

In this section we propose three minimalistic protocols for exchanging keys, discuss the underlying assumptions and implications for these.

6.5.1 Perfect forward secrecy

There are several overlapping definitions (explanations) of Perfect Forward Secrecy (PFS). The one that will be used in the remainder of this report is the following. If a private encryption function of any of the parties or their shared secret somehow is revealed to an adversary, then the adversary shall not be able to derive any information about

- any previous session-keys, or
- any future session-keys.

Also, the exposure of a session-key should not aid an adversary in the quest for subsequent or previous session-keys. This means that if a session-key is compromised, only information contained in that session will leak.

6.5.2 Sufficient information for an exchange

The aim of this section is to investigate the minimal amount of information needed in a key-exchange (where only one common key is to be exchanged), depending on the shared information. We do not consider negotiations of common crypto algorithms and other information that some of todays key-exchange protocols include. Neither do we consider PFS at this stage. The messages in our protocol turned out to show large similarities to SKEME [21].

Known public key encryption function Assuming that the caller knows the public encryption function of the callee, obtained from either the callee in person, trusted common acquaintance or some CA, the protocol in Figure 6.1 is believed to be the minimal protocol for this kind of key-exchange.

The initiating message m_A consists of nonce N_A used to derive the key, the identity ID_A of the caller to make it possible for the callee to look up the public encryption function for the caller. A certificate $CERT_A$ may be included if the caller doesn't expect the callee to know the public encryption function E_A. The callee can in this case verify the identity and the public encryption function of the caller. Authentication is in this case based on the common belief that if the receiver of a message can decrypt it correctly, the receiver is in possession of the private key (not necessarily proving that the receiver really is who he claims to be). The additional hash-value $HASH(N_A||N_B)$ is included in the response message m_B to ensure that no errors have been introduced during transmission and to prove that the callee actually was able to decrypt m_A.

The session-key can be defined as $HASH(N_A||N_B)$. The reason for including nonce from both parties is that it reduces the effects if one of them is using a defect generator for the nonce.

Alice (caller)		Bob (callee)
Public enc. function E_A Knows E_B		Public enc. function E_B
$m_A = \{E_B(N_A), ID_A[, CERT_A]\}$	$\longrightarrow$	
	$\longleftarrow$	$m_B = \{E_A(N_B\|\|HASH(N_A\|\|N_B))\}$

Figure 6.1. Minimal key-exchange protocol for known public key. The session-key is defined as $HASH(N_A||N_B)$.

Known public Diffie-Hellman key Assuming that the caller knows the public Diffie-Hellman (DH) key of the callee, obtained from either the callee in person, trusted common acquaintance or some CA, the protocol in Figure 6.2 is believed to be the minimal protocol for a key-exchange.

The initiating message m_A consists of nonce N_A used to derive the key, the identity ID_A of the caller to make it possible for the callee to look up the public Diffie-Hellman key for the caller. A certificate $CERT_A$ may be included if the caller doesn't expect the callee to know her public Diffie-Hellman key $h_A = g^a$. The callee can in this case verify the identity and the public Diffie-Hellman key of the caller. Authentication is based on the assumption that only the caller and callee can calculate g^{ab}. The additional hash-value $HASH(N_A||N_B)$ is included in the response message m_B to assure that no errors have been introduced during transmission.

The session-key can be defined as $HASH(g^{ab}||N_A||N_B)$. For future use, g^{ab} can be stored and used as a pre-shared secret.

Alice (caller)		Bob (callee)
Private DH-key a		Private DH-key b
Public DH-key $h_A = g^a$		Public DH-key $h_B = g^b$
Knows h_B		
$m_A = \{N_A, ID_A[, CERT_A]\}$	$\longrightarrow$	
	$\longleftarrow$	$m_B = \{N_B, HASH(N_A \| \| N_B)\}$

Figure 6.2. Minimal key-exchange protocol for known public DH-key. The session-key is defined as $HASH(g^{ab}||N_A||N_B)$.

