Security Analysis of Voice-over-IP Protocols

Prateek Gupta and Vitaly Shmatikov
The University of Texas at Austin

Abstract

Transmission of voice communications as datagram packets over IP networks, commonly known as Voice-over-IP (VoIP) telephony, is rapidly gaining wide acceptance. With private phone conversations being conducted on insecure public networks, security of VoIP communications is increasingly important. We present a structured security analysis of the VoIP protocol stack, which consists of signaling (SIP), session description (SDP), key establishment (SDES, MIKEY, and ZRTP) and secure media transport (SRTP) protocols. Using a combination of manual and tool-supported formal analysis, we uncover several design flaws and attacks, most of which are caused by subtle inconsistencies between the assumptions that protocols at different layers of the VoIP stack make about each other.

The most serious attack is a replay attack on SDES, which causes SRTP to repeat the keystream used for media encryption, thus completely breaking transport-layer security. We also demonstrate a man-in-the-middle attack on ZRTP which disables authentication and allows the attacker to impersonate a ZRTP user and establish a shared key with another user. Finally, we show that the key derivation process used in MIKEY cannot be used to prove security of the derived key in the standard cryptographic model for secure key exchange.

1 Introduction

Achieving end-to-end security in a voice-over-IP (VoIP) session is a challenging task. VoIP session establishment involves a jumble of different protocols, all of which must inter-operate correctly and securely. The VoIP protocol stack is shown in figure 1. For the purposes of our analysis, we will divide it into four layers: *signaling*, *session description*, *key exchange* and *secure media (data) transport*.

Signaling is an application-layer (from the viewpoint of the underlying communication network) control mechanism used for creating, modifying and terminating VoIP sessions with one or more participants. Signaling protocols include Session Initiation Protocol (SIP) [23], H.323 and MGCP. Session description protocols are used for describing multimedia and other sessions for the purposes of session announcement, session invitation and other forms of multimedia session initiation. SDP [17] is an example of a session description protocol. Key exchange protocols are intended to provide a cryptographically secure way of establishing secret session keys between two or more participants in an untrusted environment. These session keys are then used to set up cryptographically secure data channels.

From the security perspective, key exchange is the fundamental building block in secure session establishment. Security of the media transport layer — the layer in which actual voice datagrams are transmitted — depends on the *secrecy* of session keys and *authentication* of session participants. Since the established key is typically used in a symmetric encryption scheme, key secrecy requires that nobody other than the legitimate session participants be able to distinguish it from a random bitstring. Authentication requires that, after the key exchange protocol successfully completes, the participants' respective views of

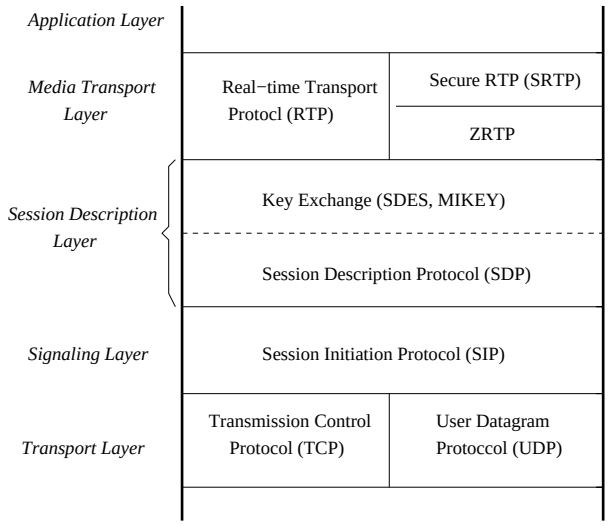

Figure 1: Voice-over-IP protocol stack

sent and received messages must *match* (*e.g.*, see the notion of "matching conversations" in [6]). Examples of key exchange protocols for VoIP sessions include SDP's Security DEscriptions for Media Streams (SDES) [1], Multimedia Internet KEYing (MIKEY) [2] and ZRTP [27].

