Key Agreement for Secure Voice over IP

JOHAN BILIEN

KTH Microelectronics
and Information Technology

Master of Science Thesis
Stockholm, Sweden 2003

IMIT/LCN 2003-14

Key Agreement for Secure Voice over IP

Master of Science Thesis
December 2003

Johan Bilien <Johan.Bilien@via.ecp.fr>
Center for Wireless Systems and Telecommunication Systems Laboratory
Kungl Tekniska Högskolan
Stockholm

Supervisor and examiner: Professor Gerald Q. Maguire Jr. <maguire@it.kth.se>
Advisors: Erik Eliasson <eliasson@it.kth.se>
Jon-Olov Vatn <vatn@it.kth.se>

Preface

This work was conducted as a degree project at the Telecommunication Systems Laboratory (TS-lab), Department of Microelectronics and Information Technology (IMIT), Royal Institute of Technology (KTH), Stockholm, between June and December 2003.

I would like to express my sincere gratitude to:

- Professor Gerald Q. Maguire Jr., for the opportunity he gave me to conduct this project, for his support and for his inestimable suggestions and comments.
- Jon-Olov Vatn, for his constant help and support, and for his precious advices and experience,
- Erik Eliasson, for his technical help and programming advices, and for his patience debugging my mistakes,
- the `#handhelds.org` crew, especially Jamey Hicks, pb and ryan, for their help and advices regarding Linux on iPAQ,
- Magnus Brolin, Elisabetta Carrara, Fredrick Lindholm and Karl Norrman from ERICSSON, for their attention, and comments on our paper.

Abstract

This thesis reviews the usual properties and requirements for key agreement protocols. It then focuses on MIKEY, a work-in-progress protocol designed to conduct key agreements for secure multimedia exchanges. The protocol was implemented and incorporated in a SIP user agent - *minisip*. This implementation was used to measure the additional delay required for key exchange during call establishment. Finally, some schemes are proposed regarding the use of MIKEY in advanced VoIP scenarios, such as conferences and terminal mobility.

Contents

Chapter 1

Introduction

Voice over IP (VoIP) has recently raised much interest as the number of providers offering VoIP services has increased. Numerous areas have raised concerns. Technical questions, for instance interoperability and quality of service, present interesting challenges. Legislation of VoIP is currently being discussed, a subject of intense debate is whether or not a VoIP provider must follow the same legal constraints as current Public Switched Telephone Network (PSTN) operators. Economical aspects give other interesting subjects of study: will the *traditional* operators have to adapt to these new competitors? What prevents a private person from connecting his PSTN access to his Internet connection, thus being able to offer de-regulated telephony services?

As VoIP becomes more common, security and privacy issues must be considered. PSTN eavesdropping has always been the subject of controversy, since it requires technical means only available to *official* organizations. As VoIP goes over packet based networks, end users have no real control on who may be able to listen to their communications. Wireless networks, for instance, offer very easy and straight forward possibilities of eavesdropping. We believe that VoIP offers enough flexibility to set up end-to-end security associations, thus providing private communication channels over public networks, and allowing a *higher* level of confidentiality than today's PSTN telephony.

After presenting a quick overview of the state of the art in VoIP, we will try to define what *secure* VoIP involves, and to describe a general model for it. We then focus on key agreement protocols: we will try to list their requirements and properties, study some examples, and see why a new protocol is being developed for the specific purpose of multimedia exchanges (MIKEY [4]). An implementation of this protocol allows us to measure the additional delay required in a *secure* call setup. Finally, we will see how

a key agreement scheme can be used in advanced VoIP scenarios, such as conferences and device mobility.

Part I

Introduction to Voice over IP and Voice over IP security

Chapter 2

Voice over IP overview

2.1 Definition

According to the International Engineering Consortium, Voice over IP can be defined as follows [23]:

> A voice-over-Internet protocol (VoIP) application meets the challenges of combining legacy voice networks and packet networks by allowing both voice and signaling information to be transported over the packet network.

2.2 SIP: A signaling protocol for VoIP

Every person wishing to be reachable through the VoIP system should be assigned a unique ID. Unlike traditional phone numbers which designate a particular phone, these IDs will identify a person (or device or service). At first one might consider using the host's IP addresses for that purpose. But this would allow and bind only one user per network interface. Moreover, when using any type of address translation, such as a NAT gateway or mobile IP, the IP address is not unique. More and more, an IP address is either assigned temporarily, or shared between several hosts. Therefore, a new kind of ID must be assigned to each user, and a protocol has to support the dynamic translation from this new ID to the network address to which the ID's owner is currently reachable. This new ID should, as much as possible, be easy to obtain and remember.

Another requirement for VoIP is to replace the original circuit switched network's signaling messages by an equivalent IP system. For instance, when a user Alice wants to establish a call to another user Bob, a signal must

inform the latter that there is an incoming call request, then a response message must be transmitted, stating whether or not the call was accepted.

The Session Initiation Protocol [37] (SIP) was designed to handle these two tasks. Each user is assigned a SIP Uniform Resource Identifier (URI). Its form is very similar to an e-mail address: an '`@`' symbol separates a user part from a domain part, for instance `sip:alice@somedomain.org`.

In order for the user's SIP client to contact a peer, a mechanism must be implemented to provide a translation from this SIP URI to the actual location (network address) of the user. Therefore, each client has to register with a SIP registrar, which will keep in a database the current IP address of the registered clients. The registrar may also be responsible for authentication and accounting for these users, in which case the client is presented with a HTTP digest challenge.

When a client wants to contact another client given a specific SIP URI, it has to find the registrar to which the user is registered. The look up for a registrar is very similar to locating a SMTP server for a given e-mail address. Several methods could be tried to contact this registrar:

- the registrar for this SIP URI is already known by the client ;
- the registrar's network address is designated by a DNS SRV [13] record for the domain contained in the SIP URI;
- the SIP URI domain part has an DNS A record, which points to the registrar ; or
- the SIP URI domain, prepended with 'sip.' has a DNS A record, which points to the registrar.

In many situations, the client may rely on a SIP proxy server to look up this registrar and to forward the SIP messages. This can be useful, for example, if the users are behind a NAT gateway and are not assigned publicly reachable network addresses. The SIP proxy may also speed up the lookup by keeping a cache of the recently looked up registrars. Very often, the SIP proxy and the SIP registrar are actually running on the same host.

SIP defines a set of signaling messages:

- `INVITE` is used by a user to request a call to another user
- `ACK` is used by the receiver to accept the call
- `BYE` is used to end a call
- `OPTIONS` is used to request a client's capabilities

- `CANCEL` is used to cancel a request
- `REGISTER` is used by a client to register with its registrar

When both users are using their registrar as a SIP proxy, the establishment of a call will typically be conducted as shown in figure 2.1.

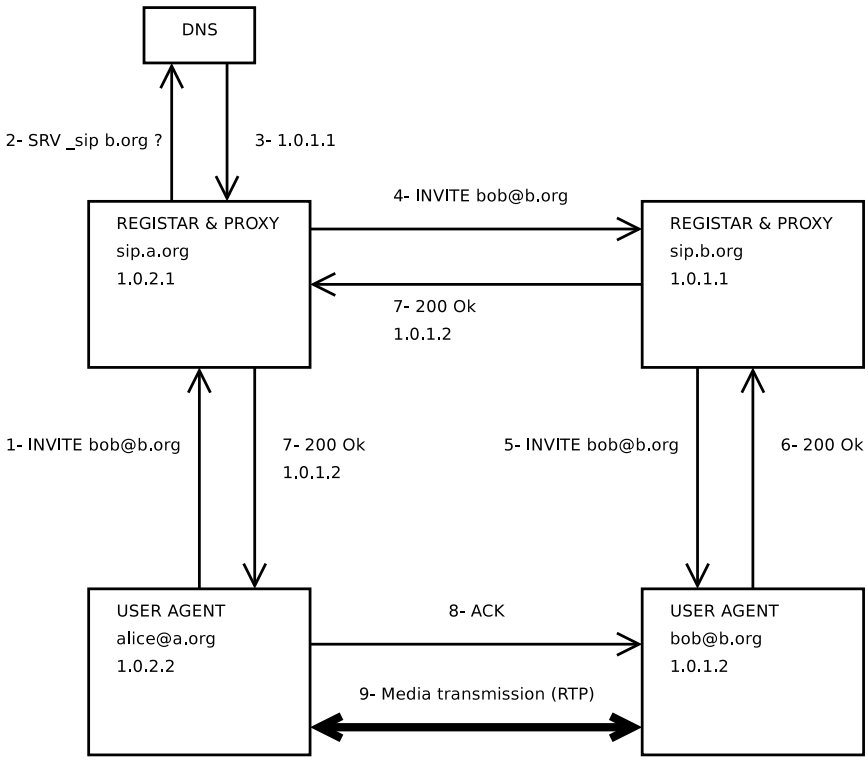

Figure 2.1: Simplified VoIP call establishment using SIP

Many implementations of SIP are available. Microsoft has adopted SIP in its Windows Messenger [29], delivered with Windows XP. Kphone [30] is an open source implementation, released under the GPL [10], and part of the KDE [41] project. In my experiments, I used Erik Eliasson's *minisip* client.

2.3 SDP: Session description and CODECs negotiation

VoIP is very flexible about the type of services it provides. A call is basically an exchange of multimedia content, which may consist of a voice channel, a

video channel, a shared virtual white board, ...

Each multimedia channel will be encoded in one or more defined formats that both peers must handle. For sound and video streams, the data will often be compressed to reduce bandwidth consumption. There are nowadays a significant number of encoders/decoders (CODECs) available, and they are usually not compatible with each other. Therefore both peers have to share a CODEC pair and agree on which one to use before the call is established.

To reduce the number of roundtrips and thus the call establishment delay, SIP allows the initiator to describe which type of call she would like to establish, and what CODECs and formats she proposes to use, directly in the `INVITE` message. The receiver then describes his own capabilities in the `200 OK` message. Thus after one roundtrip, common pairs are chosen.

For this purpose, SIP uses the Session Description Protocol [17] (SDP). SDP is a textual protocol designed to describe multimedia sessions. SDP provides information about the type of media (audio, video, ...), the CODEC (MPEG, PCM μ-law, ...), the transport protocol for media stream (RTP, ...), and port numbers. It consists of several fields, among which are:

- Version field (v:) carries the SDP protocol version
- Origin field (o:) carries information on the user proposing the session (name, network address, ...)
- Session Name field (s:) gives a title for the session
- E-mail address field (e:) may provide the e-mail address of the caller
- Phone Number field (p:) may provide the phone number of the caller
- Time field (t:) describes when the session description is valid
- Connection Data field (c:) gives information about the network connection to establish for the session
- Media Announcements field (m:) describes the type of media used (audio/video, CODEC, ...)
- Attribute field (a:) gives additional properties for the session or one of the media streams, such as the aspect ratio.

A typical SDP description for a call established with SIP may look as follows:

```
v=0
o=3344 3344 IN IP4 130.237.251.200
s=Minisip Session
c=IN IP4 130.237.251.200
t=0 0
m=audio 1061 RTP/AVP 0
a=rtpmap:0 PCMU/8000/1
m=video 1062 RTP/AVP 31
a=rtpmap:31 H261/90000
```

Each media stream is given an Media Announcement field which assigns it one or more CODECs and a network port. When describing his own capabilities, the responder puts a 0 in the network port of the streams he does not want to establish. In our case, the responder does not want to establish a video stream, so his description could be:

```
v=0
o=3344 3344 IN IP4 130.237.251.200
s=Minisip Session
c=IN IP4 130.237.251.200
t=0 0
m=audio 1061 RTP/AVP 0
a=rtpmap:0 PCMU/8000/1
m=video 0 RTP/AVP 31
```

2.4 RTP: Media transport

Once the call is established, for example with SIP and SDP, both users know the other's network address and a set of mutually agreed multimedia sessions which each includes a media type, a CODEC, and a network port.

To actually conduct the call, the users simply transmit to each other the multimedia data, in most cases in a bidirectional channel. The Real-time Transport Protocol (RTP) [11] is designed to carry this data over an IP network, primarily over the UDP transport layer.

An RTP packet is divided in a header and a payload. The payload contains the multimedia stream. The header carries additional information, including:

- A timestamp is added for synchronization purposes
- A sequence number allows the receiver to know the order in which the multimedia payloads should be processed

- A flag tells if padding was added to the payload
- The Synchronization Source Identifier describes who created the packet
- A list of Contributing Sources Identifiers describes who has modified the content since its creation (used mostly in case of a multicast session)

RTP also provides a control protocol, the Real-time Transport Control Protocol [11] (RTCP). RTCP uses an additional channel for monitoring of the stream. Each participant provides information about the received stream. Knowing the quality of the end-to-end path may be useful for instance with adaptive CODECs which adapt their bitrate and thus the quality, according to the available path capacity.

RTCP defines a set of packet types, including:

- Sender Report (SR) carries transmission and reception information from the active senders
- Receiver Report (RR) gives statistics about reception from the participants who are not sending
- Source Description Items (SDES) provides an additional identifier of the different sources of the stream
- BYE is used to signify the end of a user's participation

Chapter 3

Securing Voice over IP

3.1 Several levels of security

Defining what one actually means by a *secure* VoIP system is important. In general, the following aspects could be considered as high priorities:

User authentication during registration: During the registration process, the SIP registrar should authenticate the client. For instance, a user shall not be able to identify itself as another user, who could then receive the latter's calls. User authentication is also sometimes required for accounting purposes.

Media stream encryption: The main problem regarding security when replacing a circuit switched network with a packet based one is that you no longer know or control the path your data is using. The packets may pass through a number of gateways, over which you have no control at all. In the case of a wireless or a hubbed network, everyone connected to the same access point or hub will receive a copy of your information. Therefore, in order to provide communication confidentiality, an end-to-end encryption scheme must be used between the two participants.

Mutual authentication of users: Even if the user may recognize each other's voice, a strong mutual authentication is still required to avoid man-in-the-middle attacks. Moreover, a user may set up a policy to reject any calls initiated from an unknown source. In this case, the filtering of unsolicited calls requires strong authentication of the caller.

Other aspects can be considered, although they generally have a lower priority, these include

Secured SIP Signaling: SIP users' privacy expectations may include the confidentiality of the outgoing and incoming SIP traffic: the users may want to keep private the list of calls they made, and the identities of their peers. In this case, the SIP traffic should be secured as well, which involves SIP message encryption and integrity control, and strong authentication of the SIP proxies involved.

Registrar authentication by the user: During the registration process, strong authentication of the registrar by the user prevents fake SIP registrars, who could try to gain the user's authentication information through, for instance, repeated challenges.

In this study, we will focus only on the aspects listed as having a higher priority. Although the paper mentioned in chapter 13 describes some of the additional privacy and security aspects.

3.2 Securing the media stream

3.2.1 Requirements

The encryption of the stream should be provided by a strong cryptographic algorithm. Since the communication takes place via a full duplex channel for each stream, a symmetric cryptographic scheme is preferred (for performance reasons), so that one key per channel is sufficient. Since VoIP is being increasingly used on embedded (and even mobile) systems, the algorithms used should require low computational costs or be implemented in hardware.

3.2.2 SRTP: a secure profile for RTP

One first alternative for securing the media stream is to add encryption at the application layer.

The Secure Real-time Transport Protocol [5] (SRTP) is a secure profile for RTP. It is currently specified in a draft from the IETF. It adds encryption and integrity control to every RTP packet, as shown in figure 3.1.

An optional integrity control can also be added to the data, using a Message Authentication Code (MAC), as well as a Master Key Identifier (MKI) which tells the receiver which cryptographic key to use.

A cryptographic context is responsible for keeping the state of the ciphered stream. The overall packet protection process is described in figures 3.2 and 3.3.

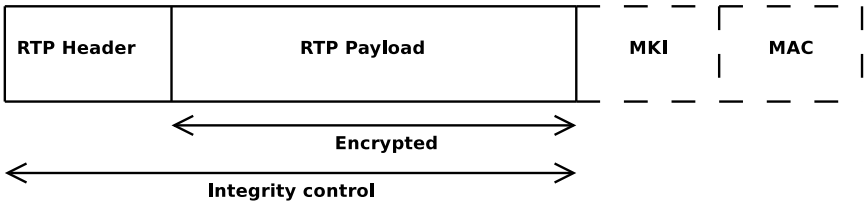

Figure 3.1: An SRTP Packet

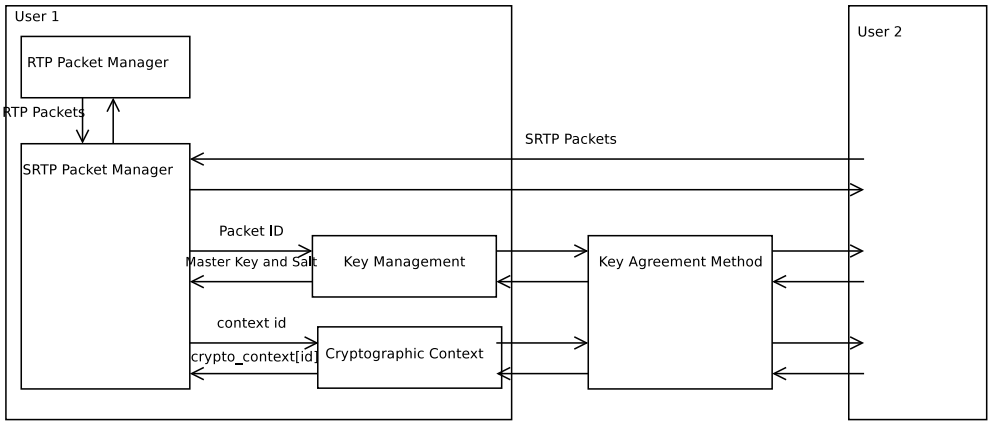

Figure 3.2: Secured media stream using SRTP

Encryption

The encryption is provided by the Advanced Encryption Standard [32] (AES) algorithm. This algorithm was selected for its rather low computational requirements and because it is often implemented in hardware.