Pre-shared secret Assuming that caller and callee share a common secret S, the protocol in Figure 6.3 is believed to be the minimal protocol for a key-exchange.

The initiating message m_A consists of nonce N_A used to derive the key, and the identity ID_A of the caller to make it possible for the callee to look up the pre-shared key for the caller. The callee returns the message m_B consisting of nonce N_B and a checksum $(HASH(N_A||N_B))$.

The session-key can be defined as $HASH(S||N_A||N_B)$.

Alice (caller)		Bob (callee)
Pre-shared secret S		Pre-shared secret S
$m_A = \{N_A, ID_A\}$	$\longrightarrow$	
	$\longleftarrow$	$m_B = \{N_B, HASH(N_A \| \| N_B)\}$

Figure 6.3. Minimal key-exchange protocol for pre-shared secret. The session-key is defined as $HASH(S||N_A||N_B)$.

No information about the callee If no public key or pre-shared secret exists, the caller must in some way receive this from a trusted source. Therefore we assume that a public key (either a DH-key or encryption function) can be obtained from this trusted source and one of the previous cases can then be applied.

Perfect forward secrecy The protocols above fulfill the second requirement of PFS, but fails the first. To also fulfill the first requirement, the nonces should be substituted with DH-keys g^r and g^s, where r and s are numbers that are randomly generated for each session. The session key is then defined as $HASH(X||g^{rs})$, where X is the empty string for the known public key case, S for the shared secret case and g^{ab} for the DH-case. This requires that there exist certain standard groups for which the DH-exchange can be executed. The disadvantage of this approach is the cost of the extra DH-exponentiations.

As outlined above there is a computational cost involved in adding PFS; it might be questionable whether it is worth it or not. This is of course dependent on the application.

Alice (caller)		Bob (callee)
INVITE	$\longrightarrow$	
	$\longleftarrow$	OK
ACK	$\longrightarrow$	

Figure 6.4. Simplified structure of a SIP transmission.

In the scenario using public-key encryption without DH, an attacker must have knowledge of the private keys of both communicating parties to be able to eavesdrop on their sessions and reconstruct the key to the previous sessions. This might be sufficient security for multimedia-sessions where in most cases the data is not in need of very long-term protection.

6.6 Proposed solution

We will in this section discuss possible solutions to the problems posed above within the three identified trust-models.

For the e2e scenario we require a key-exchange protocol that places as little burden as possible on the terminals, since these to a large extent, might be computationally weak. IKE gives a PFS solution to the key-exchange problem but does not consider a solution for thin clients. (W)TLS is in general less computationally intensive than IKE, but does not provide PFS and also has the same disadvantage as IKE does, that the messages can become quite large. From an e2e perspective both IKE and (W)TLS are possible, where (W)TLS might be more efficient. Neither (W)TLS nor IKE can in their original forms be integrated in SIP/SDP (see below). Therefore it might be better to define a new key-exchange protocol suitable for integration in other protocols (such as SIP/SDP).

We will in the next section describe a solution that integrates a key-exchange protocol in SDP and how this will be affected by the different trust-models previously mentioned.

6.6.1 Integration in SIP/SDP

SIP First we will briefly describe what SIP/SDP [15, 14] is. SIP (Session Initiation Protocol) is a standard protocol for setting up sessions. In a simplified manner, depicted in Figure 6.4, SIP consists of three messages. The caller sends an `INVITE` message to the callee, who responds with an `OK`. Finally, the caller sends an `ACK` message (acknowledge) to the callee.

The first two messages are passed through SIP-servers, who have knowledge of the location of the terminals in their respective domains. This approach allows terminals to be mobile as long as they register to their SIP-server whenever they change location. The acknowledge message is not routed by the SIP-servers, but directly between the caller and the callee.

SDP SIP is text based, and allows another protocol to piggy-back on the messages sent. A protocol designed for describing sessions, SDP (Session Description Protocol), is such a piggy-back protocol. SDP, which is also text based, has a number of lines of the type "`x=...`", where x is what is to be described followed by the description. Examples of what can be described are media type, from whom the message was sent, and via which hosts the message was routed.