Secure media transport aims to provide confidentiality, message authentication and integrity, and replay protection to the media (data) stream. In the case of VoIP, this stream typically carries voice datagrams. Confidentiality means that the data under encryption is indistinguishable from random for anyone who does not have the key. Message authentication implies that if Alice receives a datagram apparently sent by Bob, then it was indeed sent by Bob. Data integrity implies that any modification of the data in transit will be detected by the recipient. An example of a secure media transport protocol is Secure Real-time Transport Protocol (SRTP) [5], which is a profile of Real-time Transport Protocol (RTP) [24].

Our contributions. We analyze security of VoIP protocols at all layers of the VoIP stack. In particular, we focus at inter-operation between protocols at different layers. A protocol may be secure when executed in isolation, but the composition of protocols in different layers may be insecure. Moreover, a protocol may make assumptions about another protocol that the latter does not satisfy.

- We show how to cause SRTP protocol to repeat the keystream used for datagram encryption. This enables the attacker to obtain the `xor` of plaintext datagrams or even completely decrypt them. The SRTP keystream is generated by using AES in a stream cipher-like mode. The AES key is generated by applying a pseudo-random function (PRF) to the session key. SRTP, however, does not add any session-specific randomness to the PRF seed. Instead, SRTP assumes that the key exchange protocol, executed as part of RTP session establishment, will ensure that session keys never repeat. Unfortunately, S/MIME-protected SDES, which is one of the key exchange protocols that may be executed prior to SRTP, does not provide any replay protection. As we show, a network-based attacker can replay an old SDES key establishment message, which will cause SRTP to repeat a keystream that it used before, with devastating consequences.

- We show an attack on the ZRTP key exchange protocol that allows the attacker to disable authentica-

tion and thus completely break security of key establishment. ZID values, which are used by ZRTP participants to retrieve previously established shared secrets, are *not* authenticated as part of ZRTP. Therefore, an attacker can initiate a session with some party A under the guise of another party B, with whom A previously established a shared secret. As part of session establishment, A is supposed to verify that B knows their shared secret. The attacker deliberately chooses values that cause verification to fail, in which case A decides — following ZRTP specification — that B has "forgotten" the shared secret and proceeds to execute key exchange without authentication. As a result, the attacker shares a key with A, who thinks she shares this key with B. Our analysis of ZRTP is supported by the AVISPA formal analysis tool [3].

- We show several minor weaknesses and potential vulnerabilities to denial of service in other protocols. We also observe that the key derived as the result of MIKEY key exchange cannot be used in a standard cryptographic proof of key exchange security (*e.g.*, [10]). Key secrecy requires that the key be indistinguishable from a random bitstring. In MIKEY, however, the joint Diffie-Hellman value derived as the result of the protocol is used directly as the key. Membership in many Diffie-Hellman groups is easily checkable, thus this value *can* be distinguished from a random bitstring. (This observation does not immediately lead to any attacks.)

The rest of the paper is organized as follows. In section 2, we describe the protocols, focusing on SIP (signaling), SDES, ZRTP and MIKEY (key exchange), and SRTP (transport). In section 3, we describe the attacks and vulnerabilities that we discovered. Related work is in section 4, conclusions are in section 5.

2 Protocols

2.1 Signaling: SIP

Session Initiation Protocol (SIP) [23] is an application-layer signaling protocol used for creating, modifying and terminating sessions with one or more participants. A SIP network consists of the following entities: *end points*, a *proxy* and/or *redirect server*, *location server*, and a *registrar*. End points or *User Agents* represent phone devices or software modems. SIP users are not bound to specific devices; they register themselves with the registrar and use a special form of address resolution to identify other users. SIP user identification is based on a special type of Uniform Resource Identifier (URI) called SIP URI, similar to email addresses. A location server stores the address bindings of users when they register themselves with the registrar.

SIP servers can operate in a *proxy mode* or *redirect mode*. In proxy mode, the server intercepts messages from the end points, inspects their `To:` field, contacts the location server to resolve the username into an address and forwards the message to the appropriate end point or another server. In redirect mode, the only difference is that instead of forwarding the packet along the actual route, the redirect server returns the address to the end points and the onus of transmitting the packets is placed on the end points.