AES is used in stream-cipher mode: the algorithm is used in a chain to produce a stream of keys, which is then used as a one time pad to encrypt the data (with a bit to bit logical exclusive-or operation). Figure 3.4 illustrates the use of a block-cipher in stream-cipher mode.

SRTP proposes two modes: the counter-mode [20] and f8 [1] which is used by UMTS encryption.

In counter-mode, AES is applied to consecutive integers to build a key stream. The first of these integers (initialization vector) depends on the source identifier, the packet index, and a salting key. Counter-mode is illustrated in figure 3.5.

In f8-mode, AES is applied in a chain to produce the key stream. The initialization vector (IV) depends on the timestamp, the sequence number,

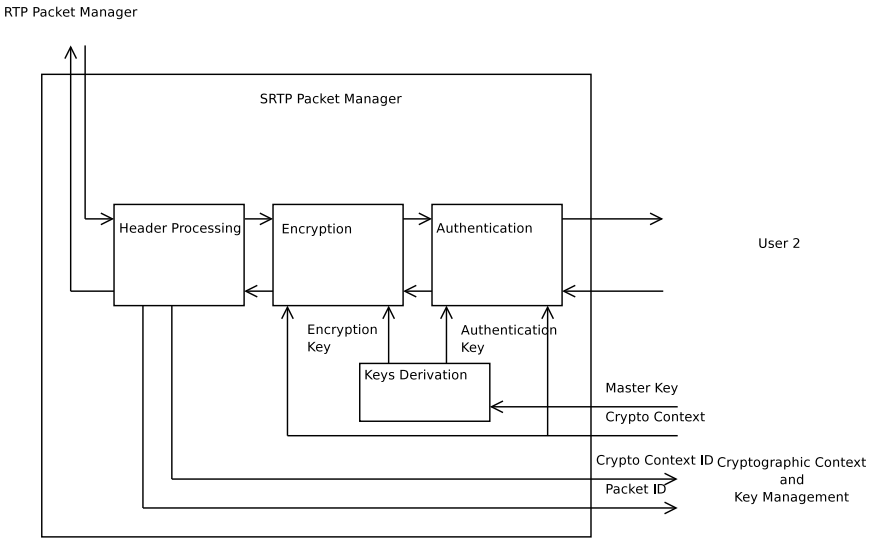

Figure 3.3: SRTP packet processing

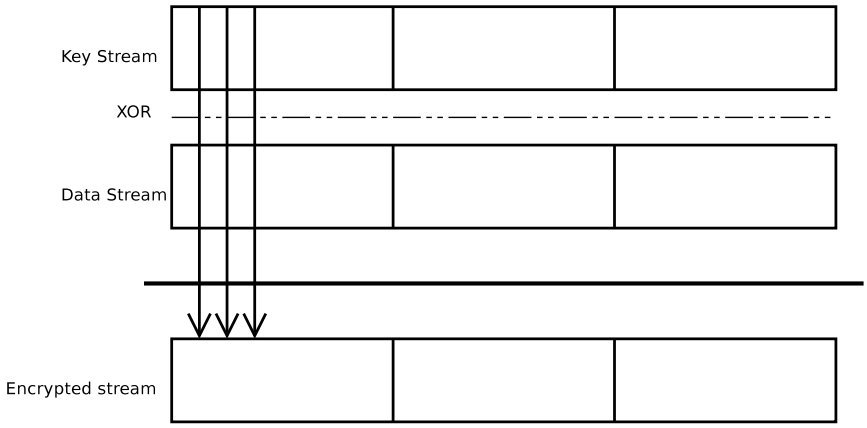

Figure 3.4: Stream-cipher mode encryption

the source identifier, the roll-on counter, and other flags of the RTP packet. f8-mode is illustrated in figure 3.6.

In both cases, the initialization vector IV and the secret encryption key K_e must be shared by the participants.

Note that the header of the RTP packet is not encrypted, so that header compression may be applied. RFC 3096 [8] describes a way to compress IP, UDP and RTP headers.

Encryption is applied to both RTP packets and RTCP packets.

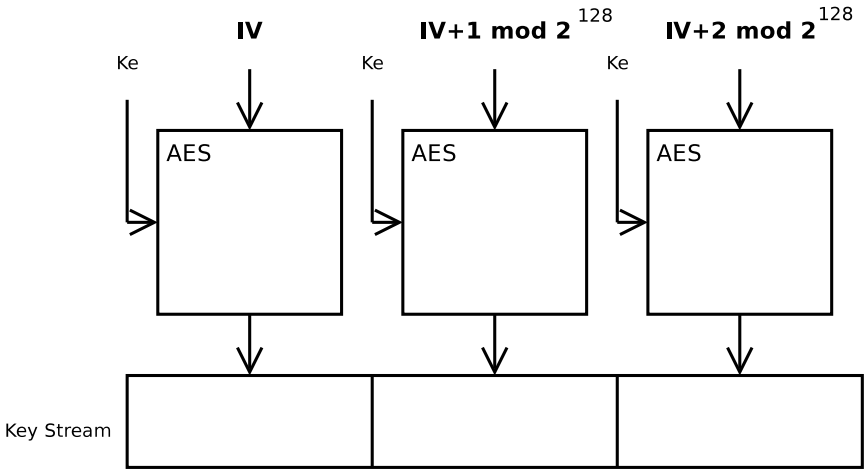

Figure 3.5: AES used in counter-mode

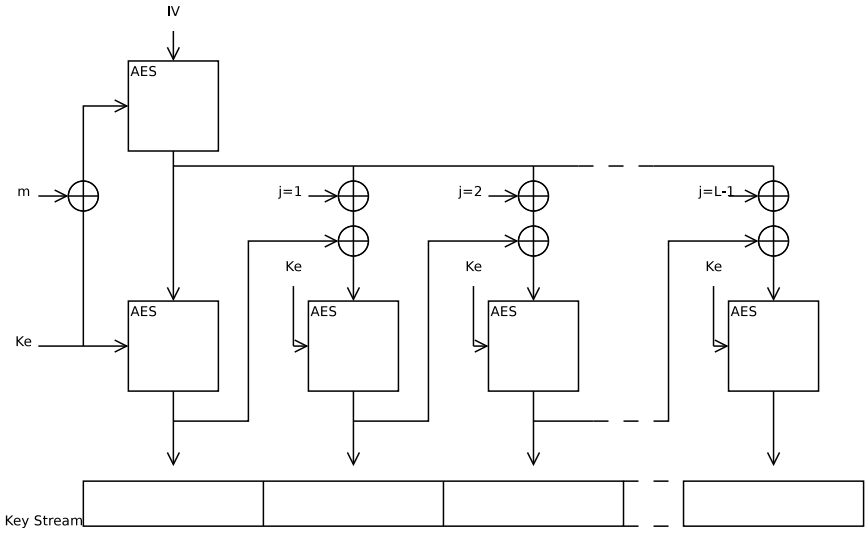

Figure 3.6: AES used in f8-mode

Integrity control

Integrity control is performed using the Keyed-Hashing for Message Authentication [33] (HMAC) algorithm, with the Secure Hash Algorithm 1 [44] (SHA-1) hashing function. The MAC is computed *after* the encryption was performed. It covers both the header and the encrypted payload. To reduce the overhead, the resulting MAC is truncated to its first

4 bytes. By default, the authentication key used should be 128 bits long.

Using a MAC is mandatory for RTCP packets, and recommended for RTP packets.

3.2.3 Encryption at the network layer

Another alternative for protection of the media content would be the use of a network layer encryption. Using an IP Encapsulated Security Payload (ESP) [27], as defined by the IPSec [22] IETF working group, would ensure the required encryption and integrity protection. However, using IPSec for VoIP has some drawbacks: IPSec being usually implemented by the operating system, setting up a new IPSec security association for each call would require a strong interaction between the user agent and the operating system. Moreover, ESP adds more overhead to the packet than for instance SRTP, and IPSec protected packets may have difficulties when going through firewalls, since the transport layer information (ports) are encrypted.

An encryption of the transport layer could also be considered. However, Transport Layer Security [9] (TLS) being based on a reliable transport layer (TCP in most cases), is not suitable for media streaming (a lost packet should not be resent, since this implies additional delays).

3.3 Securing the signaling messages

In order to authenticate the users before the phone call is established, the signaling process must be conducted in a secure way. That requires that each participant in this process has authenticated its partners.

3.3.1 Secured SIP registration

The first step in securing the signaling process is for the SIP registrar to authenticate its users. Therefore, SIP proposes the use of authentication schemes similar to HTTP: a basic authentication and a digest challenge scheme. The basic scheme is being deprecated because of its weakness (the user/password being sent as clear text). The digest challenge authentication is based on a user/password scheme. The server sends a random nonce, and the client answers with a hash based on this nonce, the username and the password. The hash function used is MD5.

This method allows the server to identify the client, but not the client to authenticate the server. For instance a fake server could retrieve the client's username and password by proposing carefully chosen challenges.

This fake server could then impersonate this client. This scheme is also vulnerable to dictionary attacks: the nonce and hash being sent in the clear, an eavesdropper could try to recompute the hash with different passwords, until the values match.

To provide mutual authentication, a Public Key Infrastructure (PKI) could be used with the TLS protocol. This requires a more complex implementation and infrastructure, but also provides a higher level of security. As we will see, a PKI may also be useful in other parts of the VoIP security system.

TLS can perform an authentication of each end of the link (registrar and user agent), but also provides an encryption and integrity control of the SIP messages exchanged. Unfortunately it can only be run on top of a reliable transport layer (TCP is most cases) and adds a significant overhead to the network traffic: the initial three messages of a TCP handshake, and 3 roundtrips for TLS authentication and agreement on keys and cryptographic parameters.

The RFC 3261 [37] describes the use of TLS between the client and the server in single-authentication mode (only the server proves its identity by providing a certificate, the client is then authenticated with HTTP Digest over the TLS link). This permits the user-agent not to have a certificate. On the other hand, such a user-agent is `not` able to act as a TLS server. If the first TLS connection is broken for some reason, and the user-agent does not notice it, the server will not be able to contact it to transmit incoming messages (such as an `INVITE` message). Keeping a connection alive may be an important constraint in some cases, such as device mobility. However, keeping the TLS connection alive significantly reduces delay.

If TLS is used in mutual-authentication mode between the user agent and the SIP registrar, the certificate of the user-agent should point to the Fully Qualified Domain Name (FQDN) of the user-agent host. This presents several drawbacks:

- If a user moves from one host to another, they will need one certificate per host.
- This certificate cannot be used in end-to-end authentication schemes, such as the one provided by MIKEY, for which the certificates should point to the SIP ID.

Using TLS mutual-authentication, with the client certificate designating the SIP ID, would solve those problems. The association user-agent/host is provided securely in the SIP `REGISTER` message because it is carried by the TLS link. Because it owns a certificate, the user-agent may act as a TLS

server, thus allowing the SIP server to reconnect if the connection is closed and incoming messages must be forwarded.

3.3.2 Securing communication between SIP proxies

Once the user-agent and registrar have authenticated each other, the registrars and different proxies engaged in the signaling must also authenticate themselves. This is usually done based on hop-by-hop TLS links.

Each time two proxies want to establish a link, they exchange certificates and check them, for instance with a Certificate Authority (CA). They then establish a secured (TLS) link.

3.3.3 SIPS URI

To allow the user to require a secure SIP transaction, a new type of SIP URI has been created in the latest SIP specifications [37]. A `sips:alice@a.org` type of URI tells the user agent and the proxies that the SIP message should be carried on a secured channel along the whole path. Hop-by-hop TLS links should be established between the different proxies, until the message reaches the destination domain. The (last hop) connection between the responder's registrar and the responder itself, may also be secured, but that is left to the responder domain's security policy. If these conditions are not all fulfilled, the connection setup should fail.

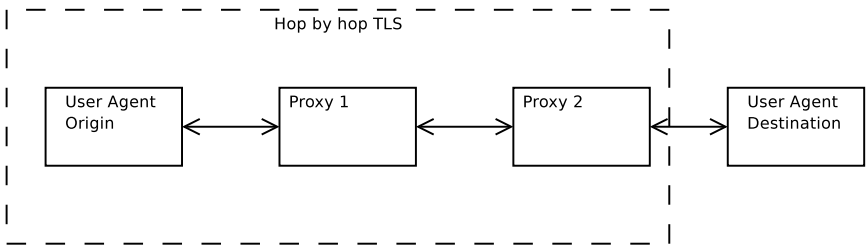

Figure 3.7: SIP secure chain is required when using a SIPS URI

3.3.4 Securing the session description

If protecting the user's identity and traffic information is not considered as an important issue, then securing the whole SIP signaling path may appear to be too much overhead. However, it may still be useful to protect the media description (SDP) contained in the SIP `INVITE` messages. Especially if this

SDP contains non-protected information regarding the exchange of keys and cryptographic parameters. Downgrade attacks (modifying the cryptographic parameters negotiation so that a weak cryptographic scheme is chosen) must be prevented.

The SIP specifications [37] provide a description of how Secure/Multipurpose Internet Mail Extensions (S/MIME) [36] could be used for protection of SIP contents, such as SDP descriptions. This allows encryption, integrity protection, and digital signatures of the SIP contents, using a PKI.

3.3.5 Some words about the requirement for a PKI

Some of the schemes described for protection of both the signaling and the media content can benefit from the presence of a PKI. When using hop-by-hop TLS links, a PKI allows each proxy server to authenticate the next one in the chain, even if they establish a connection for the first time.

A PKI can also help end-to-end mutual authentication of the caller and the callee. For instance, schemes like IKE and MIKEY can use certificates for mutual authentication.

Depending on what level the VoIP solution is deployed, PKIs with different levels of complexity can be considered. If the VoIP network is reduced to one local organization, a simple self-signed certificate could be used as a local CA for the whole organization. This local CA would be used to sign a certificate for each of the VoIP users and servers belonging to this organization. This has the advantage of simplicity and does not require additional costs to get a signature from a *trusted* external CA. However, secure calls would of course be limited to the members of the organization.

When several providers are involved, one solution would be to have each VoIP provider get a certificate signed by a *trusted* CA, and use this provider certificate to sign personal certificates for each of their users and servers. This configuration is shown in figure 3.8.

To avoid the cost for having a commercial CA involved, VoIP providers could setup mutual trust agreements: providers would sign each other's certificate, so that their users could establish secure calls with each other.

Some additional issues regarding the certificate handling are discussed in section 3.3.1.

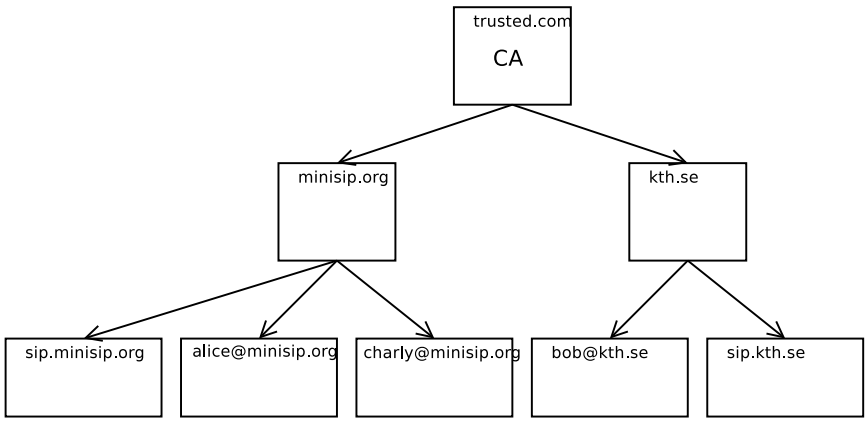

Figure 3.8: PKI configuration involving two VoIP providers

Part II

Key Agreement

Chapter 4

Key agreement requirements

In all the schemes used for securing VoIP, a set of parameters have to be exchanged. Especially for securing the streaming media, an encryption key, an authentication key, a suite of cryptographic algorithms, and a set of other cryptographic parameters must be agreed upon.

4.1 General key agreement requirements

The key agreement phase has to fulfill several requirements, including: to preserve confidentiality of all subsequent traffic, an eavesdropper must not be able to derive the session keys. No attacker should be able to modify the on-going negotiation, resulting in a weaker security association.

4.1.1 Confidentiality

One of the main and most obvious requirement in the key agreement process is the confidentiality of the agreed key. An attacker eavesdropping the traffic should not be able to deduce the exchanged key with a complexity lower than that required to successfully attack the security protocols for which use the key is intended.

4.1.2 Protection against downgrading attacks

Another requirement, which applies to any type of negotiation protocol, is protection against downgrading attacks: it should not be possible for an attacker to modify the messages so that the finally negotiated parameters result in a weaker cryptographic scheme (for example by reducing the size of the key).

The most straight forward protection against this kind of attack is to digitally sign the exchanged messages.

4.2 Optional features

Some additional features may be suitable for a key agreement protocol, depending on the level of security ensured by the security protocol for which the key is intended and the circumstances in which this security protocol is used.