There are some particular lines that are of interest for our purposes, namely the attribute lines. These lines can be specified for the session as a whole, and for each media type. Note that there can be more than one attribute line whenever there can be one, so using an attribute does not reduce the possibilities for further expansions. The attribute lines have no restrictions on their content and begins with `a=`, for attribute.

Integration of the key-exchange The key-exchange protocols used in this proposal will be those mentioned in Section 6.5.2. We will further assume that there is only need for one session key per multimedia session. That session key can then be used to derive separate keys for each media stream in a way similar to the proposal in [5]. However, the scheme can easily be extended to exchange a key for each media stream.

The protocol messages are carried as attributes in the SDP session description. We illustrate the idea with an example. The important thing to note is the use of attribute lines.

```
C->S: INVITE sip:schooler@cs.caltech.edu SIP/2.0
         Via: SIP/2.0/UDP csvax.cs.caltech.edu;branch=8348
           ;maddr=239.128.16.254;ttl=16
         Via: SIP/2.0/UDP north.east.isi.edu
         From: Mark Handley <sip:mjh@isi.edu>
         To: Eve Schooler <sip:schooler@caltech.edu>
         Call-ID: 2963313058@north.east.isi.edu
         CSeq: 1 INVITE
         Subject: SIP will be discussed, too
         Content-Type: application/sdp
         Content-Length: 187
         ...
         e=mbone@somewhere.com
         c=IN IP4 224.2.0.1/127
         a=[key-exchange parameters]
         t=0 0
         m=audio 3456 RTP/AVP 0

S->C: SIP/2.0 200 OK
         Via: SIP/2.0/UDP csvax.cs.caltech.edu;branch=8348
           ;maddr=239.128.16.254;ttl=16
         Via: SIP/2.0/UDP north.east.isi.edu
         From: Eve Schooler <sip:schooler@caltech.edu>
         To: Mark Handley <sip:mjh@isi.edu>
         Call-ID: 2963313058@north.east.isi.edu
         CSeq: 1 INVITE
         Contact: sip:es@jove.cs.caltech.edu
         Content-Type: application/sdp
         Content-Length: 102
         ...
         a=[key-exchange parameters]
         m=audio 6543 RTP/AVP 0

C->S: ACK sip:es@jove.cs.caltech.edu SIP/2.0
         Via: SIP/2.0/UDP north.east.isi.edu
         From: Mark Handley <sip:mjh@isi.edu>
         To: Eve Schooler <sip:schooler@caltech.edu>
         Call-ID: 2963313058@north.east.isi.edu
         CSeq: 1 ACK
```

If a separate key is required for each media stream, a unique nonce is added as an attribute for the media descriptor. Note that the key-exchange parameters in this case still are present in the session description.

```
C->S: INVITE sip:schooler@cs.caltech.edu SIP/2.0
         Via: SIP/2.0/UDP csvax.cs.caltech.edu;branch=8348
           ;maddr=239.128.16.254;ttl=16
         ...
         Content-Length: 187
         ...
         e=mbone@somewhere.com
         c=IN IP4 224.2.0.1/127
         a=[key-exchange parameters]
         t=0 0
         m=audio 3456 RTP/AVP 0
         a=[NONCE1]
         m=video 7654 RTP/AVP 31
         a=[NONCE2]
         m=application 6543 UDP 0
         a=[NONCE3]
         ...
```

Assuming a 1024-bit public RSA-key, the size of a typical SIP/SDP-message will increase by a factor 4 if a certificate is included and the size of the certificate is approximately 700 bytes (X.509 certificate). If the certificate is not included the size of the SIP-message is increased by approximately 200 bytes. If instead ElGamal over elliptic curves is used the increase in size of the message will be even less.

In the case of a shared secret the protocol adds a negligible amount of extra data to the SDP-message.

Below follows discussions of the protocols suitability in the earlier identified trust-models and comparisons to other protocols.