SIP uses a HTTP-like request-response mechanism for initiating a two-way communication session. The protocol itself is modeled on the three-way TCP handshake. Figure 2 shows a SIP connection setup with an intermediate proxy server between end points. In order to set up a connection between Alice's and Bob's UAs, Alice's SIP URI is first resolved into the IP address of the UA under which Alice is currently registered. SIP address resolution and routing is usually not done by the UA itself, but rather delegated to the proxy server for the UA's domain. In our example, Bob's proxy will make a DNS lookup to determine the address of Alice's proxy server. During the setup process, communication details are negotiated between UAs using the Session Description Protocol (SDP), described in section 2.2.

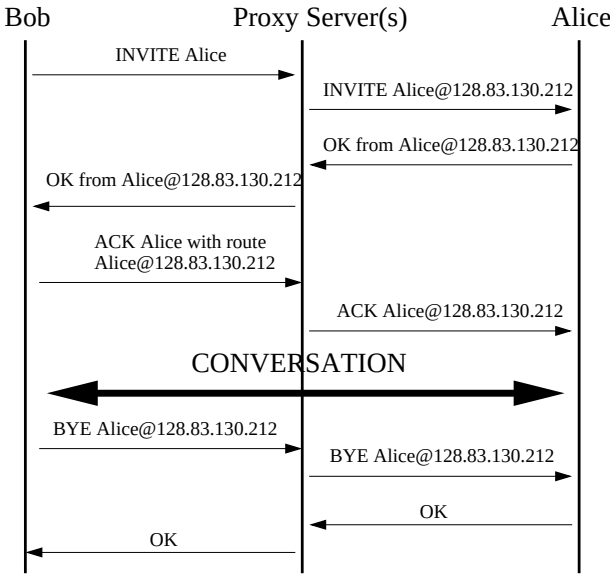

Figure 2: SIP protocol exchange

To place a call to Alice, Bob's UA sends an INVITE request to the proxy server containing SDP info, which is then forwarded to Alice's UA, possibly via her proxy server (after address resolution by Bob's proxy). If Alice wants to talk to Bob, she sends an OK message back to Bob containing her SDP preferences. Bob then responds with an ACK. Media exchange takes place directly between Alice's and Bob's respective UAs. From the network security point of view, this implies that both hops must be secured on a hop-by-hop basis, and the direct path must be secured as well.

SIP messages can be transported over a TCP stream, provided the packet size is smaller than the Maximum Transmission Unit (MTU), or embedded into UDP datagram packets. Therefore, security mechanisms used to encrypt and authenticate multimedia streams must support UDP as a transport layer protocol. This requirement excludes several popular security mechanisms such as the TCP-based Transport Layer Security (TLS) [12], which also requires a Public Key Infrastructure (PKI). SIP also presents challenges for firewalls and Network Address Translators (NATs), but those are outside the scope of this paper.

2.2 Session description: SDP

Session Description Protocol (SDP) is a format for describing multimedia session parameters for the purpose of session announcement, session invitation, and so on. We omit the details, which can be found in [17]. A multimedia *session* is a set of multimedia senders and receivers and the data streams flowing between them. *Session announcement* is a mechanism by which a session description is conveyed to users proactively, before they request it. SDP carries the following information: media type (audio/video), transport protocol, media format (MPEG, *etc.*), transport port and unicast/multicast address for media. A single session may consist of multiple media streams. A session announcement consists of a session level description (details that apply to all media streams) and, optionally, several media-level descriptions. Because SDP is purely a format specification, it is independent of the transport layer and may be carried, for example, by SIP.