4.2.1 End to end authentication

The key agreement protocol may include a mutual strong authentication of both sides of the negotiation. Depending on the situation, the authentication may have an additional requirement: for privacy issues, it may be required that the identities of both participants are not revealed to a passive listener. One way to fulfill this criteria is to perform first an anonymous key exchange, then use the resulting key to encrypt an exchange of identities and proof of identities. A disadvantage of an anonymous key exchange is that it makes it easier for an attacker to perform denial of services and man-in-the-middle attacks.

The end-to-end authentication in the key agreement can be performed via different methods:

- by digitally signing a part of the message that is session dependent, or a hash of that part,
- by encrypting a challenge with the other party's public key, or
- if pre-shared keys are used, by deriving an authentication key from the shared key, and computing a MAC of the message.

4.2.2 Replay protection

Replay protection may be added to prevent an attacker from replaying a previous eavesdropped key exchange, for instance to identify itself as someone else. Some of those protection methods include timestamps or a sequence number in the exchanged data, or use a cache of previous transactions.

4.2.3 Perfect Forward Secrecy

Many of the key agreement protocols use long term secrets, to process several key agreements. If this secret happens to be revealed later, it is important that it gives as little information as possible about the key exchanges that were processed using this secret key. For example if session keys are encrypted with one's public key, the disclosure of the matching private key gives access to all the exchanged keys.

The Diffie-Hellman type of key exchange is the only currently known type that provides theoretical Perfect Forward Secrecy. For further details on Diffie-Hellman, see section 5.3.

4.2.4 Irrepudiable proofs of communication

Another issue may be that a key agreement session may constitute an irrepudiable proof of communication between Alice and Bob. For example, if Alice has digitally signed Bob's identity, she cannot deny that she has been communicating with him. Conversely, some key agreements are designed to avoid the creation of such a proof.

4.2.5 Protection against Denial-of-Service attacks

Since most of the key exchange protocols require cryptographic computations, they could easily lead to DoS attacks, i.e. an attacker could for example engage a large number of key agreements simultaneously, to reduce the available computational resources of its target. Therefore, no heavy computation should be necessary on the responder's side, before sufficient confidence in the incoming request is established.

Several methods provide protection against denial-of-service attacks:

- The use of cookies: Bob will not start the key agreement until he has sent a cookie to Alice and Alice has returned it. This prevents the use of a connection with a faked source network address from the initiator. Stateless cookies, which allows Bob to know that a specific cookie was sent to Alice without having to keep state information about previously sent cookies, are preferred. Such a stateless cookie could be a hash of Alice's network address concatenated with a secret key.

- The use of puzzles: Upon initial connection, Bob sends a cryptographic problem to the initiator, and delays the key agreement session until the initiator has returned the solution of the problem. The problem might be finding a number whose hash is given.

4.2.6 Easy re-keying and key derivation

Since most of the cryptographic protocols get weaker after a vast amount of data has been used as input with the same key, it is often desirable for a key agreement protocol to provide a simple way to program the negotiation of a new key after a given number of uses of the previously negotiated key. If possible, the new key negotiation should be simpler than the first one.

Another requirement for strong cryptography is the use of a different key per cryptographic function. If both encryption and authentication control are provided by the security protocol, two different and independent keys should be negotiated. This is commonly done by using the negotiated key as input to a Pseudo Random Function that will generate several other keys.

4.3 Some VoIP specific key agreement requirements

In the case of VoIP, several additional requirements are placed on the key agreement process.

4.3.1 Low computational resources

VoIP processing is often embedded into small portable devices, whose computational resources may be lower than typically available on personal computers. Therefore, the algorithms used for the key exchange process (and even more so for the security protocol itself) should consume little computational power.

One way to handle this problem is to use the most common cryptographic algorithms (AES, HMAC-SHA-1, RSA...) for which hardware implementations are available. A chip designed specifically for a given cryptographic operation is likely to require less resources than a general purpose processor.

4.3.2 Low delays for call establishment

In the case of VoIP, the key agreement becomes part of the call establishment process. For the user's comfort, the total time required to establish a call should not be greatly extended by the addition of the key exchange agreement. Refer to chapter 13 for further details on this issue.

Chapter 5

Common key agreement schemes

Several general schemes for secure key exchanges have been conceived to fulfill the key agreements requirements.

5.1 Pre-Shared Key

In this key agreement scheme, Alice and Bob already share a secret key S. They will use this secret key to generate an encryption key k_e. Alice then creates a session key K, encrypts it using the encryption key, then sends it to Bob.

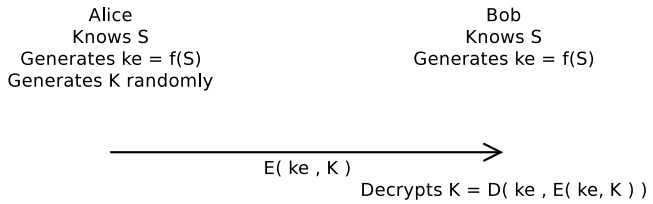

Figure 5.1: Pre-Shared Key agreement protocol

Authentication can be added by deriving a second key k_a from the shared secret key S, and using it to compute a MAC of the initiation message and optional verification message.

Group key agreement can also be performed, if each of the participants shares the secret key S. However, having the secret S shared among more than two persons increases the risk of it being disclosed, and prevents par-

ticipants from being authenticated more precisely than simply belonging to the group.

5.2 Digital envelope

The digital envelope key agreement schemes make use of a public key for the exchange of the secret key. Thus Alice creates a random secret key K, then transmits it encrypted with Bob's public key. Alice then sends it to Bob, who can decrypt it with his private key.

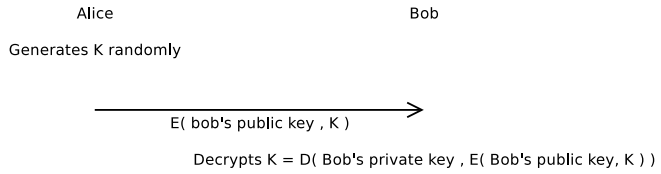

Figure 5.2: Digital Envelope Key agreement protocol

This type of key agreement can easily be extended to a group of users. The initiator generates the secret key, then sends it encrypted to all the partners using their respective public key. However, it requires either a PKI or the pre-exchange of public keys.

The transmitted messages may be digitally signed by the initiator to ensure their authenticity, thus avoiding a man-in-the-middle attack. In the example above, Alice would sign using her private key, which Bob can verify by using Alice's public key.

These two protocols have the drawback that if the private key or the shared secret key were ever to be disclosed, then all the keys exchanged with these keys would be compromised. In other words, there is no forward secrecy.

5.3 Diffie-Hellman

In 1976, W. Diffie and M.E. Hellman published a protocol for secured key agreement on insecure channels. It is based on the discrete logarithm assumption that if p is a prime number, it is "hard" to compute x given $y^x \bmod p$.

The protocol is conducted as shown on figure 5.3: each of the two participants randomly generates a secret number x_i. They then send to each other $g^{x_i} \bmod p$, and compute the secrete key as $K = g^{x_a x_b} \bmod p$, where g is a

generator for the group Z_p, that is a number such that $\forall y \in Z_p \; \exists x; y = g^x$. If p is prime, such a number always exists.

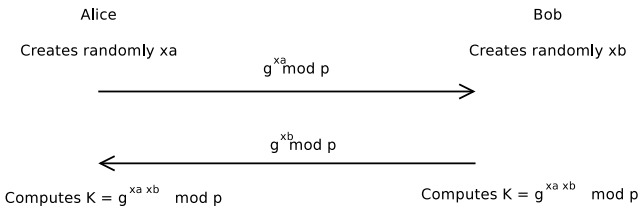

Figure 5.3: Diffie-Hellman key agreement protocol

This key exchange protocol is vulnerable to the man-in-the-middle type of attack; as shown in figure 5.4, an attacker could receive the values from the partners, replace it with its own generated Diffie-Hellman values, on both exchanges. He would then be able to build two secret keys, using the partners values and its own value, and using those keys he could decrypt the received packets with one of the keys and re-encrypt them with the other. Therefore, the Diffie-Hellman values are often digitally signed before transmission.

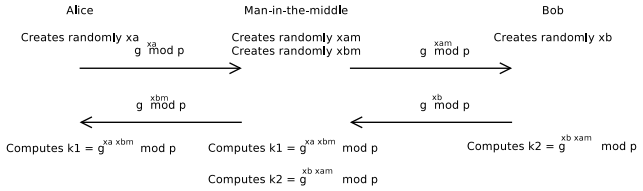

Figure 5.4: Man-in-the-middle attack on the Diffie-Hellman protocol

As stated in section 4.2.3, Diffie-Hellman is the only applicable key agreement scheme which provides perfect forward secrecy. Unfortunately, Diffie-Hellman schemes have the drawback of not providing a method for group key agreements.

Chapter 6

Some existing key agreement implementations

6.1 A general framework for key agreement protocols: ISAKMP

The Internet Security Association and Key Management Protocol (ISAKMP) [28] defines a general framework to implement key agreement protocols, without defining them. This includes the definition of several phases, a set of data payloads that may be exchanged, and the way to transmit these over the usual network transport protocols.

6.1.1 ISAKMP payloads

ISAKMP defines the notion of payloads: a key exchange message is composed of a set of predefined payloads containing specific data required for the process. Among them:

- A fixed header contains the initiator's and responder's cookies, a message ID, and several flags depending on the type of exchange;
- The Security Association payload identifies the secured protocol for which the key is exchanged;
- The Proposal payload carries the cryptographic parameters proposed by the initiator;
- The Key Exchange payload contains the data used in the actual key exchange (for instance the Diffie-Hellman values);

- The Certificate payload is used to send a digital certificate;
- The Signature payload may contain a digital signature of the message; and
- The Nonce payload is used for the exchanges of nonces

6.1.2 ISAKMP over IP networks

Protocols using ISAKMP as framework should use the UDP protocol on port number 500. UDP was preferred over TCP to avoid some common denial of service (DoS) attacks, such as *SYN flooding*. However, the use of UDP introduces another vulnerability to DoS: if big packets are to be sent (as it is often the case in key agreements schemes, when for instance certificates are sent), the IP packet gets fragmented during its transport. An attacker can then flood the receiver with IP fragments, in order to overflow the reassembling system of the victim, thus preventing the establishment of new security associations. Kaufman, Perlman and Sommerfeld [6] describe those attacks and propose several defenses.

6.2 Internet Key Exchange

The Internet Key Exchange (IKE) [18] protocol was specifically designed to handle the key agreement required for the establishment of IPSEC [18] sessions. It is based on two previous key agreement protocols: Oakley [34] and SKEME [21]. It makes use of the ISAKMP framework. All the key agreements are processed with a Diffie-Hellman exchange.

6.2.1 Three kinds of authentications

In the context of IPSEC and Virtual Private Networks (VPNs), authentication of both peers is a major concern: for instance in the latter case, access to a private network should not be granted to intruders. Hence, authentication of all the parties must be performed. Three different schemes are available:

- Digital Signatures are applied to a hash of the negotiated key and other parameters, both by the receiver and the initiator.
- Public Keys: in this case, Alice sends a random nonce encrypted with Bob's public key. Bob uses his private key to decrypt it, and sends back a hash of the decrypted nonce. Alice can then check that Bob was able

to decrypt the nonce and thus authenticates Bob. The same kind of exchange is performed in the other direction to authenticate Alice.

- Pre-Shared Keys: in this situation, both partners share a common secret key. Authentication is performed by computing a hash of the negotiated key and this secret key.

6.2.2 Two phases

IKE is divided in 2 phases. The first one performs one of the authentication schemes previously described, and agrees on a IKE security association (key and security parameters). This association is then used to agree on one or several IPSec security associations, in a faster scheme called **Quick Phase**.

The first phase can be performed in two modes of operation:

- The **Main Mode** requires 3 roundtrips: the first roundtrip negotiates the cryptographic algorithms and parameters, the second is the Diffie-Hellman exchange, and the last is an authentication of the partners and an integrity control of the computed Diffie-Hellman key. This mode is an implementation of the ISAKMP Identity Protection Exchange: by proceeding first with the key agreement, then with the authentication (at the cost of one roundtrip), it allows usage of the negotiated key to encrypt the authentication information and avoids its disclosure to an eavesdropper.

- The **Aggressive Mode** only requires three messages: the first roundtrip combines the negotiation of the cryptographic parameters, the exchange of Diffie-Hellman values, and the authentication of the receiver. The last message provides authentication of the initiator. This mode reduces the number of messages by half, but does not protect the identity information.

6.2.3 Some issues concerning IKE

Some security issues have been highlighted in the IKE protocol. When using the the Pre-Shared Key (PSK) authentication in aggressive mode, the receiver sends a hash of a value depending only on publicly transmitted values and on the PSK. By eavesdropping this value, a brute-force dictionary attack can be performed to retrieve the secret key. The attacker can then authenticate itself with this key, or use it to eavesdrop on further communications. This weakness was described by Anton Rager [3].

Another problem often cited is the high-complexity of IKE. The combination of the different modes and authentication scheme, and the high number of tasks performed (authentication, cryptographic parameters exchange, and key agreement), leads to complex implementations and configuration, and has made inter-operability difficult. Moreover, a complex security protocol is always harder to analyze. Therefore, the IETF has been working on a new version of IKE [26], with simplification as a design goal. Only one mode should remain, and only two roundtrips should be required is most cases. It no longer relies on ISAKMP, as it defines its own payloads.

Chapter 7

Multimedia Internet KEYing

Depending on the situation, a key agreement may have to fulfill very specific criteria. MIKEY [4] is a key agreement specifically designed for protected multimedia exchanges.

7.1 Design goals

The main design goal of MIKEY is to fit the key agreement into the media negotiation process. The latter is usually performed with an offer/answer model using SDP, e.g., the initiator sends her media processing capacities and the responder chooses among the proposed ones which media stream(s) he would like to establish. Thus media negotiation is usually conducted in one roundtrip, therefore MIKEY tries to perform the key agreement and a mutual authentication also within this roundtrip. MIKEY also tries to remain as simple as possible, as opposed to IKE.

MIKEY uses common cryptographic standards (AES in counter-mode for encryption, HMAC-SHA-1 for MAC ...). This makes it easier to find optimized hardware or software implementations.

7.2 Overview

MIKEY provides a way to exchange a Transport Encryption Key (TEK) Generation Key (TGK) and security policies for a Crypto-Session Bundle (CSB), for instance a set of SRTP sessions. It also describes the way to derive a TEK for each of the Crypto-Session (this TEK is the SRTP master key). Figure 7.1 gives an illustration of this principle.

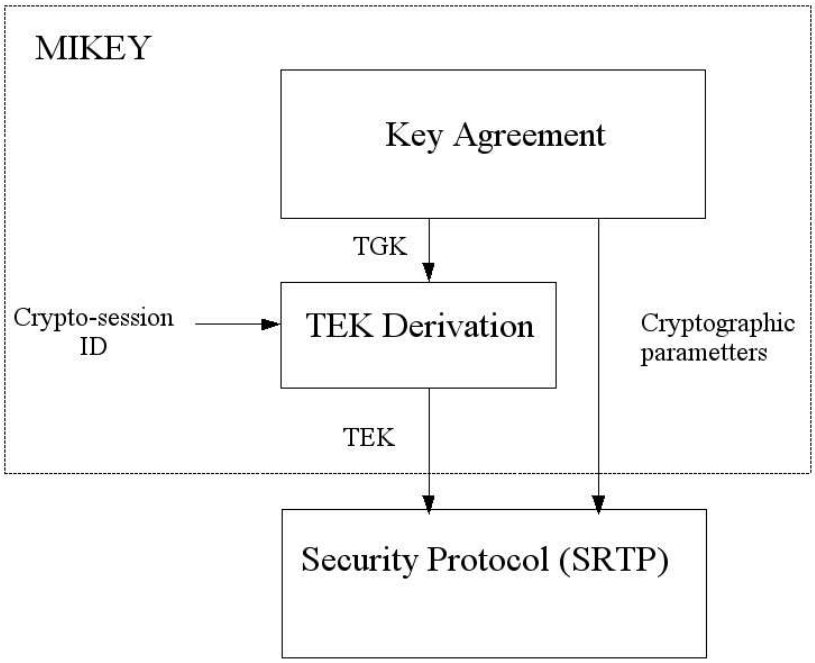

Figure 7.1: MIKEY overview

7.3 MIKEY and security properties

7.3.1 Mutual authentication

A common mutual authentication scheme is to use a set of challenges/responses: each of the participants is given a number and has to perform a one-way operation involving the authentication secret on that number. For example, a hash of that number concatenated with a shared secret or a digital signature of the number, will provide strong authentication. It is important that the challenges are different each time, to prevent replay attacks. Unfortunately, this scheme requires at least three messages for the authentication of the initiator (the initiation message, the responder sending the challenge, and the response from the initiator.

To reduce (by one) the number of messages, thus fitting into the offer/answer model, MIKEY uses timestamps as challenges. Therefore, the initiator knows the challenge and can provide the response in the *initiation* message.

The use of predictable challenges may increase the risk for reflection attack; for instance, if the challenge response only depended on the time and the shared secret, a responder could simply send back the received message, as a challenge response for his identity. Therefore, in MIKEY, the MAC or signature depends on the whole message, including a header that states the type of message (initiation or response), the identity of the sender and the timestamp.

7.3.2 Replay protection

The timestamp used for the authentication challenge/response, is also used to provide replay protection. The received timestamp is stored, and a message is discarded if the same timestamp is used a second time. The number of timestamps stored, as well as the timestamp control accuracy, is considered to depend on the local security policy.