6.6.2 End to end trust

In the e2e trust-model there are two distinct ways to negotiate the key; either with the aid of SIP, or to set up a separate key negotiation between the two end points. In the first case the proposed integration above can be applied, and in the second case protocols like IKE and (W)TLS can be used. If a separate key negotiation is used SIP has to be extended anyhow to provide for key negotiation to take place. For computationally weak terminals (W)TLS is probably a better alternative than IKE if one can disregard the PFS-property.

6.6.3 Centralized trust

One of the first solutions that come to mind in the centralized trust-model is to use a system like Kerberos. This could be a quite good alternative but it requires that the different networks the end users belong to trust each other (i.e., they already have some sort of policy agreements) and both uses Kerberos.

Another solution that might be better is to let the SIP servers negotiate the key (transparently to the user) and provide the user with the key in the key-field of the SIP-message when done; the SIP-server has a relation to the terminal, so they can be assumed to have a shared secret. This solution relys on the security of SIP, which is out of scope for this thesis. There are integrity problems involved in this approach. Is the SIP-server allowed to modify the SIP-messages between users? But on the other hand, the whole idea of a centralized solution introduces integrity problems.

6.6.4 Mixed trust

This trust-model introduces a lot of problems for systems using different policies that are not compatible, e.g., Kerberos and IKE. To circumvent this problem the solution using integrated key negotiation in SIP can be an alternative. Using this approach different systems need not know the internal structure of its negotiation partner. In centralized systems the negotiator for the user can be viewed as a distributed terminal which will become transparent to a user in another network. A distributed terminal is a set of network entities, e.g., a cellular phone and a SIP-server.

6.7 Conclusions

We can conclude that it is possible to integrate some sort of key negotiation protocol in SIP/SDP, though it is not possible to integrate IKE and (W)TLS due to the required amount of messages. If IKE or (W)TLS are to be used, then a separate communication must be set up.

The solution above using SIP/SDP suggests that it is possible to have a system general to the three trust-models. It is also possible to extend this to include negotiation of crypto algorithms, hash-functions etc. for the multimedia session. This is easily accomplished by adding new attributes in SIP/SDP.

Since it cannot be assumed that all participants share a pre-shared secret, some form of PKI-solution is needed for non-centralized trust-models. Trusting a CA or other authority is of course against the philosophy of the e2e-trust. But that is the only solution, unless everyone physically can exchange a pre-shared secret or public key. This is however not practical for communities of reasonable size; it takes approximately 28000 years to physically exchange keys with the entire world population without parallelization. The centralized trust-model reduces this problem to the authorities of the realms.

If the terminals are computationally strong, the choice of protocol is more or less a policy question. On the other hand terminals that are weak will have trouble with executing protocols like IKE. An example comparison of IKE and (W)TLS when public keys are known to the participants shows that the number of computationally demanding operations differs severely. In this case IKE requires 6 hard computations and (W)TLS only 2 (this is when the certificate is not validated in any of the protocols).

In this chapter we have only been concerned with the actual exchange of the key and more or less assumed that the set of crypto algorithms used is standardized and fixed.

An open question is whether PFS is needed in this particular case or not.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

To summarize the report we briefly describe what was found and achieved with each objective, stated in Chapter 1, in mind.

Examine requirements for encrypted data that is transmitted in a wireless and packet-switched environment. Encryption of data in a wireless, packet-switched environment raises special constraints on the encryption mechanism. The main issues are that the encryption must add minimal redundancy and should not propagate bit-errors.

Study the proposal for UMTS security (3GPP's f8-mode of operation) and investigate if it fits multimedia sessions based on RTP. 3GPP's algorithms and f8-mode suit the purposes of confidentiality for RTP based multimedia sessions, if slightly modified.

Study the Kasumi and AES/Rijndael block ciphers which may be used in f8-mode. Neither Kasumi nor Rijndael have yet shown any weaknesses that might be exploited by an adversary. Both algorithms promise sufficient speed and size, even for thin clients. However, the 64-bit block-size of Kasumi showed to be too small to accommodate for the IV. As it turns out, the Kasumi implementation will be heavily used by the terminal, and will have no time to deal with encryption of multimedia streams.