2.3 Key exchange

SDES. SDES [1] defines a key transport extension of the Session Description Protocol for unicast media streams. This provides a way to signal and negotiate cryptographic key(s) and other session parameters for media streams in general, and for SRTP in particular. The attribute called "crypto" is limited to two-party unicast media streams where each source has a unique cryptographic key. The crypto attribute for SRTP is defined as: $\texttt{a} = \texttt{crypto} : \langle \texttt{tag} \rangle \langle \texttt{crypto} - \texttt{suite} \rangle \langle \texttt{key} - \texttt{params} \rangle [\langle \texttt{session} - \texttt{params} \rangle]$, where tag is a decimal number used as an attribute identifier; $\texttt{crypto} - \texttt{suite}$ defines the encryption and authentication algorithms to be used in SRTP. The $\texttt{key} - \texttt{params}$ attribute specifies one or more cryptographic keys for the crypto-suite as $\langle \texttt{key} - \texttt{method} \rangle : \langle \texttt{key} - \texttt{info} \rangle$. The only method supported for key exchange is inline :, where the key itself must be included in plaintext. $\texttt{session} - \texttt{params}$ (optional) are specific to the transport used for SDP. Since the key is included directly in the SDP attachment of a SIP message, SIP must ensure that the message is protected at the transport layer.

SIP security mechanisms are described in detail in appendix A. For our purposes, it is enough to observe that transport-layer protection in SIP can be done using either TLS [12] (if the transport layer is TCP), or S/MIME [21]. Use of TLS is deprecated because TLS provides does not provide end-to-end protection over a chain of proxies. Moreover, it assumes that the next hop in the SIP proxy chain is trusted. S/MIME, by contrast, provides end-to-end confidentiality and authentication for SDP payload encoded as MIME [15].

Note that S/MIME does *not* provide any replay protection. Hence, if S/MIME is used to protect SDP payload, then the application must provide a separate defense against replay attacks. In general, most applications have limited replay protection because it requires state maintenance and/or loose clock synchronization. In section 3.2, we will show how the attacker can exploit the lack of replay protection in S/MIME-protected SDES to completely break security of an SRTP session.

ZRTP. ZRTP [27] describes an extension header for Real-time Transport Protocol (RTP) to establish a session key for SRTP sessions using Diffie-Hellman key exchange. An implementation of ZRTP is available as Zfone [26]. The main distinguishing feature of ZRTP, as opposed to other key exchange protocols, is that it does not require prior shared secrets or the existence of a separate PKI infrastructure. This is an important consideration since it eliminates the need for a trusted certificate server. The RFC states that the protocol is resilient to man-in-the-middle attacks and provides confidentiality, and in the cases where a secret is available from the signaling protocol, authentication. Since Diffie-Hellman (DH) key exchange is malleable and does not provide protection against man-in-the-middle attacks, ZRTP uses a *Short Authentication String* (SAS), which is essentially a cryptographic hash of two Diffie-Hellman values, for authentication. Instead of PKI, users rely on cached Diffie-Hellman secrets for authentication in a series of key exchange sessions.

Figure 3 shows a ZRTP key exchange between users Alice and Bob. To start a ZRTP key exchange, an end point sends a ZRTP HELLO message to the other end point. The HELLO message contains SRTP configuration options and a unique ZID, which is generated once at installation time. This ZID will be used by the recipient to retrieve cached shared secrets. On receiving a HELLO message, the other end point determines the algorithms to be used for the ZRTP exchange and replies with a HELLOACK message (if the protocol is supported). After both parties have exchanged HELLO and HELLOACK messages, key exchange begins with a COMMIT message. The HELLO and HELLOACK messages are optional and an end point can directly initiate a ZRTP session by sending a COMMIT message. The sender of the COMMIT message (Bob in our example) is called the *initiator*, Alice is the *responder*.

We describe the Diffie-Hellman exchange in some detail, focusing only on the relevant message fields and omitting the rest. Bob acts as the initiator by sending a COMMIT message. hash, cipher and pkt describe the hash, encryption and public key algorithms, respectively, chosen by Bob from the intersection