7.3.3 Denial of Services

The usual protection against denial of services (see section 4.2.5) require at least an additional roundtrip. This is not compatible with the design goals of MIKEY. Therefore, MIKEY provides no specific protection against denial of services.

In the case of VoIP, the responder can wait until the phone is picked up before doing any heavy computation, thus providing some de-facto protection. But when the responder is a machine, for example an answering machine a conference server, or a video-on-demand server, other protections should be considered. The use of schemes with low computation requirements, such as pre-shared keys, would be preferred in these situations.

7.3.4 Identity hiding

Identity hiding key agreements requires at least two roundtrips: for instance the first one allows a key exchange and the second one the identity exchange, encrypted with the exchanged key. This is incompatible with the design goals of MIKEY. Therefore, MIKEY does not provide identity hiding, identities are sent in clear text.

If we consider the use of MIKEY within a SIP session, identity hiding would be useless: identities are sent unencrypted in the SIP header. Therefore, identity hiding requires the encryption of the whole SIP message, for instance by using TLS as transport protocol.

7.3.5 Perfect Forward Secrecy

Among the three key agreement types provided by MIKEY, the one based on Diffie-Hellman provides perfect forward secrecy.

7.4 Three types of key agreement

MIKEY provides three different types of key agreements. The choice of using one or the other depends on the available authentication infrastructure (PKI, pre-shared keys, ...) and computational resources.

7.4.1 Pre-shared key (PSK)

This key agreement scheme uses a pre-shared key. It is conducted as shown in figure 7.2. Note that the response message, used to authenticate the responder, is optional. f is a pseudo-random function described in the draft [4].

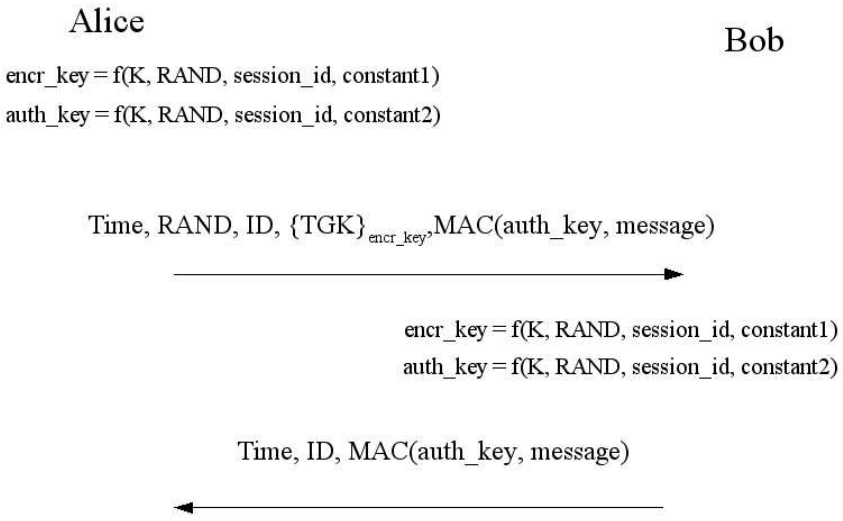

Figure 7.2: Key exchanged based on MIKEY and pre-shared key

7.4.2 Public-key encryption (PKE)

This schemes requires Bob to have a pair of public/private key for encryption, and Alice to have a pair of public/private key for signature. It is similar to the pre-shared key scheme, except that an envelope key (*env_key*) is used instead of the shared key. This envelope key is transmitted encrypted with Bob's public key in the first message. See figure 7.3.

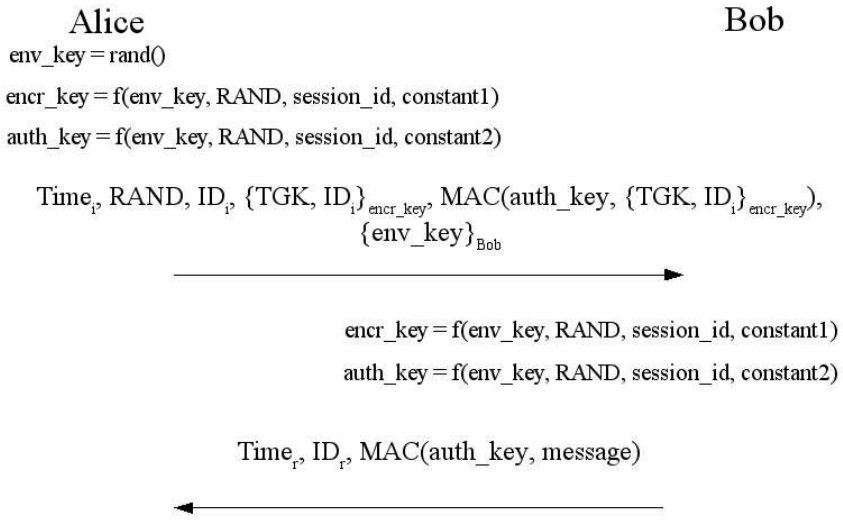

Figure 7.3: Key exchanged based on MIKEY and public-key encryption

7.4.3 Diffie-Hellman (DH)

This scheme requires both Alice and Bob to have a couple of public/private key pair for signatures. The signatures are used both to protect against a man-in-the-middle attack and to authenticate each participant. This scheme requires more computations, but provides perfect forward secrecy.

7.4.4 Cryptographic operations

Each of the schemes proposed in MIKEY requires different types of cryptographic operations. Table 7.1 summarizes these operations. Diffie-Hellman

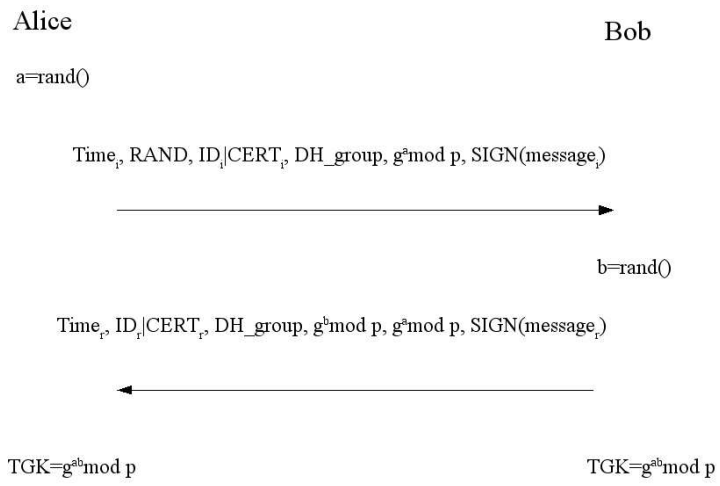

Figure 7.4: Key exchanged based on MIKEY and Diffie-Hellman

requires much more computations, but some of them can be performed in advance (see chapter 13 for further details on the measurements of the computation times).

7.5 Re-keying features

MIKEY includes a protocol to perform an easy and fast re-keying. This is useful since most security protocols (including SRTP) require a renewal of the keys after a certain amount of use (2^{48} packets for SRTP).

	Alice	Bob
PSK		
	• Generation of a random number RAND (at least 128 bits) • Pseudo-random function based on HMAC-SHA-1 • Computation of a MAC, using HMAC-SHA-1	• Pseudo-random function based on HMAC-SHA-1 • Computation of a MAC, using HMAC-SHA-1
PKE		
	• Generation of a random number RAND (at least 128 bits) • Generation of an envelope key • Pseudo-random function based on HMAC-SHA-1 • Computation of a MAC, using HMAC-SHA-1 • Digital signature	• Pseudo-random function based on HMAC-SHA-1 • Computation of a MAC, using HMAC-SHA-1 • Digital signature control
DH		
	• Generation of a random number RAND (at least 128 bits) • Generation of a random number a (1536 bits) • Computation of g^a mod p • Digital signature control • (Certificate control) • Computation of $(g^b)^a$ mod p	• Generation of a random number b (1536 bits) • Computation of g^b mod p • Digital signature • Digital signature control • (Certificate control) • Computation of $(g^a)^b$ mod p

Table 7.1: Cryptographic operations involved in the MIKEY schemes

Chapter 8

Objectives

In our study, we have focused on an evaluation of the recent MIKEY protocol. We will examine the usability of MIKEY in different VoIP scenarios.

8.1 Implementation

When this study started, no public implementation of MIKEY was available. Therefore, the first step was to implement the protocol. This allowed further experiments, and was thought to help discover some interpretation issues in the draft.

8.2 Call establishment delays

Adding security features to VoIP is likely to add some delays:

- to the call establishment (for key agreement and SIP secured transmission) and
- to the media processing (for encryption and authentication).

In our study, we will try to measure the actual influence of the added security in the call establishment delay. The earlier thesis of Israel Abad Caballero [24] measures the effects of added security in the media processing.

8.3 Security in advanced VoIP scenarios

VoIP, and especially the SIP protocol, offers very important flexibility regarding call configurations. This flexibility should not be limited by the

associated key agreement protocol. We will study how MIKEY could be used in some advanced VoIP configurations, specifically group conferencing and mobility.

8.3.1 Group conversations

Group conversations are very easily set up with SIP. Several configurations can be used, such as multicast RTP sessions, centralized multiple unicast sessions, or multiple peer-to-peer unicast sessions. The key agreement protocol should be flexible enough to handle all these situations. Moreover, it should be possible to transition easily and securely from a single point to point session to a group conference configuration.

8.3.2 Security and mobility

VoIP, and especially SIP based systems, allows calls to be transferred from one location to another. We will focus on two types of mobility:

Session mobility: The user switches from one device to another, without loosing the on-going calls.

Device mobility: The user moves from one network to another, without loosing the on-going calls.

In both cases, the security aspects have a major role. The parameters (including the security context) have to be transmitted from the previous configuration to the new one in a secure way. Moreover, denial of service attacks by inducing an unsolicited move to another network or device must be prevented. We will study how MIKEY can be applied to each of these scenarios.

Part III

Implementation

Chapter 9

MIKEY

As the definition of MIKEY was still a work in progress and there was no publicly available implementation yet, a new implementation had to be written from scratch, based on the draft specification [4].

9.1 A GPL library

The implementation was as independent from the chosen user agent as possible, so that it may be reused by other applications. Hence, it was planned so that it could be released as a library, under the GNU General Public License (GPL) [10].

9.2 Implementation design

Since the MIKEY protocol is very object oriented (i.e. a payload object, with common characteristics and several variations), using the C++ or Java languages came as a natural choice. Moreover, because the implementation will be used for delay measurements, it should be fast enough in order to avoid implementation specific problems. Therefore, a compiled language (C++) was preferred.

9.2.1 Objects architecture

The basic object in the library is the `MikeyMessage`. It is inherited by key agreement type specific objects (`MikeyMessagePSK`, `MikeyMessageDH`, `MikeyMessagePK`). These objects are shown in the upper part of figure 9.1.

A MikeyMessage contains a list of `MikeyPayload` objects. A `MikeyPayload` may be a `MikeyPayloadHDR`, `MikeyPayloadT`, etc.

The `MikeyMessage::MikeyMessage()` builds the initiation message.

`MikeyMessage::build_response()` creates a response message from the received message, and `MikeyMessage::parse_response()` processes this response.

An additional class, `KeyAgreement`, was created to handle the key agreement parameters and to store the result of the key exchange. It should be (as much as possible) independent from the MIKEY protocol. The `KeyAgreement` class will contain the TEK Generation Key (TGK) and salt negotiated during the key agreement. The `KeyAgreementDH`, `KeyAgreementPSK`, and `KeyAgreementPK` subclasses add the key agreement type specific inputs (the group and secret key for Diffie-Hellman, the pre-shared key, or the peer's public key).

Some general classes are used both by the MIKEY and SRTP implementations, such as an AES in counter-mode encryption function. The Diffie-Hellman exchange also require some certificate handling functions, for which we use the OpenSSL [35] libraries.

9.2.2 API

One important question was to know at which level the API should be placed. Should the programmer using the library have to care about the actual content of the MIKEY messages, or should he just receive the output of the key agreement or an error indication? The decision of defining three levels of APIs was made to allow an easy and quick use of the library, but also allow more control on the messages' content if needed.

- at the highest level, the application simply asks for a key agreement message when initiating, or gets the received message and sends the response when responding. The keys used must be specified in a library-specific configuration. This layer has `not` been implemented yet.
- at a lower level, the application will choose what kind of key agreement it would like to perform and what keys will be used. This is the API used by the current *minisip* user agent (this is described in chapter 11).
- at the lowest level, the application builds its own MIKEY messages, adding payloads with the provided functions.

9.3 Implementation state

The two first levels of the API have been implemented for both the Diffie-Hellman and pre-shared key schemes. The public key encryption scheme

was temporarily left aside, since it requires a PKI and does not add perfect forward secrecy.

Further development should consider:

- Adding the public key encryption scheme
- Adding re-keying features (SRTP requires a new master key at least every 2^{48} packets).
- Implement the highest level API

9.4 Some issues raised by the implementation

A few practical issues regarding the interpretation of the MIKEY draft [4] have raised during the implementation.

The timestamp payload allows the use of either a 64 bit timestamp value as described in [31], or the use of a 32 bit counter. But the use of this counter is not really described. Note, the initiation vector (IV) used for AES in counter mode is generated using the timestamp, and hence assumes a timestamp of 64 bit is available - no description of what is to be done when using a 32 bit counter is given.

Many security protocols, such as SRTP, require the establishment of both a secret key and a common salt value. MIKEY provides several methods to exchange this salt value. It can be derived from the TGK, or, when using the pre-shared key and public key encryption schemes, provided directly in the encrypted part of the MIKEY initiation message. However, the draft does not state if the choice of which method to use is left to the user, or if maybe the first solution should only be used with the Diffie-Hellman scheme (for which the second solution is impossible, since in that case MIKEY messages do not have *any* encrypted part).

The payloads order in a message is just a recommendation. This makes the implementation a bit more complex in some cases. For instance, the Diffie-Hellman response message includes two Diffie-Hellman payloads, one containing Alice's public D-H value, the other Bob's one. Therefore, Alice has to check the D-H value of each payload and compare it with the one she sent, to determine which payload contains Bob's value.

Other problems are related to the error handling. An error payload is used to transmit a description of the errors which occurred. A set of error types is defined, but this set is very limited. Maybe an error type could be defined for errors that are not covered by the other types.

Some problems are more related to the interaction of MIKEY with other layers, such as SIP. Further details of these are given in section 11.4.

Those problems were reported to Elisabetta Carrara, Karl Norrman, Fredrik Lindholm and Magnus Brolin from Ericsson Research, during a meeting on the 10th of December. These issues should be fixed in the next versions of the specifications.

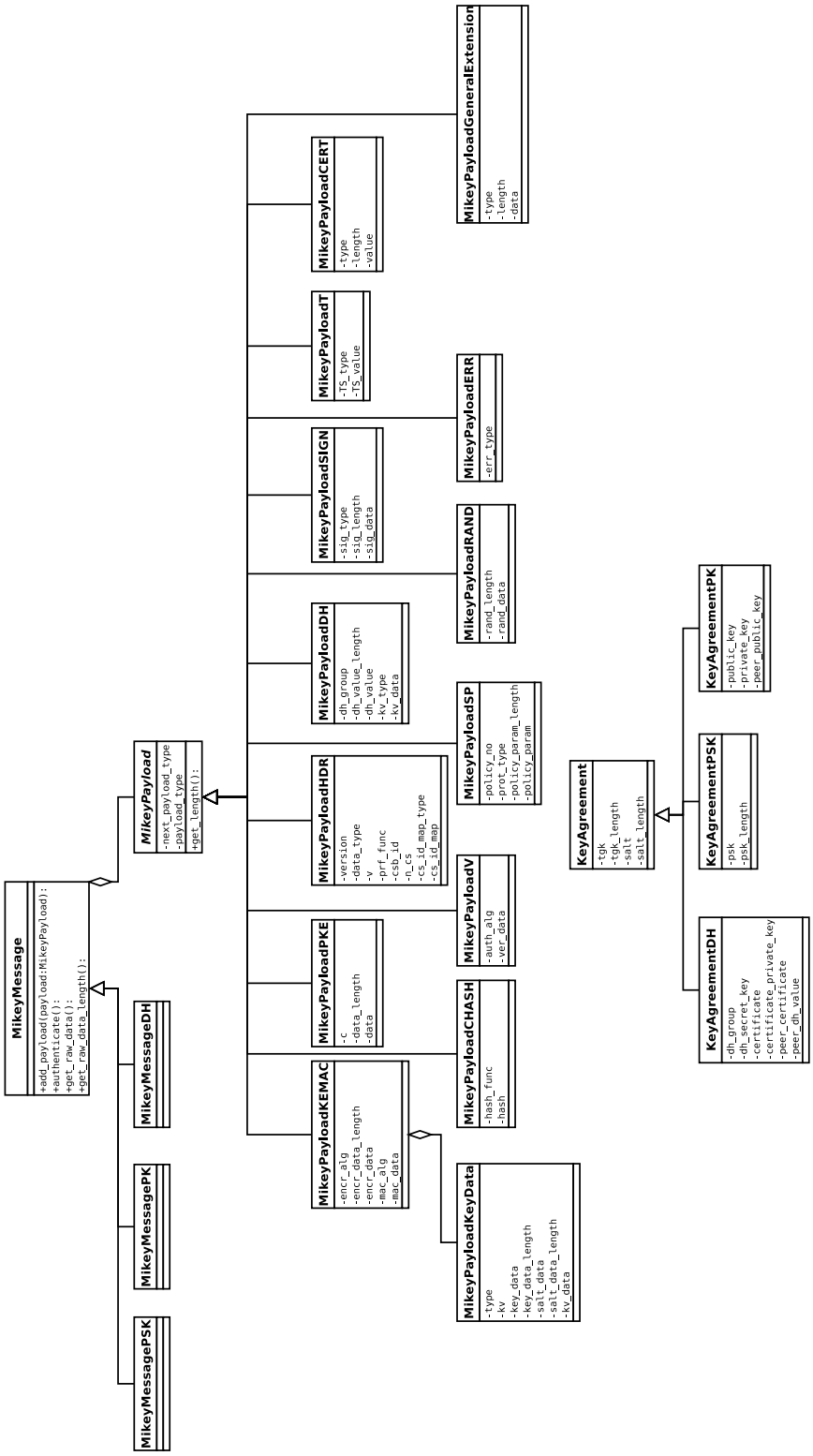

Figure 9.1: MIKEY library design

Chapter 10

SRTP

10.1 Work performed

The SRTP implementation is based on previous work by Israel Abad Caballero [24]. It was previously incorporated in the *minisip* user agent.