Implement and evaluate encryption of RTP-streams and integrate the implementation in Ericsson's testbed for VoIPoW simulation. The implementation of the block ciphers and f8-mode run at speeds which stand quite well in comparison with other existing measurements. Furthermore, the integration in the testbed shows that the implementation meets the demands of VoIPoW.

Investigate the implications of using authentication and integrity control of the media streams. Standard authentication and integrity checks affects the audio and video quality in such a negative way that it should be avoided in wireless environments.

Find a proposal for the authentication/integrity problem, implement it and do simulations. As stated above authentication and integrity of the media streams should be avoided, due to quality reasons. It will, as we view it, not be needed in most applications anyway. If authentication and integrity is important for an application, it can be done by the use of a keyed hash.

Discuss and search for problems and scenarios for the key-exchange problem for multimedia sessions on thin clients. Propose a basis for a discussion of a solution to the problem. We identified three models of trust, end to end, centralized and a combination of the two. For each of the models, the supposed minimum of information needed to exchange a key was found. We argued that a security solution involving thin clients, suffers from efficiency problems if perfect forward secrecy is required. Hence, IKE is not very suitable.

7.2 Future work

This section will discuss what work is being done at current time, and what could be done in the future.

7.2.1 Key-management

The main issue is, at the time of writing, to solve the key-management problem for multimedia sessions. Work is in progress at Ericsson to solve that. Two drafts have been submitted to IETF, one containing the requirements [4], and one with a solution [7]. These drafts needs some further work.

7.2.2 Implementation

The implementation of Rijndael has, as discussed in Section 3.5, some efficiency problems in one module. The module should be optimized in the ways pointed out in the same section.

The SRTP implementation should be kept up to date with the changes in the specifications.

When a satisfying solution for the key-distribution problem is derived, it should be implemented in the test-bed.

7.2.3 SRTP

The draft should be finalized, so that it can go to last call in November 2001.

Bibliography

[1] Bellare, M., Desai, A., Jokipii, E., Rogaway, P., *"A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation"*, Proceedings of the 38th Symposium on Fundations of Computer Science, IEEE, 1997, p.394–403.

[2] Biham, E., Shamir, A, *"Differential cryptanalysis of DES-like cryptosystems"*, Journal of cryptology, vol. 4, no. 1, 1991, p.3–72.

[3] Blom, R., Carrara, E., Näslund, M., *"Conversational multimedia security in 3G networks"*, IETF-draft (work in progress), November 2000.

[4] Blom, R., Carrara, E., Lindholm, F., Arkko, J., *"Design Criteria for Multimedia Session Key Management in Heterogeneous Networks"*, IETF-draft (work in progress), July 2001.

[5] Blom, R., Carrara, E., Näslund, M., Norrman, K., *"RTP encryption for 3G networks"*, IETF-draft (work in progress), November 2000.

[6] Burmeister, C., Clanton, C., Degermark, M., Fukushima, H., Hannu, H., Jonsson, L.E., Hakenberg, R., Koren, T., Le, K., Liu, Z., Martensson, A., Miyazaki, A., Svanbro, K., Wiebke, T., Yoshimura, T., Zheng, H., *"RObust Header Compression (ROHC)"*, RFC 3095, July 2001.

[7] Carrara, E., Lindholm, F., Näslund, M., Norrman, K., Arkko, J., *"Key Management for Multimedia Sessions"*, IETF-draft (work in progress), July 2001.

[8] Daemen, J., Rijmen, V., *"AES proposal: Rijndael"*, available at `www.esat.kuleuven.ac.be/~rijmen/rijndael/` (visited October 31, 2000).

[9] Diffie, W., Hellman, M., *"New directions in cryptography"*, IEEE Transactions on Information Theory, Vol. 22, Number 6, November 1976, p.644–654.

[10] Electronic Frontier Foundation, *"Cracking DES, Secrets of Encryption Research, Wiretap Politics & Chip Design"*, 1st Edition July 1998, ISBN 1-56592-520-3.