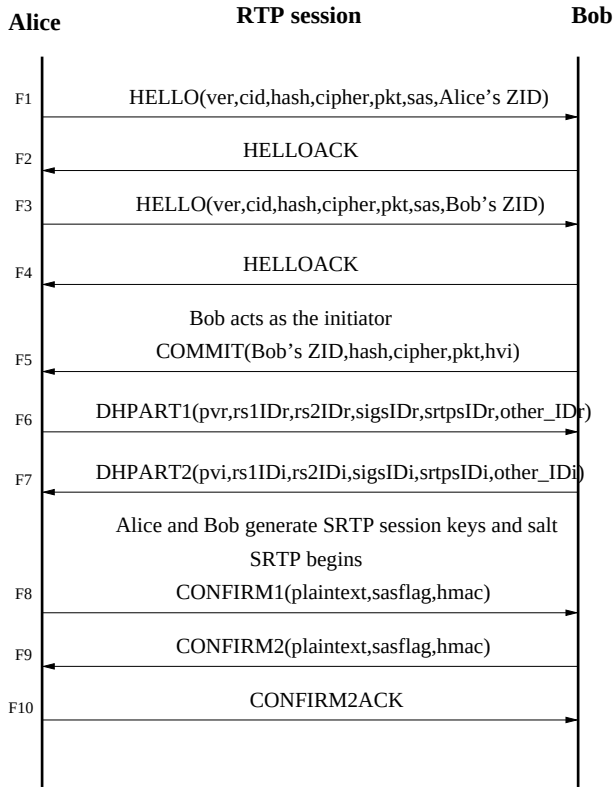

Figure 3: Establishment of SRTP session key using ZRTP

of algorithms in the sent and received HELLO messages. Bob chooses a random exponent `svi` and computes the value $\texttt{pvi} = \texttt{g}^{\texttt{svi}} \bmod \texttt{p}$, where `g` (generator of the Diffie-Hellman group G) and `p` are determined by the `pkt` value. `hvi`, called the hash commitment, is the hash of the Diffie-Hellman value generated by Bob concatenated with `hash`, `cipher`, `pkt` and `sas` from Alice's HELLO message.

Upon receipt of the COMMIT message, responder Alice generates her own Diffie-Hellman secret `svr` and computes the corresponding public value `pvr`. The final shared secret is computed as the hash of the Diffie-Hellman shared secret (DHSS) together with any other secrets shared between Alice and Bob, sorted by Bob's shared secret IDs. For each shared secret, its ID is `HMAC` of the string "`Responder`" computed using this secret as the key. Alice uses Bob's ZID to retrieve the two shared secret values, `rs1` and `rs2`, and any other possible secrets. Bob's behavior in response to the DHPART1 message is similar.

Upon receipt of the DHPART2 message, responder Alice checks that Bob's public DH value is not equal to `1` or $\texttt{p} - \texttt{1}$. RFC states that this check thwarts man-in-the-middle attacks. In section 3, however, we will describe how an attacker can successfully launch a man-in-the-middle attack against the protocol without the participants ever detecting the attack. If the check succeeds, Alice computes the hash of the received value and checks whether it matches `hvi` received in the COMMIT message. If not, Alice terminates the protocol. Otherwise, she stores the shared secret IDs received from the DHPART2 message as set `A`.

Alice then computes the set of shared secret IDs that she *expects* to receive from Bob. For each secret, its ID is computed as `HMAC` of the string "`Initiator`", keyed with the secret itself. Let `B` be the set of these expected IDs. Alice then computes the intersection of sets `A` and `B`. Secrets corresponding to IDs

in the intersection are stored as set D, sorted in ascending order. The final session key is computed as the hash of the joint Diffie-Hellman secret concatenated with the set of secrets in set D. Finally, cached shared secrets rs1 and rs2 are updated as $\texttt{rs2} = \texttt{rs1}$ and $\texttt{rs1} = \texttt{HMAC}(\texttt{session key}, \text{“known plaintext”})$ on both sides. The master key and the salt for the SRTP session are computed as HMAC of known plaintexts using the new session key. CONFIRM message also sends an HMAC of a known plaintext under the session key. The sasflag specifies whether the end points supports verification of SAS.

Multimedia Internet KEYing. MIKEY [2] is another key exchange protocol for SRTP. It can operate in three different modes: pre-shared key with key transport, public key with key transport, public key with key exchange using authenticated Diffie-Hellman (DH). An advantage of MIKEY is that the key can be negotiated using MIKEY as part of the SDP payload during the session setup phase in SIP. Thus, it requires no extra communication overhead.