Part of the new code was rewritten to eliminate the dependency on the library libsrtp [7]. Specifically, the pseudo-random functions to derive the session keys from the master key and replay protection were added.

10.2 Implementation structure

The `SRtpPacket` is defined as a subclass of the `RtpPacket`. It adds the authentication tag and the optional MKI to the RTP packet.

The `CryptoContext` object holds all the cryptographic parameters related to one SRTP stream (including the corresponding SRTCP stream). When given an RTP packet, it can encrypt it and add the authentication tag, resulting in the corresponding SRTP packet. Conversely, given an SRTP packet it can check its integrity and decrypt its content, resulting in an RTP packet.

10.3 Implementation state

The current implementation allows full protection of the RTP packets. Some additional work is required for protection of the RTCP packets (RTCP support is not complete in *minisip*) and some optimizations could be performed on the stream-cipher generation, specifically adding pre-computation.

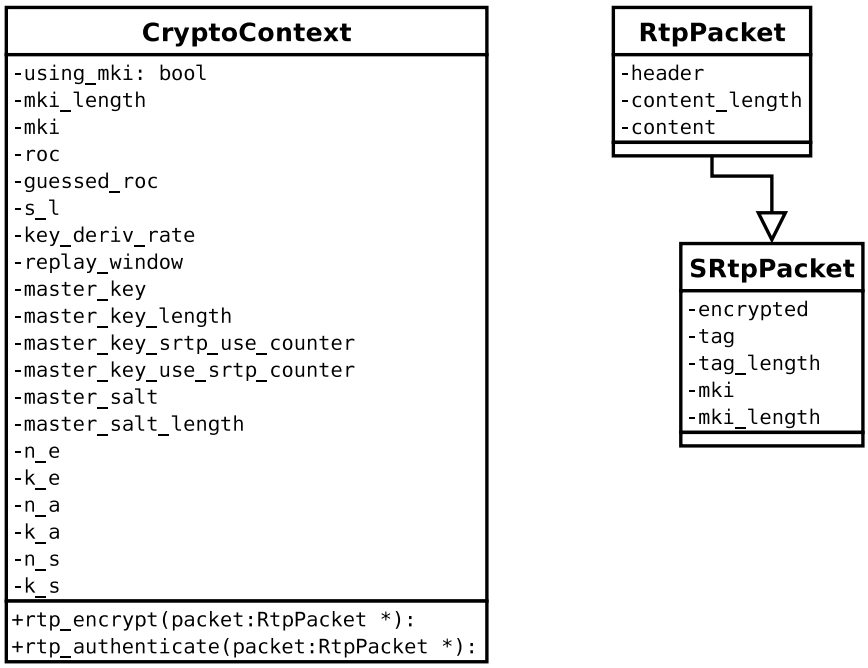

Figure 10.1: SRTP implementation design

Chapter 11

Integration with a SIP User Agent

The integration of MIKEY into a SIP user agent raised some additional issues. These are described in detail below.

11.1 SIP and MIKEY

In order not to add additional roundtrips to the call establishment, the key agreement conducted with MIKEY should be contained in the SIP `INVITE` transaction. The simplest case, when no error occurs, is conducted as illustrated in figure 11.1. MIKEY usually requires no more than one roundtrip, so the whole key exchange can be included in the `INVITE` transaction.

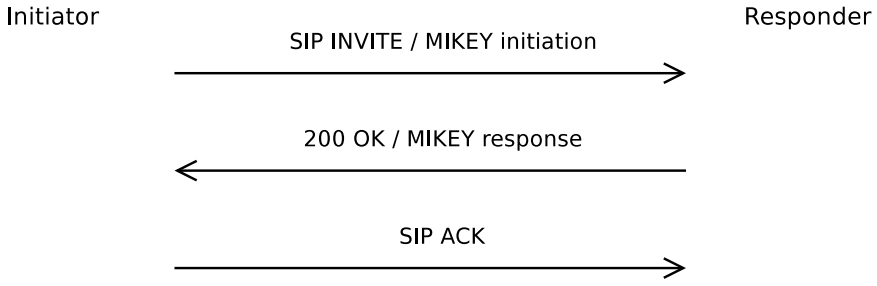

Figure 11.1: MIKEY in a SIP `INVITE` transaction

An Internet Draft [25] defines how MIKEY should be integrated into the SIP messages, as part of the session description. The MIKEY messages are added as an SDP attribute `key-mgmt`, either at the session level (in which

case it applies to all the streams), or at the media level (if it is specific to protecting only one stream).

Several `key-mgmt` attributes can be included if the initiator wants to offer several alternative ways of exchanging keys. To avoid a downgrade attack, each key agreement message must contain the list of the key agreement methods and protect its integrity. In the case of MIKEY, this is done by adding a GENERAL EXTENSION payload containing this list.

Re-keying should be included into `RE-INVITE` messages. However, this has not yet been implemented.

11.2 User interface

A user interface for setting security parameters has been created. The user is given a choice of security agreement methods that can be enabled. For each of them, some fields must be completed (certificates for Diffie-Hellman, Pre-Shared Key). If at least one type of key agreement is enabled, the user is then given the possibility to establish secured out-going calls.

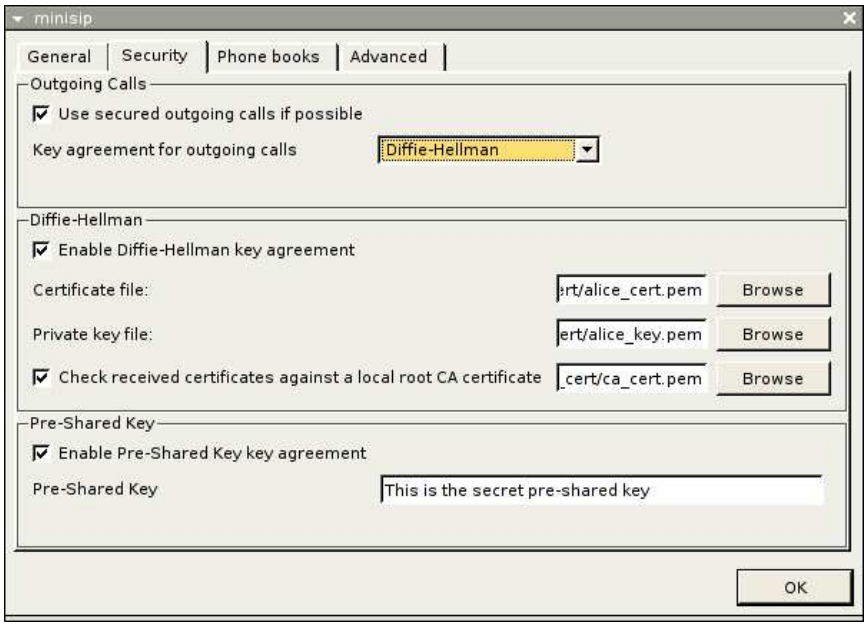

Figure 11.2: Security settings in a graphical *minisip* user interface

11.3 Interaction with non-secured user agents

Some open issues are raised when dealing with compatibility with non-secured SIP user agents. In our example, Alice wants to establish a secure call with Bob, but Bob's user agent does not have any security features. Thus Alice will include a MIKEY message which will be discarded by Bob's user agent. Bob will send a SIP OK response, without any MIKEY message. What should be Alice's user agent's behaviour? Several alternatives may be considered, depending on Alice's security policy.

A first solution could be to ask Alice if she wants to fall back to an unsecured call if Bob's response shows that his user agent could not handle the secured call offer. If Alice agrees to fall back to a non-secured call, her user agent continues with a non-secured call initiation (for instance by answering with a usual ACK). If she refuses it, a CANCEL request is sent to end the call. But this leads to some unresolved issues:

- Requesting user interaction in the middle of a transaction does not fit well in the SIP protocol: a transaction should normally time out after a fixed amount of time, so the user would have only a limited amount of time to answer the query.
- If the call is (finally) canceled, Bob may be called, but the call will be canceled as soon as he answers it. Bob will not even know why the call was canceled.

Another option is to ask the user before the call is established, if security is required or just preferred. This could be part of the user's configuration.

- If Alice would prefer to establish a secure call, but if it is not possible then fall back to a non-secured call, she may propose two streams in her SDP offer: one that uses SRTP and has a `key-mgmt` attribute, and another one that uses RTP. Bob's user agent should then reject the SRTP offer (by putting its port to 0), and accept the RTP one. This requires strictly compliant user agents.
- If Alice requires the call to be secured, she will only include the SRTP stream in her SDP offer. The user agent should then send a 606 (Not Acceptable) response to inform her that it is not able to handle any of the proposed media streams. Therefore the call initiation ends. If Bob's user agent is not standard compliant and sees the SRTP offer as a normal RTP stream, and answers a `200 OK`, Alice's user agent should then send a `CANCEL` request to end the call.

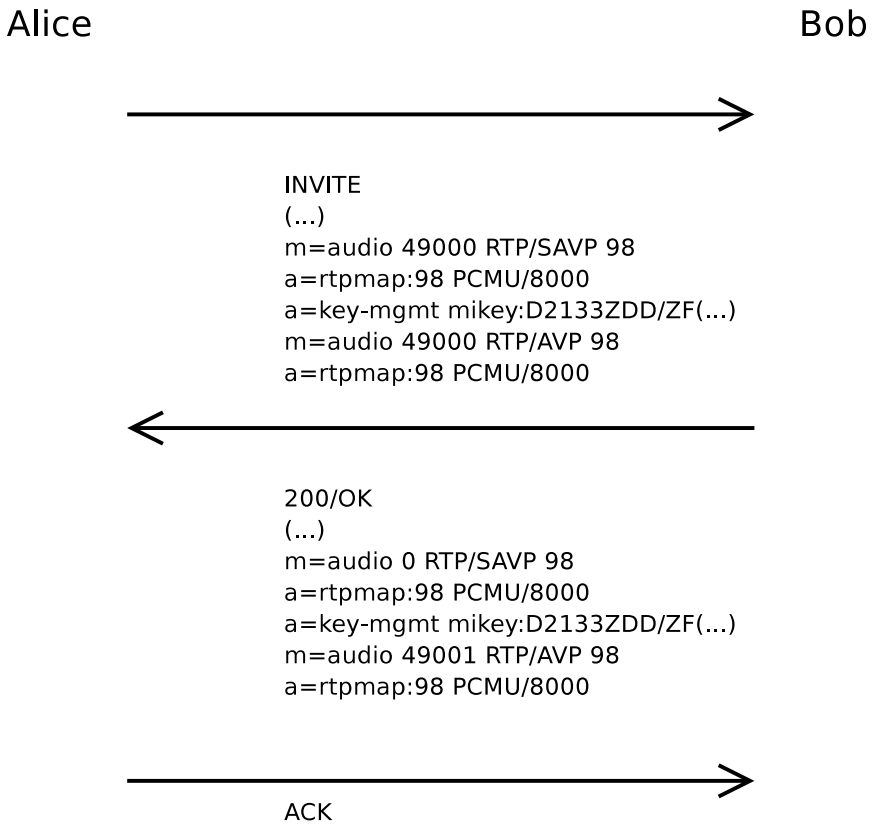

Figure 11.3: Call establishment with fall-back to non-secured call

11.4 MIKEY error handling within SIP

Other issues appear when handling MIKEY errors within a SIP transaction. The MIKEY specifications states that a MIKEY error should be reported with a SIP message. But this does not always fit well into a SIP transaction.

If for instance the error is detected in the MIKEY response message included in Bob's SIP `200 OK` message, Bob's user agent would not be expecting a SIP response at that instant (since it has not sent a new SIP request). A solution (the one implemented) is to then send a `CANCEL` request instead, containing the MIKEY error message.

Finally, the question was raised of what the behaviour of the user agent should be if only one out of several MIKEY messages was erroneous. Should the whole bunch of streams be rejected, or only the one with a wrong MIKEY message? This can be considered to depend on the local security policy. The safest solution is to reject all the messages.

Figure 11.4: Call establishment without fall-back to non-secured call

Chapter 12

Additional development

Some additional development was performed, to allow a fully secured call to be conducted from a handheld device, specifically an HP iPAQ h5550 running Linux.

12.1 TCP and TLS SIP transport

In order to perform a fully secured call, included secured call establishment, the TLS support by the user agent is required. Hence both TCP and TLS transport were added to the *minisip* user agent.

A connection oriented transport protocol such as TCP and TLS is handled in a different way than the UDP protocol. For instance, the connection should be established once, then used to transmit different SIP packets. Therefore, a pointer to the transport connection should be added to the transaction object, so that it can, for example, send a response using the connection on which it received a request.

Another major difference from when using UDP transport, a SIP message may be divided between several `read` calls on the file descriptor, so that a buffer must be kept so that a complete SIP packet can be processed. Moreover, since the SIP packet does not include a length header, it has to be fully parsed (in order to find the `content-length` header, the end of the headers and beginning of the content) to determine if the packet is complete. This requires mixing the transport and semantic layers in the SIP user agent, which is not very practical for the programmer. Having the length of the packet placed at a fixed offset would make implementations easier.

Using TLS comes with additional issues. Even more so than with TCP, the user agent should reuse the same TLS connection as much as possible to avoid an overly long connection establishment delay. Certificates should

be verified only when connecting for the *first* time to a host. A database of trusted certificates should be kept. These certificates could also be aged out of the database to force re-verification, with a duration set by the local security policy.

Other issues regarding how certificates are handled when using SIP over TLS are discussed in section 3.3.1.

12.2 ALSA sound support

Advanced Linux Sound Architecture (ALSA) [2] is a free software project which aims to provide Linux with a set of high quality drivers for soundcards, as well as an advanced API to use them (libasound). With version 2.6 of Linux, ALSA will become the official sound drivers, replacing the aging Open Sound System (OSS) .

ALSA usually provides better support for the full-duplex mode, in which the soundcard is able to play and record sound at the same time, independently. This is of course a required feature for VoIP, and therefore using ALSA in *minisip* is likely to allow more hardware to be used. ALSA also allows more precise control of the soundcard, which is useful when trying to set the hardware buffer size to an optimal value.

ALSA support is now selected at application compilation time (with a `--enable-alsa` flag to the `configure` script). In future versions, having the sound output loaded as dynamic modules would allow a user to switch between an ALSA soundcard and an OSS one without having to recompile the application.

12.3 Port to iPAQ

During our work, we were given several HP iPAQ h5550 PDAs for development. These handheld devices have built-in Wi-Fi support (802.11b) and Bluetooth. Moreover, they were clearly designed with the idea of being used as telephones: the microphone is placed in the bottom, so that it comes just in front of the mouth when the speaker is placed on the ear. It also has a built-in vibrator, which can be controlled by software. So this device really was the perfect platform to build a prototype of wireless IP telephone.

The iPAQ came with Microsoft Pocket PC installed. Since *minisip* was originally developed for Linux, the easiest way to start the port was to use Linux on the iPAQ. Linux also offered more flexible configuration and in theory only required a re-compilation for the ARM platform. Additionally,

it was much easier to get development and support from the Linux community than to look for information in MICROSOFT's knowledge database.

The `handhelds.org` project centralizes all the current development of Linux on handhelds devices. For instance it hosts the FAMILIAR LINUX [16] distribution, which provides a full operating system for iPAQs, based on the Linux kernel and the Debian [39] ARM distribution. This distribution includes two desktop environments: GPE [14], based on GTK [12] and OPIE [15] based on Qt [42]. Since *minisip* uses Qt, we chose to focus on integration with the OPIE environment. When our work started, our model of iPAQ was not yet very well supported by the Linux kernel. Especially the touchscreen drivers had a problem which made the device crash on the first stroke. But thanks to the impressive work of the `handhelds.org` project, and to the release by HP and Samsung of some additional technical specifications, those problems were fixed in the beginning of November.

We chose to cross-compile *minisip* on an i386 platform. Compiling on the iPAQ is theoretically possible, but a compilation environment require too much space to fit on the limited built-in storage of the iPAQ, and the rather small computational resources available on a handheld device would have made the compilation very slow. Another option was to use ARM compilation farms offered by both HP and the `handhelds.org` project. But having our own compilation environment is more flexible, since we could always cross-compile additional libraries.

We used a toolchain provided by SHARP [38] for development on the Zaurus platform, which is also based on ARM. Other toolchains were tried (such as the one provided by `handhelds.org`), but some problems were encountered, such as incompatible standard C++ libraries between the iPAQ and the toolchain. The additional necessary libraries were cross-compiled (libcrypto, libssl [35], qt [42]).