[11] ETSI/SAGE, *"Specification of the 3GPP Confidentiality and Integrity Algorithms"*, available at `www.etsi.org/dvbandca/` (visited January 5, 2001).

[12] Federal Information Processing Standards Publication 180-1, *"Secure hash standard"*, available at `www.itl.nist.gov/fipspubs/fip180-1.htm#FORE_SEC` (visited January 23, 2001).

[13] Guardian Digital Inc., *"Security Dictionary"* `www.linuxsecurity.com/dictionary/` (visited Januray 9, 2001).

[14] Handley, M., Jacobson, V., *"SDP: Session Description Protocol"*, RFC 2327, April 1998.

[15] Handley, M., Schulzrinne, H., Schooler, E., Rosenberg, J., *"SIP: Session Initiation Protocol"*, RFC 2543, March 1999.

[16] Harkins, D., Carrel, D., *"The Internet Key Exchange (IKE)"*, RFC 2409, November 1998.

[17] Kent, S., Atkinson, R., *"IP Authentication Header"*, RFC 2402, November 1998.

[18] Kent, S., Atkinson, R., *"IP Encapsulating Security Payload (ESP)"*, RFC 2406, November 1998.

[19] Koblitz, N., *"Algebraic aspects of cryptography"*, Algorithms and computation in mathematics, Vol. 3, Springer Verlag, 1998, ISBN 3-540-63446-0.

[20] Kohl, J., Neuman, C., *"The Kerberos Network Authentication Service (V5)"*, RFC 1510, September 1993.

[21] Krawczyk, H., *"SKEME, A Versatile Secure Key Exchange Mechanism for Internet"*, IEEE Symposium on NDSS, 1996, p.114–127.

[22] Lenstra, A., Verheul, E. *"The XTR public key system"*, Advances in cryptology 2000, Lecture notes in computer science, vol. 1880, Springer-Verlag, 2001, p.1–19.

[23] Matsui, M., *"Linear Cryptanalysis Method for DES Cipher"*, Advances in Cryptology—Eurocrypt '93, Lecture notes in computer science, vol. 765, Springer-Verlag, 1994, p.386–397.

[24] Matsui, M., *"New Block Encryption Algorithm MISTY"*, Proceedings of Fast Software Encryption, Lecture notes in computer science, vol. 1267, Springer-Verlag, 1997, p.54–68.

[25] McGrew, D., Oran, D., *"The Secure Real Time Protocol"*, IETF-draft (work in progress), November 2000.

[26] Menezes, A., Van Oorschot, P., Vanstone, S., *"Handbook of applied cryptography"*, CRC Press, 1997, ISBN 0-8493-8523-7.

[27] Papadimitriou, C., *"Computational Complexity"*, Addison-Wesley Publishing Company, Inc., 1994, ISBN 0-201-53082-1.

[28] Perrig, A., Canetti, R., Briscoe, B., Tygar, D., Song, D., *"TESLA: Multicast Source Authentication Transform"*, IRTF-draft (work in progress), June 2001.

[29] Postel, J., *"User Datagram Protocol"*, RFC 768, August 1980.

[30] Postel, J., *"Internet protocol"*, RFC 791, September 1981.

[31] Rivest, R., *"The MD5 Message-Digest Algorithm"*, RFC 1321, April 1992.

[32] Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V., *"RTP: A Transport Protocol for Real-Time Applications"*, RFC 1889, January 1996.

[33] Tannenbaum, A., *"Computer networks"*, Prentice-Hall, 1996, ISBN 0-13-349945-6.

[34] Westberg, L., Lindqvist, M., *"Realtime Traffic over Cellular Access Networks"*, IETF-draft (work in progress), November 2000.

[35] Ziv, J., Lempel, A., *"A Universal Algorithm for Sequential Data Compression"*, IEEE Transactions on Information Theory, Vol. 23, Number 3, May 1977, p.337–343.

[36] *"WAP WTLS Version 05-Nov-1999"*, `www1.wapforum.org/tech/documents/SPEC-WTLS-19991105.pdf` (visited January 25, 2001).