Compiling *minisip* for the iPAQ platform required some modifications. The version of Qt provided with FAMILIAR LINUX was version 2.3, while we had done our previous development using version 3.2. Therefore some functions and classes were not available, and had to be replaced or removed. Unfortunately, the embedded version of Qt is compiled without support for C++ exceptions. This was and remains a big problem for us, since *minisip* uses them in many places. Some changes were made to avoid throwing exceptions, and *minisip* now compiles for the iPAQ. However, some work remains to be done to completely remove the use of exceptions, or to find a workaround.

Additional development was also required to make the user interface more suited for the iPAQ screen and inputs. For example, the vibrator is used to signal an incoming call and the settings dialog was reduced to fit on the

smaller screen. For future development, the call dialog that is now used to represent a call should be embedded in the main window, because several windows are not very handy on handheld devices. The iPAQ application keys could also be used in a more efficient way.

Some people felt worried about having the iPAQ with its Wi-Fi antenna near their brain when in a call, although we believe that it should not have worse consequences than a typical GSM phone, which sends much more powerful radio signals than a Wi-Fi interface. One solution is to use a Bluetooth headset. This should be supported on those iPAQs, but some development is required to use the headsets' button to answer calls or to hang up. A normal headset could also be used, the built-in audio jack having support for both input and output.

Today, *minisip* is working very well on iPAQ. The sound quality is surprisingly good. Power consumption remains a problem, but things should be improved when the power saving features are better in Linux.

Part IV

Call establishment delays

Chapter 13

Call establishment delay

Adding a key agreement scheme to the call establishment is likely to add delays. In the case of MIKEY, the key agreement results in no additional network transmissions [1], but some cryptographic operations (see table 7.1) which may require significant additional computation time.

The effect of adding security features was measured, and the results have been presented in a paper co-written with Jon-Olov Vatn and Erik Eliasson. The paper is currently under review for the WiOpt'04 [43] conference. My contribution to it includes measurements, thoughts on the inclusion of MIKEY in the SIP messages, comments on the results, and the conclusions. The paper is included in Appendix B.

Our main conclusion is that adding the key exchange to the call establishment does not add significant delays. In the most requiring case (Diffie-Hellman with no pre-computation), the additional delay is 290 ms, which is considered as acceptable for the user. Note that since computations are done in parallel with network transmissions, the network delays will often be the limiting factor.

These additional delays depend on the key agreement scheme used: Diffie-Hellman requires longer computation time, however many operations can be done in advance (choice of the secret values a and b, and computations of g^a and g^b (see section 5.3 for details on the Diffie-Hellman scheme). The pre-shared key scheme adds a delay of less then 50 ms.

[1] Not considering any additional network traffic regarding certificate verification

Part V

Security and advanced VoIP scenarios

Chapter 14

Security in group conversation

The SIP protocol provides simple solutions for group telephony. The MIKEY specification [4] gives some examples on how to use the protocol in these cases. We will review them and describe some additional schemes.

14.1 Peer-to-peer small group conferencing

The simplest solution, which is well suited for small groups, is to set up peer-to-peer calls, resulting in a chain or grid of participants. This solution is very flexible: the conversation may start with two participants, then one of them decides to call a third one, and so on. An example is illustrated in figure 14.1.

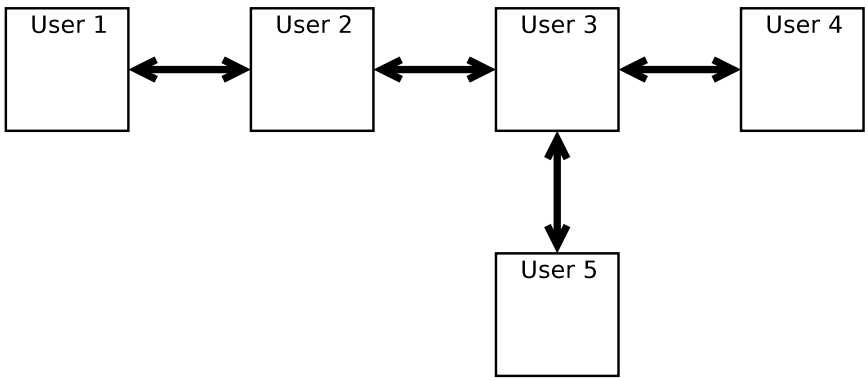

Figure 14.1: Peer-to-peer conference configuration

However, this solution does not scale very well: long chains may result in significant delays between distant participants; a participant connected to an

important number of peers will require much media mixing processing. For large groups, a dedicated conference server, as described below, is preferred.

The security mechanisms described for a two-party call, can be extended to this peer-to-peer conference scenario. Each link in the grid is established using the usual SIP `INVITE` dialog, therefore a MIKEY key exchange can be performed for each of the links. This results in a unicast SRTP stream for each link.

MIKEY [4] gives the example of a three-party peer-to-peer group, in a closed chain. In this example, the stream going between the three parties is the same, and is thus protected with the same key and parameters. Therefore, MIKEY uses the same session ID, TGK and cryptographic parameters for each security association. This prevents the use of the Diffie-Hellman scheme, for which the resulting key depends on two participants, so it cannot be used to transmit a specific key already used in an on-going session.

Another solution is to have different streams for each link (for example avoiding a participant from receiving his own voice). In that case, nothing prevents each link from having different security associations, and thus the Diffie-Hellman scheme can be used for each. A straight forward drawback is the higher resource consumption required for each user agent, since a different encrypted stream must be created for each connected participant. Israel Abad Caballero's results regarding encryption performance [24] show that creating a small number of encrypted outgoing streams and decrypting the same number of incoming streams, should be possible on modern hardware.

14.2 Multicast session

A multicast session corresponds to a one-to-many situation, typically a multimedia presentation being held by one participant to several (largely) passive receivers. This configuration is shown in figure 14.2.

In this situation, each receiver must share the same session key, since only one encrypted stream is created for all the receivers. The Diffie-Hellman scheme cannot be used, since the sending participant must be the only one influencing the choice of the key. MIKEY [4] proposes the use of MIKEY pre-shared key and public key encryption schemes, initiated by the sending participant, and establishing the same TGK for each receiver.

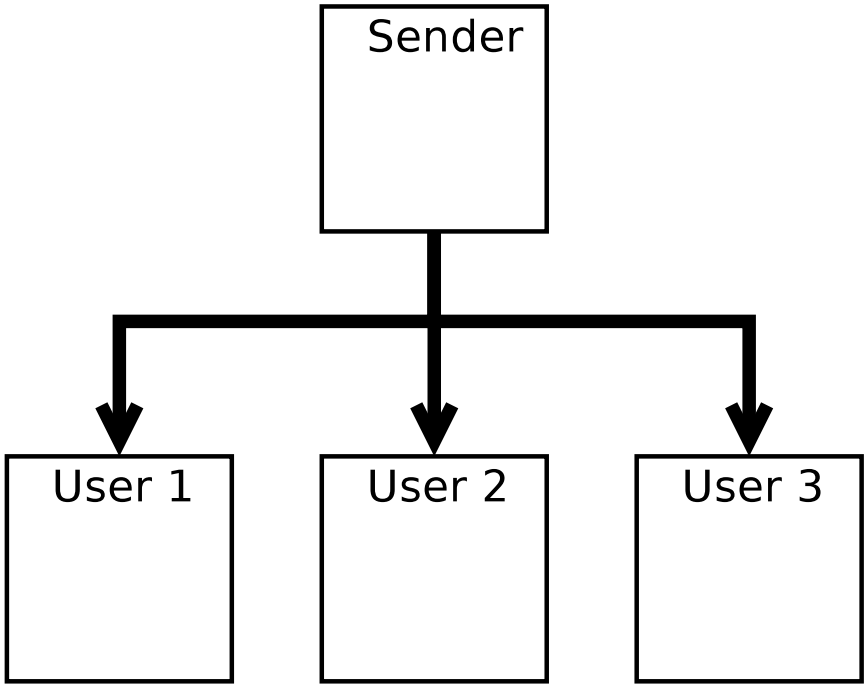

Figure 14.2: Multicast conference configuration

14.3 Conference server

When the number of participant increases, using a dedicated conference server may help to decrease the computational resources required by the user agents. The server will require higher resources, but the operations involved (mixing of the incoming streams, encryption of each outgoing streams with different keys) could be parallelized and divided internally in a cluster. Figure 14.3 illustrates this configuration.

In this case, the user agents would initiate a MIKEY key exchange with the server when connecting to it (for instance through a usual SIP `INVITE` dialog). The resulting session keys can be different for each server/user agent link, since the stream sent through those links will differ anyway (a user does not want to receive his own voice).

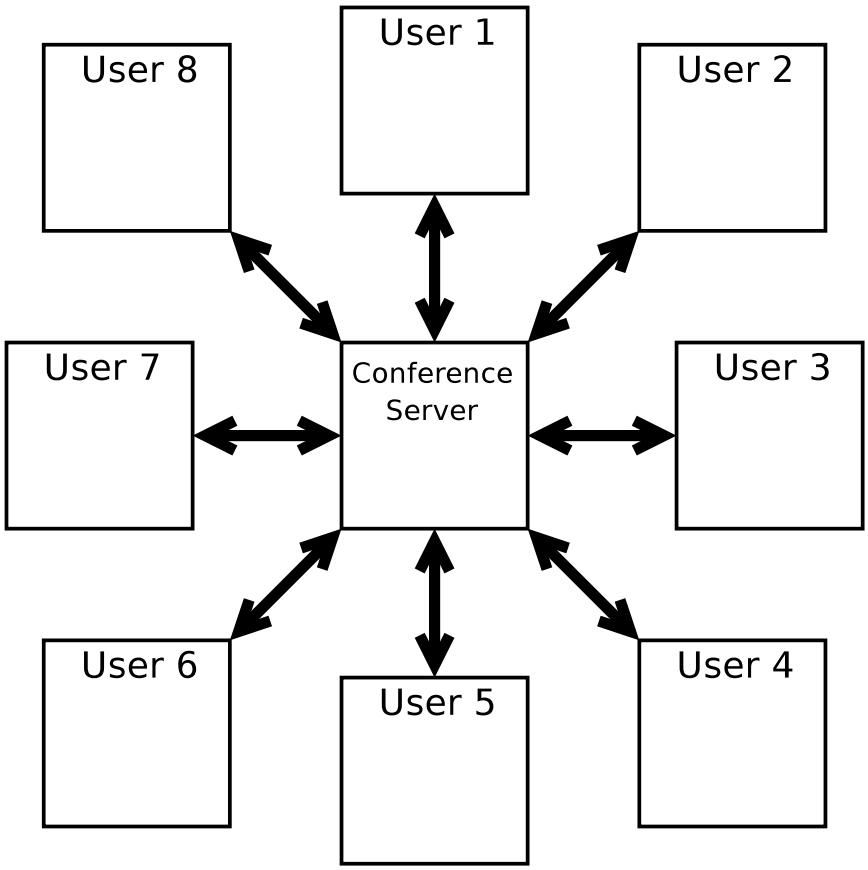

Figure 14.3: Conference involving a central server

Chapter 15

Security and mobility

VoIP allows much flexibility regarding mobility of its users. SIP for instance, allows users to switch devices (session mobility), by asking their partner to recontact them at a new location, or to move between networks, re-inviting the partner from the new network location (from SIP point of view those are identical). Schulzrinne and Wedlund describe [19] how SIP handles mobility. We will try to add security context transmission to these scenarios, when the transition occurs within a call.

15.1 Security and session mobility

Session mobility is the ability for a user to move between two devices, while continuing on-going calls. For instance, a user could start a call using his fixed telephone, then move the session to his mobile telephone.

SIP provides a simple solution for session mobility. The `REFER` method [40] is used to tell a partner (Alice) she should contact a specific SIP URI. A typical application is the transfer of on-going calls from one device to another: if Bob wants to transfer a call with Alice from his laptop (10.0.0.1) to his PDA (10.0.0.2), he will send a `REFER` message to Alice, asking her to re-INVITE him on the new device.

Keeping the same session parameters (media formats and cryptographic context) makes the transition transparent to Alice's user agent media layer, and allows the stream to be transmitted to both of Bob's devices, while Bob moves from one to the other. To transfer the media parameters, the new `INVITE` message sent by Alice could contain an SDP description of the on-going media session. This requires that Bob's user agents are both able to handle the media format. MIKEY allows the transmission of the required cryptographic parameters to join an on-going SRTP session (for

instance when a new receiver joins a multicast session). This feature can be used in our case, by including a MIKEY initiation message in Alice's SDP description. Note that Alice has to decide the TGK (the same as the one used in the on-going session), therefore the Diffie-Hellman scheme cannot be used in this situation. This scheme is illustrated in figure 15.1.

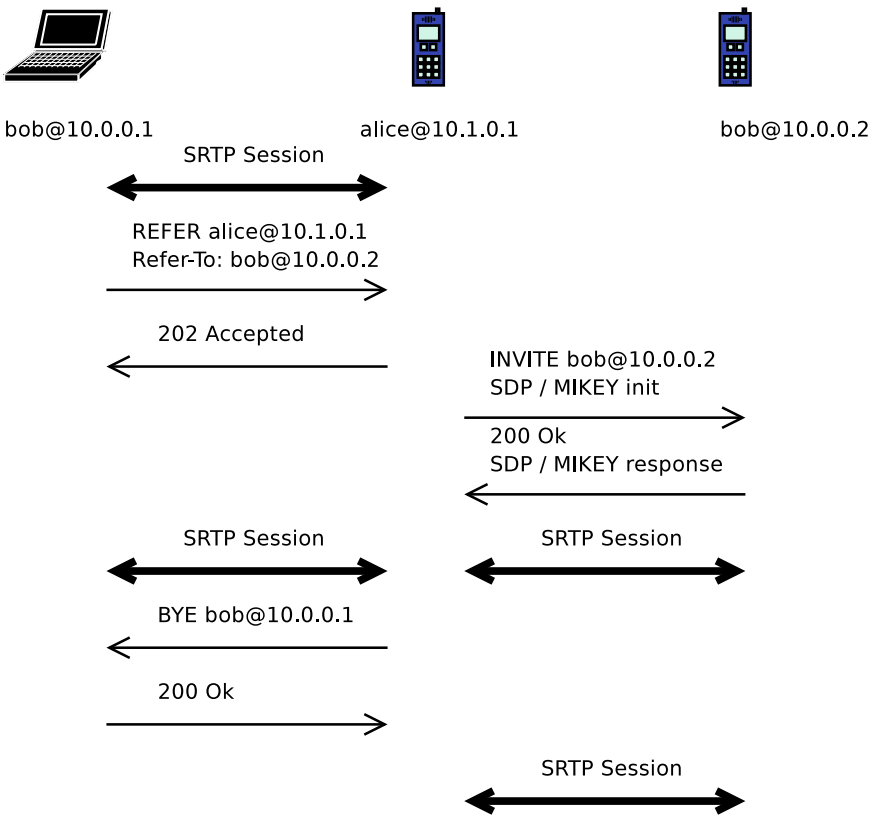

Figure 15.1: Secure session mobility using SIP

The MIKEY response message, sent by Bob's second user agent, allows Alice to authenticate Bob's new terminal. This prevents an attacker from hijacking the on-going session, by sending a `REFER` request to its own terminal (even though he would not be able to decrypt the SRTP stream).

15.2 Security and device mobility

Device mobility is the ability for a device to move from one network to another, thus changing its network address, while still being reachable, and

in our case while maintaining an on-going call.

A common scheme for this situation is the use of Mobile IP: IP tunneling is used to make Bob always reachable at a given IP address. However, this introduces additional transmission delays (the traffic incoming to Bob goes through a third host (home agent), and the tunneling adds overhead to the traffic (an additional set of IP headers).

SIP provides a simpler way to handle this situation, for the specific case of VoIP. Schulzrinne and Wedlund suggest [19] that if Bob detects he has been assigned a new network address, while he is in a call with Alice, he should send a new `INVITE` message to Alice, in which he gives a new `Contact` header and a new SDP description, both refering to the new network address. Adding a MIKEY message to that SDP description allows Alice to authenticate it, and prevents an attacker from initiating unsolicited call transfers. This MIKEY message does not have to carry a TGK and cryptographic parameters, since the old key and parameters can be re-used. There is also no need for a MIKEY response message. Secure device mobility is shown in figure 15.2.

During the transfer, some SRTP packets will probably be lost, since it will take some time for Bob to realize he has moved to a new network, and for Alice to be notified. However this should not affect the security context attached to the SRTP streaming, which can handle packet loss.

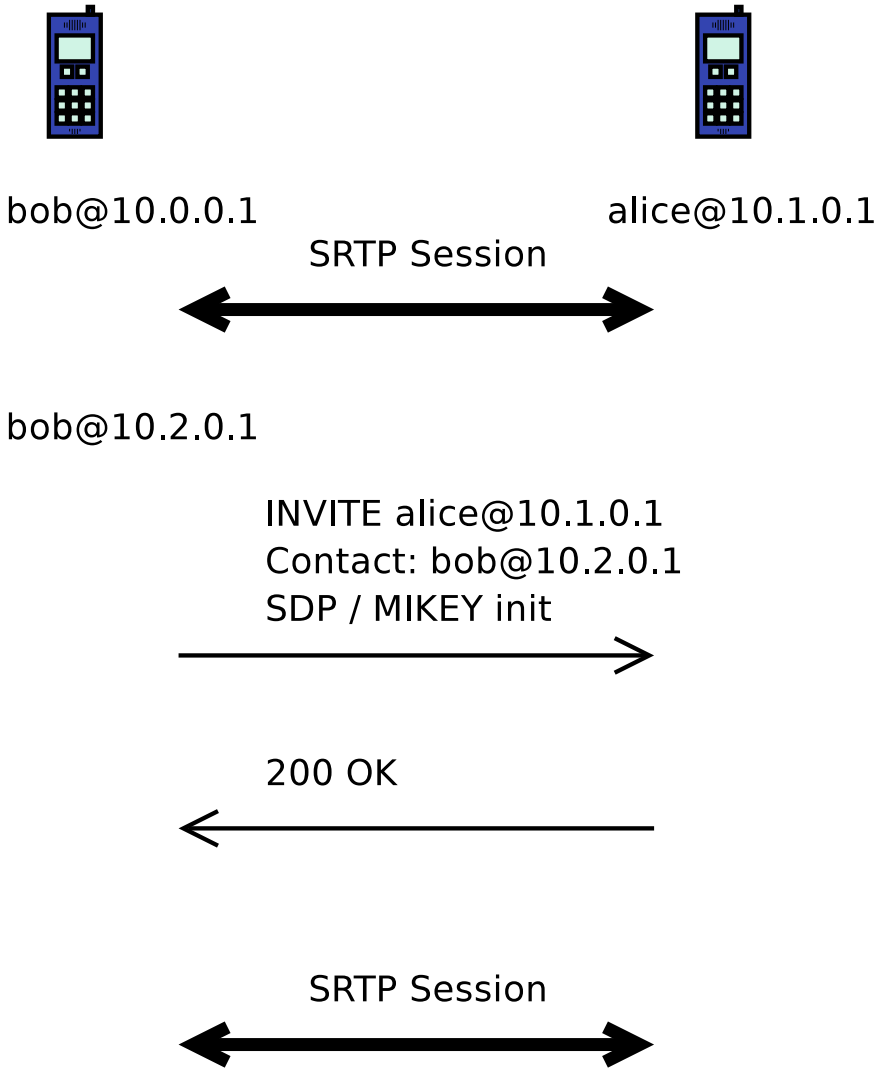

Figure 15.2: Secure device mobility using SIP

Chapter 16

Conclusions

In this thesis, we have presented a model for *secure* VoIP, with a focus on the key agreement protocol. After reviewing some requirements and properties of usual key exchange schemes, we tried to evaluate a work-in-progress definition of a protocol, Multimedia Internet KEYing, for the specific purpose of VoIP protection. An implementation of the protocol was written; this implementation was used to measure the additional delay required to process the key exchange during the call setup.

The MIKEY protocol gave full satisfaction for the purpose of authentication and key exchanges for VoIP calls. It offers both performance, adding little delay to the call establishment, and flexibility, allowing several types of infrastructure (shared key or PKI). Its simplicity should facilitate interoperability easy, once the small interpretation issues raised by our implementation are resolved.

This thesis also tries to show a complete implementation of end-to-end security association for VoIP calls, as a proof of concept. It builds upon earlier work from Israel Abad Caballero [24], and on the *minisip* user agent by Erik Eliasson. The *minisip* user agent is now able to set up encrypted and integrity protected calls, using a dynamic key exchange. This is almost completely transparent for the user, and we hope that these security features will be used by default. A port for Linux on the HP iPAQ allows our software to be used as simply as a typical cordless telephone. A release as open source code is planned for both the *minisip* user agent and the MIKEY library.

Finally, we provide some examples of advanced VoIP scenarios, and some ideas of how to apply MIKEY to these cases. Overall, MIKEY appears flexible enough to handle these specific cases. Hence conferencing and mobility can be conducted in a secure way.

Chapter 17

Future work

Security in VoIP offer numerous possibilities for further development.

Many of the ideas described in this section were suggested by Erik Eliasson <eliasson@it.kth.se> and Professor Gerald Q. Maguire Jr. <maguire@it.kth.se>.

Using minisip as a starting point and applying the security schemes for conferences scenario described in this thesis, a secure conference server could be implemented. Some interesting features would be:

- The use of spatial audio to place the participants in a virtual 3D environment
- The possibility to move from a 2 users point to point session to a multi-user session managed by the conference server, in a smooth and transparent way

The MIKEY library requires some further development, such as the implementation of the re-keying feature and the public key encryption scheme, and the creation of the higher level API.

Once the MIKEY library is released as open source software, it should be maintained. This involves the review and inclusion of third party fixes and additions, along with the creation of support mailing lists for users and developers.

Bibliography

[1] 3rd Generation Partnership Project. *f8 and f9 Specification*. `ftp://ftp.3gpp.org/Specs/latest/Rel-5/35_series/35201-500.zip`, 1999.

[2] ALSA project. *Advanced Linux Sound Architecture*. `http://www.alsa-project.org/`, 2003.

[3] Anton Rager (AVAYA). *IPSec/IKE Protocol Hacking*. `http://ikecrack.sourceforge.net/ike-hacking-toorcon2k2.ppt`, 2002.

[4] J. Arkko, E. Carrara, F. Lindholm, M. Näslund, and K. Norrman. *MIKEY: Multimedia Internet KEYing*, Internet Draft. `http://www.ietf.org/internet-drafts/draft-ietf-msec-mikey-07.txt`, June 2003.

[5] M. Baugher, D. McGrew, D. Oran, R. Blom, E. Carrara, K. Norrman, and M. Näslund. *The Secure Real-time Transport Protocol*, Internet Draft. `http://standards.ericsson.net/fli/draft-ietf-avt-srtp-05.txt`, June 2003.

[6] Charlie Kaufman (IBM), Radia Perlman (Sun Microsystems), Bill Sommerfeld (Sun Microsystems). *DoS Protection for UDP-Based Protocols*, 2003.

[7] David A. McGrew (CISCO). *libsrtp: a library for secure RTP*. `http://srtp.sourceforge.net/srtp.html`, 2002.

[8] M. Degermark and Ed. *Requirements for robust IP/UDP/RTP header compression*, RFC 3096. Internet Engineering Task Force, `ftp://ftp.isi.edu/in-notes/rfc3096.txt`, July 2001.

[9] T. Dierks and C. Allen. *The TLS Protocol Version 1.0*, RFC 2246. Internet Engineering Task Force, `ftp://ftp.isi.edu/in-notes/rfc2246.txt`, January 1999.

[10] Free Software Foundation. *GNU General Public License v2*. http://www.gnu.org/licenses/gpl.html, June 1991.

[11] Audio-Video Transport Working Group, H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications*, RFC 1889. Internet Engineering Task Force, ftp://ftp.isi.edu/in-notes/rfc1889.txt, January 1996.

[12] gtk.org. *GTK + The GIMP Toolkit*. http://www.gnu.org/, 2003.

[13] A. Gulbrandsen, P. Vixie, and L. Esibov. *A DNS RR for specifying the location of services (DNS SRV)*, RFC 2782. Internet Engineering Task Force, ftp://ftp.isi.edu/in-notes/rfc2782.txt, February 2000.

[14] handhelds.org. *GPE: The GPE Palmtop Environment*. http://gpe.handhelds.org/, 2003.

[15] handhelds.org. *Open Palmtop Integrated Environment*. http://opie.handhelds.org/, 2003.

[16] handhelds.org. *the Familiar Project*. http://familiar.handhelds.org/, 2003.

[17] M. Handley and V. Jacobson. *SDP: Session Description Protocol*, RFC 2327. Internet Engineering Task Force, ftp://ftp.isi.edu/in-notes/rfc2327.txt, April 1998.

[18] D. Harkins and D. Carrel. *The Internet Key Exchange (IKE)*, RFC 2409. Internet Engineering Task Force, ftp://ftp.isi.edu/in-notes/rfc2409.txt, November 1998.

[19] Henning Schulzrinne, Elin Wedlund (Columbia University). *Application-Layer Mobility Using SIP*. http://www.cs.columbia.edu/~hgs/papers/Schu0007_Application.pdf, 1999.

[20] R. Housley. *Using AES Counter Mode With IPsec ESP*, Internet Draft. http://www.ietf.org/internet-drafts/draft-ietf-ipsec-ciph-aes-ctr-04.tx%t, May 2003.

[21] Hugo Krawczyk (IBM). *SKEME: A Versatile Secure Key Exchange Mechanism for Internet*. http://www.research.ibm.com/security/skeme.ps, November 1995.

[22] IETF Secretariat. *IP Security Protocol (ipsec)*. http://www.ietf.org/html.charters/ipsec-charter.html, 2003.

[23] International Engineering Consortium. *Voice and Fax over Internet Protocol*. http://www.iec.org/online/tutorials/vfoip/, 2003.

[24] Israel Abad Caballero. Secure Mobile Voice over IP. Master's thesis, Royal Institute of Technology (Stockholm), http://www.it.kth.se/docs/Reports/DEGREE-PROJECT-REPORTS/030626-Israel_%Abad_Caballero-final-report.pdf, June 2003.

[25] J.Arkko, E. Carrara, F. Lindholm, M. Näslund, and K. Norrman. *Key Management Extension for Session Description*, Internet Draft. http://www.ietf.org/internet-drafts/draft-ietf-mmusic-kmgmt-ext-08.txt, August 2003.

[26] Charlie Kaufman. *Internet Key Exchange (IKEv2) Protocol*, Internet Draft. http://www.ietf.org/internet-drafts/draft-ietf-ipsec-ikev2-09.txt, August 2003.

[27] S. Kent and R. Atkinson. *IP Encapsulating Security Payload (ESP)*, RFC 2406. Internet Engineering Task Force, ftp://ftp.isi.edu/in-notes/rfc2406.txt, November 1998.

[28] D. Maughan, M. Schertler, M. Schneider, and J. Turner. *Internet Security Association and Key Management Protocol (ISAKMP)*, RFC 2408. Internet Engineering Task Force, ftp://ftp.isi.edu/in-notes/rfc2408.txt, November 1998.

[29] Microsoft Corporation. *.NET Messenger Service – Free Instant Messaging Service*. http://messenger.msn.com/default.asp?client=0, 2003.

[30] Mika Mustikkamaki. *Wirlab KPhone*. http://www.wirlab.net/kphone/index.html.

[31] D. Mills. *Network Time Protocol (Version 3) Specification, Implementation*, RFC 1305. Internet Engineering Task Force, ftp://ftp.isi.edu/in-notes/rfc1305.txt, March 1992.

[32] National Institute of Standards and Technology. *ADVANCED ENCRYPTION STANDARD (AES)*. http://csrc.nist.gov/publications/fips/fips197/fips-197.ps, November 2001.

[33] National Institute of Standards and Technology. *The Keyed-Hash Message Authentication Code (HMAC)*. http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf, March 2002.

[34] H. Orman. *The OAKLEY Key Determination Protocol*, RFC 2412. Internet Engineering Task Force, `ftp://ftp.isi.edu/in-notes/rfc2412.txt`, November 1998.

[35] Ralf S. Engelschall. *OpenSSL Project*. `http://www.openssl.org/`, 2002.

[36] B. Ramsdell and Ed. *S/MIME Version 3 Message Specification*, RFC 2633. Internet Engineering Task Force, `ftp://ftp.isi.edu/in-notes/rfc2633.txt`, June 1999.

[37] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. *SIP: Session Initiation Protocol*, RFC 3261. Internet Engineering Task Force, `ftp://ftp.isi.edu/in-notes/rfc3261.txt`, June 2002.

[38] SHARP. *Zaurus SDK*. `http://www.zaurus.com/dev/tools/other.htm`, 2003.

[39] Software in the Public Interest. *Debian GNU/Linux – The Universal Operating System*. `http://www.debian.org/`, 2003.

[40] R. Sparks. *The Session Initiation Protocol (SIP) Refer Method*, RFC 3515. Internet Engineering Task Force, `ftp://ftp.isi.edu/in-notes/rfc3515.txt`, April 2003.

[41] The KDE Project. *The K Desktop Environment*. `http://www.kde.org/`, 2003.

[42] Trolltech. *Qt*. `http://www.trolltech.com/products/qt/index.html`, 2003.

[43] University of Cambridge. WiOpt'04: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, 2004.

[44] U.S. National Institute of Standards and Technology. *Secure Hash Standard*. `http://www.itl.nist.gov/fipspubs/fip180-1.htm`, April 1995.

Appendix A

List of Acronyms and Abbreviations

AES	Advanced Encryption Standard
CA	Certificate Authority
CTR	Counter Mode
DH	Diffie-Hellman
DNS	Domain Name System
ESP	Encapsulated Security Payload
FQDN	Fully Qualified Domain Name
GPL	GNU General Public License
HMAC	Keyed-Hashing for Message Authentication
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange
IPSec	IP Security Protocol
ISAKMP	Internet Security Association and Key Management Protocol
IV	Initialization Vector
LAN	Local Area Network
MAC	Message Authentication Code
MIKEY	Multimedia Internet KEYing
MKI	Master Key Identifier
PKE	Public Key Encryption
PKI	Public Key Infrastructure
PSK	Pre-Shared Key
PSTN	Public Switched Telephone Network
RSA	Rivest, Shamir and Adleman
RTCP	Real-time Transport Control Protocol
RTP	Real-time Transport Protocol
SDP	Session Description Protocol

SHA-1	Secure Hash Algorithm 1
SIP	Session Initiation Protocol
S/MIME	Secure/Multipurpose Internet Mail Extensions
SMTP	Simple Mail Transfer Protocol
SRTP	Secure Real-time Transport Protocol
SRTCP	Secure Real-Time Transport Control Protocol
TCP	Transmission Control Protocol
TEK	Traffic Encryption Key
TGK	TEK Generation Key
TLS	Transport Layer Security
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
VoIP	Voice Over Internet Protocol
WLAN	Wireless LAN
XOR	Exclusive OR operation

Appendix B

WiOpt 04 submission

Call establishment delay for secure VoIP

Johan Bilien Erik Eliasson Jon-Olov Vatn
Royal Institute of Technology (KTH)
Stockholm, Sweden

Abstract

In this paper we are studying the possibility to establish a secure VoIP telephone call using SIP. We briefly describe different security services relevant for VoIP and argue that end-to-end authentication and encryption should be provided by default. We also present message flows and novel measurements of the call establishment delay for the draft MIKEY and SRTP protocols using our own SIP user agent (minisip). Our conclusion is that the call establishment delay will not be significantly affected by introducing these security protocols. Therefore we expect that secure SIP user agents will become common even on platforms with moderate performance such as PDAs, which could then be used as secure cordless phones in WLAN environments.

1 SIP call establishment

Figure 1 gives an overview of the SIP[7] call establishment process between two Internet users (Alice and Bob). Since intend to evaluate the delays that Alice and Bob will experience we have divided the process into two phases, *Calling* and *Answering*, with the following delay definitions:

Definition 1 (Calling delay) *The time interval from when the caller (Alice) has dialed the callee (Bob) until she receives the* `180 Ringing` *message* □.

Definition 2 (Answering delay) *The time interval from when the callee (Bob) picks up his phone until he receives the* `ACK` *message.* □.

We believe that these definitions are adequate since Alice expects to hear a ringing tone shortly after dialing Bob. While the phone at Bob is ringing there are no delay requirements, however, as soon as Bob picks up his phone the call establishment should finish rapidly.

Our aim has been to compare the *Calling* and *Answering* delays for a secure and regular (non-secure) SIP call to see if the introduction of security features will have a significant impact on these delays or not. We therefore start by analyzing the call establishment process for a regular SIP call (section 1.1). In section 1.2 we describe the security services and protocols we believe should be added as well as elaborate on alternatives.

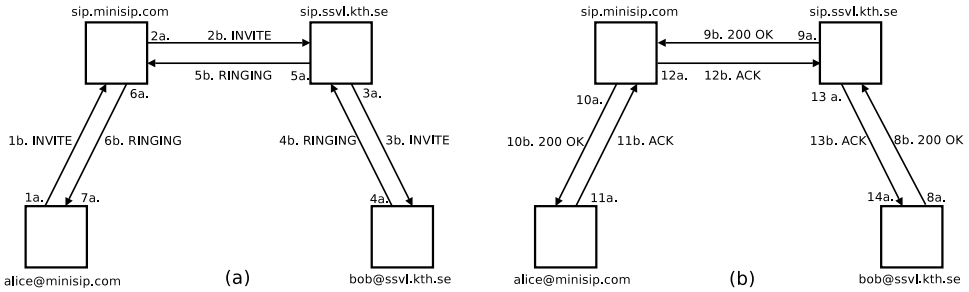

Figure 1: Overview of SIP call establishment: (a) Calling phase, (b) Answering phase

1.1 Regular SIP call establishment

Figure 1 provides a somewhat simplified picture of the signaling message flow required to establish a call using SIP. E.g., we have not shown the `100 Trying` message that each node sends in response to an `INVITE` message, but those messages will not affect the call establishment delay. We have also left out messages related to SIP registration, since we assume that registration has been done in advance.

However, what is important to note is that before a node can forward an `INVITE` message it should do a DNS look-up to find the IP address of its next hop (1a, 2a, and 3a in figure 1a). Such a DNS lookup usually involves two queries: first for a SRV record (to find a domain's SIP servers supporting a specific transport protocol) and then an A record[1] to find its IP address (there can also be an initial NAPTR query to find servers for different transport protocols[6]). The time needed to perform DNS look-ups in steps 1a, 2a and 3a will vary depending on the network delay between the various entities as well as on DNS caching effects. It is also possible that the DNS look-ups in step 1a and 3a may be skipped totally if the DNS resolution performed during SIP registration can be reused.

Another simplification in figure 1 relevant to call setup delay is that it does not show what transport protocol is used. For the regular (non-secure) call establishment we assume UDP transport, but TCP and TLS may be of interest if the SIP messages are large or if their content should be protected.

1.2 Establishing a secure SIP call

Although the previous section presents the necessary steps to establish a VoIP call between two Internet users we strongly believe that security facil-

[1]It is also of interest to query for IPv6 addresses (A6 records).

ities must be provided in order for IP telephony to take off. In particular we argue that end-to-end authentication between Alice and Bob should be possible, and that this initial authentication handshake should result in session keys to protect the voice data stream.

We believe that a user (Alice) may associate the term *secure VoIP call* with properties such as:

1. *That a call is established with the callee she has entered.* Securing the SIP Registration messages will defeat some of the simple redirection attacks. To really ensure Alice that it is Bob's user agent she is communicating with, end-to-end authentication is needed.

2. *That charging is done correctly.* If charging is wanted its correctness is vital, however, we assume that flat rate will be used for Internet calls (fixed monthly cost or free), thus we have not considered this requirement further.

3. *That calls could be blocked efficiently to avoid VoIP spamming.* If VoIP calls will be flat rate one can expect a similar situation with spam phone calls as is currently seen for email. An authentication handshake at call establishment gives the possibility to reject a call automatically based on user preferences.

4. *That the voice data is protected against eavesdropping.* If Alice initiates what is referred to as a secure call she will expect to be able to speak in private with the callee (Bob). The need for this service is probably greater for VoIP than for regular telephony (PSTN) because the possibilities to launch such attacks in an IP environment is larger, in particular since more commodity tools to do this will be available. Session keys to encrypt and integrity protect the (RTP) audio streams can be generated as a side-effect of the authentication handshake.

5. *That information about who Alice is calling (or who is calling Alice) should not be revealed by eavesdropping.* If such security is desired one will have to encrypt the SIP call setup messages in figure 1, e.g., by using TLS transport. An attacker may still be able to guess who Alice is calling by inspecting, e.g., the DNS traffic (if Bob would have his own domain) or the RTP traffic (if Bob has a fixed IP address).

6. *That Alice's identity should not be revealed by eavesdropping.* This requirement is hard to meet and we do not believe that many users find this property crucial. Even if we would be able to protect all SIP signaling from revealing her identity there may be many other ways for a persistent attacker to acquire this information. We are not considering this requirement in this study.

7. *That Alice's identity should be hidden for the callee.* A system should allow a caller to be anonymous. By introducing an initial authentication handshake we will not exclude the possibility to be anonymous, however, we give the callee the possibility to reject such calls.

It is worth noting that the usage of the secure SIP URI (SIPS[7]) does not imply end-to-end authentication or encryption of voice data – it only specifies the protection of the SIP signaling. We believe that users will find this neither satisfactory nor intuitive.

As already mentioned we envision a SIP security model based on end-to-end user authentication and protection of user data. It would probably be possible to use IKE and IPSEC to handle this task, but in this study we are using the MIKEY[1] and SRTP[3] protocols (Internet drafts). SRTP specifies format and ciphers to encrypt and integrity protect RTP data. MIKEY has been designed with the SDP Offer/Answer model in mind and is therefore well-suited to be carried in SIP messages. It provides authentication and generates a master secret to be used by SRTP.

We also recommend that SIP messages are secured hop-by-hop using TLS tunnels between the SIP entities, at least on the path between a user agent and its SIP server. The primary reason is to avoid unnecessary exposing of information about who you are talking to. Most other attacks, such as impersonating Alice or Bob, would be handled by MIKEY/SRTP.

1.3 Multimedia Internet Keying – MIKEY

This section will only describe MIKEY with focus on the aspects that affect the call establishment delay. For more information about MIKEY we refer to work being done within the IETF.

The current MIKEY draft[1] supports three different authentication mechanisms: *shared key*, *public key* and *signed Diffie-Hellman* authentication. The negotiation of MIKEY parameters (including authentication mechanism) follows the SDP Offer/Answer model and is integrity protected by the authentication mechanism offered, i.e., a MAC for the *shared key*, or a digital signature for the other two.

Shared key authentication is probably going to be used at an early deployment stage when people can manually exchange secret keys with their friends. To use secure IP telephony at large scale a PKI should be introduced. A likely scenario is that user agents will store a small set of root CAs, just like web browsers do today. SIP providers will have certificates signed by any of these root CAs, and the providers will issue user certificates to their customers. Of the certificate based authentication mechanisms we only consider *signed Diffie-Hellman*. The reasons for excluding the *public key* mechanism are that we expect it to have a longer delay (Alice will somehow have to retrieve Bob's public key before sending the `INVITE`), and we have

not yet implemented the *public key* mechanism into our user agent (minisip). Furthermore, with signed Diffie-Hellman one gets *perfect forwarding secrecy*.

MIKEY has some additional properties that may affect the call establishment delay:

- To avoid replay attacks MIKEY messages carry a time stamp. This may be a source of annoyance (and would have a major impact on the call establishment delay) if Alice's and Bob's clocks are not reasonably synchronized. The reason for using time stamps instead of *nonce* challenges is probably to make it fit into the Offer/Answer model (nonces would require (at least) a three-way handshake,) however we suggest the usage of *nonces* challenges as a fall-back mechanism. In our measurements we have assumed that the clocks are synchronized.

- When including certificates in the MIKEY messages the SIP messages become too large for UDP transport (SIP messages larger than 1300 bytes must be sent using congestion controlled transport[7]). For Diffie-Hellman we will therefore have to use TCP or TLS, and this will affect the delay of steps 1b, 2b and 3b (see figure 1a), unless those transport sessions are already up.

  It is not only the *size* of the MIKEY packet that affects the choice of transport protocol. As earlier stated one could use TLS protect the SIP messages hop-by-hop between the SIP entities. We suggest the usage of mutual authentication TLS, and that negotiation of this can be done as specified in [2]. This negotiation requires an initial message exchange between the SIP entities (SIP OPTIONS messages). We assume that this has been done in advance (at least between user agent and its SIP proxy), e.g., during SIP registration phase. An alternative would be to use TLS for server authentication, followed by a HTTP Digest challenge[7]. This is fine, although one possible problem with this is that the SIP provider would then have to give a customer **both** a shared key (to authenticate to the SIP provider) and a certificate (to use for MIKEY authentication).

  In the tests presented here we were not able to use TLS transport due to lacking support at the SIP server.

- It is not obvious in which SIP messages Bob should put his MIKEY response message; the `100 Ringing` or the `200 Ok` message? In our tests we have decided to put it in the `200 Ok` message, since the `100 Ringing` message is not transferred reliably (unless reliability is provided by the transport layer). We will discuss this further in section 3.

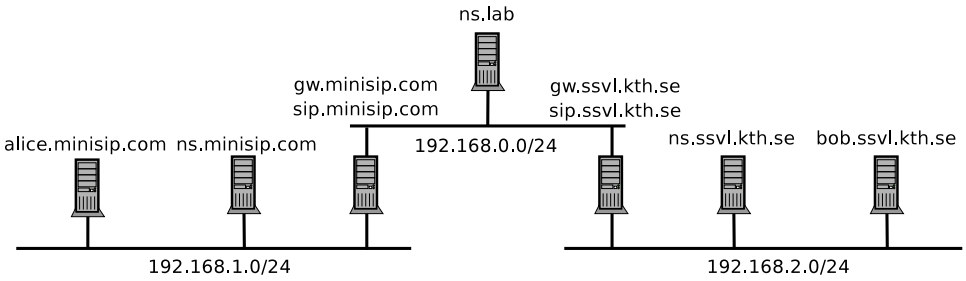

Figure 2: SIP VoIP test bed used during the measurements.

2 Testbed and measurements

Figure 2 shows the simple testbed used in this study. It contains three interconnected networks of which two represents the two domains involved (`minisip.com` and `ssvl.kth.se`), and the last represents the Internet. In each domain there is a name-server (BIND 8.2 on Linux 2.4, 500 MHz Pentium 3 laptops). The root name-server `ns.lab` manages the delegation of `minisip.com` and `ssvl.kth.se` to their respective name server. The two routers (1.1 GHz Celeron desktops) perform static routing, and each router also runs a SIP server, SIP Express Router (SER v0.8.11)[4]).

Alice and Bob use our own Linux SIP user agent, minisip, soon to be released as open source. Alice is a 700 MHz Pentium 3 laptop, running Linux 2.6 (pre-emptive), while Bob is a 1.4 GHz Pentium 4 laptop, running Linux 2.4. For cryptographic operations, minisip uses functions from libcrypto, from the OpenSSL [5] project.

The measurements were done both using network monitoring (tcpdump) and by inserting hooks (saving time-stamps) into the minisip code. The added processing in minisip is less than 5 μs per time-stamp.

2.1 Results

Tables 1 and 2 summarize the measurement results for the *calling delay* and *answering delay* respectively. Figure 3 provides a more detailed view of the Diffie-Hellman tests. The network delays for the SIP messages in the testbed are in-significant, since the distances between the SIP entities are only one hop, i.e., in a real situation network delays will be added to steps 1b, 2b, 3b etc. (see figure 1). It should also be noted that all tests presented in tables 1 and 2 were carried out using UDP transport.

The total *calling delay* is roughly twice as high when using MIKEY with shared key authentication compared to the regular (non-secure) SIP calling delay, and roughly four times as high when using MIKEY with signed

	Delays [ms]							
	1a	2a	3a	4a	5a	6a	7a	Total
No security	4	3	12	11	< 1	< 1	7	48
MIKEY, shared key	28	4	12	19	< 1	< 1	7	90
MIKEY, Diffie-Hellman	110	4	12	19	< 1	< 1	7	175

Table 1: Calling delay for the measured test cases.

	Delays [ms]							
	8a	9a	10a	11a	12a	13a	14a	Total
No security	1	4	< 1	5	3	< 1	-	20
MIKEY, shared key	4	< 1	< 1	9	3	< 1	-	25
MIKEY, Diffie-Hellman	110	< 1	< 1	35	3	3	-	160
		200 (in parallel)						310
					100 (in parallel)			

Table 2: Answering delay for the measured test cases.

Diffie-Hellman. These values may seem dramatic at first, but if we consider the absolute increase (33 ms and 115 ms respectively) the increase becomes in-significant for a human user.

For the answering delay we observed an increase of 7 ms when using pre-shared key, mainly because of the master key decryption. This can be considered as almost negligible for the user, considering network delays. The Diffie-Hellman key agreement increases the *answering delay* more dramatically (around 300 ms), but this should remain acceptable for a human user. Below follows more detailed comments about the different delay components:

- DNS look-ups take relatively little time in our testbed compared to a real network. In our tests DNS look-ups were only performed in step 2a, i.e., when Alice's SIP server had to find the SIP server of `ssvl.kth.se`, and accounted for about 2 ms of step 2a. It should be noted that in these measurements, `ns.minisip.com` already had the SIP server of `ssvl.kth.se` in its cache.

- Step 3a of the calling phase includes a look-up in the location register by the SIP server. It results in a delay of around 10 ms in all cases.

- Generating the initial MIKEY message (1a) is the main contributing factor behind the increase of the *calling delay*. In the case of pre-shared key, the required computations are

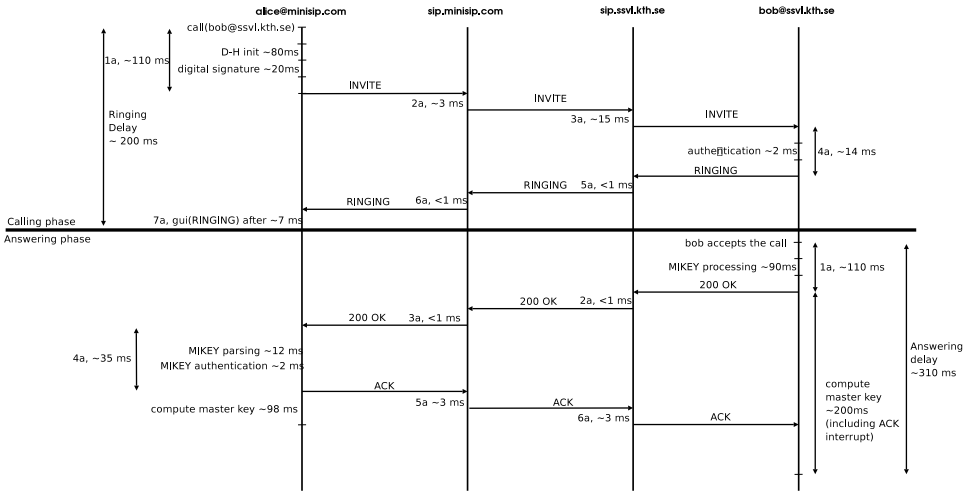

Figure 3: Messages and processing when using MIKEY with Diffie-Hellman and UDP as transport protocol.

  - derivation of transport encryption and authentication keys from the shared key. Those computations involves a pseudo-random function based on HMAC-SHA1.
  - generation of a random master key
  - encryption of this key, using AES in counter mode
  - generation of a MAC using HMAC

  Those cryptographic operations can be performed quickly and results in small delays. When using Diffie-Hellman, the generation of the first MIKEY message includes:

  - The creation of random Diffie-Hellman values (A and $g^A \bmod p$).
  - The computation of a digital signature of the message

  These operations requires higher computational resources, and results in longer delays.

- Authentication of the initiator occurs in step 4a of the *calling delay* (by controlling the MAC for pre-shared key, and controlling the digital signature and included certificates for D-H). This adds a small delay of 8 ms.

- Creation of the MIKEY response message (8a) is very short, around 3 ms, in case of shared keys (MAC computation), but rather long for Diffie-Hellman (which requires the same operations as the creation of the first MIKEY message).

- Authentication of the responder occurs in step 11a. It requires the same type of computation and delays as the authentication of the initiator.

- The Diffie-Hellman key exchange requires Alice and Bob to compute the session key resulting from the Diffie-Hellman values. This operation may be performed after they have sent their last message in the establishment, i.e., starting in parallel with 9a (Bob) and 12a (Alice). In our configuration, the small network delays resulted in Alice receiving the ACK message from Bob before she had the key ready. Therefore the parsing of the incoming message is included in and increased the key computation time. We are currenlty unable to explain the difference in computing time at Bob (200 ms) and Alice (100 ms).

3 Conclusions

As seen in our results, the additional processing time required by security operations can be considered as acceptable for a human user.

It can be assumed that the hardware we used have computational resources of the same order as the one available on the short term coming PDAs, so that secure VoIP call setups could be performed on handheld devices.

In most cases, network latencies will be the real limiting factor. Packet loss, quite common on wireless LANs, creates additional and usually larger delays. The use of MIKEY does not require more network transmissions.

Problems may raise however when an automate is involved, such as an answering machine or a conference server. Having this machine being able to process only 5 calls per second, due to a processing time of 200 ms in the *answering phase* (in case of Diffie-Hellman) may be a serious limitation. In this situation, pre-shared key should be preferred.

The Diffie-Hellman key agreement being time demanding, a pre-initialization of its parameters would greatly improved the call establishment delay. This is possible:

- For Alice, if the Diffie-Hellman group is chosen in advance. In our configuration, this would shorten the *calling delay* of about 80 ms.

- For Bob, if it is assumed that the group chosen by Alice is likely to be the fifth OAKLEY group (the only one mandatory to implement in MIKEY). This would reduce the delay of the MIKEY response computation, and thus the *answering delay*, of about 80 ms.

Note that the final generation of the master key may not be precomputed, since it is the result of the key agreement.

In our implementation, we chose to place the MIKEY response in the `200 Ok` SIP message. An alternative would be to transmit it in the `100 Ringing`, thus allowing both parties to compute the master key before Bob picks up the phone. But this solution has some serious drawbacks. A `1XX` response is not retransmitted in case of packet loss (on UDP), and thus should not be used to carry necessary information. Also, processing the key agreement before the call was accepted by the user could lead to denial of services.

The current lack of SIP servers supporting TLS prevented us from conducting measurements with a protected transport layer. According to [7], the TLS connection between the user agent and the proxy should be kept alive after the registration. Therefore, we can assume that using TLS would not result in significant additional delays. If the connection were to be shut down for some reason, re-establishing it, using TLS session resumption, would take around 6 ms (compared to 60 ms if fully established, those delays being measured in our configuration between minisip and stunnel [8]).

References

[1] Jari Arkko, Elisabetta Carrara, Fredrik Lindholm, Mats Naslund, and Karl Norrman. MIKEY: Multimedia Internet KEYing. IETF draft <draft-ietf-msec-mikey-07.txt>, June 2003. Work in progress.

[2] Jari Arkko, Vesa Torvinen, Gonzalo Camarillo, Aki Niemi, and Tao Haukka. IETF RFC 3329 Security Mechanism Agreement for the Session Initiation Protocol (SIP), jan 2003.

[3] Mark Baugher, David McGrew, Elisabetta Carrara, Mats Naslund, and Karl Norrman. The Secure Real-time Transport Protocol. IETF draft <draft-ietf-avt-srtp-09.txt>, July 2003. Work in progress.

[4] SIP Express Router. http://www.iptel.org/ser/ Last visited november 2003.

[5] OpenSSL Project. http://www.openssl.org/ Last visited november 2003.

[6] J. Rosenberg and H. Schulzrinne. Session initiation protocol (SIP): locating SIP servers. RFC 3263, Internet Engineering Task Force, June 2002.

[7] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: session initiation protocol. RFC 3261, Internet Engineering Task Force, June 2002.

[8] Stunnel – Universal SSL Wrapper. http://www.stunnel.org/ Last visited november 2003.

IMIT/LCN 2003-14

www.kth.se