An obvious disadvantage of MIKEY is that it requires the existence of either prior shared secrets or a separate PKI infrastructure, with all attendant problems such as certificate dispersal, revocation, and so on. In the pre-shared key mode, the key is generated completely by the session initiator; session responder does not participate in key derivation at all. A later extension [13] provides for a DH exchange in the pre-shared key mode.

Before describing the three modes of MIKEY, we need some notation. *Data security protocol* is the security protocol, such as SRTP, used to protect the media session. *Data security association* (Data SA) comprises the session key (TEK) and a set of parameters. *Crypto session* (CS) is a uni- or bi-directional media stream. A crypto session is protected by a unique instance of a data security protocol. Each crypto session has a unique identifier known as the CS ID. *Crypto session bundle* (CSB) is a set of crypto sessions which derive their session keys (TEKs) from a common TGK and a set of security parameters. CSB ID is a unique identifier for the crypto session bundle. *Traffic Generating Key* (TGK) is a bitstring agreed upon by two or more parties associated with a CSB. One or more TEKs can be derived from the TGK and the unique crypto session ID.

- *Pre-Shared Key Transfer*. In this mode, the key is generated by the initiator and transferred to the responder. The message is integrity-protected using a keyed MAC and encrypted. The respective keys are derived from the shared secret s and a random value using a cryptographically secure hash function. Let $\texttt{ID}_\texttt{i}$ and $\texttt{ID}_\texttt{r}$ be the identities of the initiator and responder, respectively. Message m is defined as $\texttt{m} := \texttt{HDR}, \texttt{T}, \texttt{RAND}, [\texttt{ID}_\texttt{i}], [\texttt{ID}_\texttt{r}], \{\texttt{SP}\}, \texttt{KEMAC}$, where T is the timestamp (used for replay protection), RAND is a random number used for generating encryption key ($\texttt{Encr}_\texttt{k}$) and authentication key ($\texttt{Auth}_\texttt{k}$) from the shared secret s, SP is a set of security policies and $\texttt{KEMAC} = \texttt{E}(\texttt{Encr}_\texttt{k}, \{\texttt{TGK}\})||\texttt{MAC}$. $\texttt{E}(\texttt{key}, \texttt{text})$ denotes the encryption of text with the encryption key key and || denotes string concatenation. MAC is a keyed message authentication code computed over the entire message m using the authentication key $\texttt{Auth}_\texttt{k}$. It is assumed that TGK is a chosen uniformly at random by the initiator. For mutual authentication, the initiator may request the responder to send a verification message which includes the message header HDR, timestamp T, the initiator and responder identities $\texttt{ID}_\texttt{i}$, $\texttt{ID}_\texttt{r}$, respectively, and a MAC.

- *Public Key Transfer*. As in the pre-shared key mode, the initiator's message transfers or more TGKs and set of media session security parameters the responder. The initiator's message is

$$\texttt{m} := \texttt{HDR}, \texttt{T}, \texttt{RAND}, [\texttt{ID}_\texttt{i}|\texttt{CERT}_\texttt{i}], [\texttt{ID}_\texttt{r}], \{\texttt{SP}\}, \texttt{KEMAC}, \texttt{PKE}, \texttt{SIGN}_\texttt{i}$$

  Here $\texttt{CERT}_\texttt{i}$ stands for the initiator's certificate. In this mode, the encryption and authentication keys are derived from an *envelope key* ($\texttt{Env}_\texttt{k}$) chosen by the initiator at random. PKE is the encryption of $\texttt{Env}_\texttt{k}$

under the responder's public key. Note that this requires prior knowledge of the responder's (properly certified) public key. $\mathtt{SIGN_i}$ is a signature over the entire message $\mathtt{m}$ using the initiator's private signing key. As in the pre-shared key mode, the initiator may request a verification message from the responder.

- *Public Key with Diffie-Hellman Exchange*. Let G denote a large cyclic multiplicative group with generator $\mathtt{g}$. Authenticated Diffie-Hellman key exchange is shown below:

  $\mathtt{Init} \rightarrow \mathtt{Resp}: \quad \mathtt{HDR}, \mathtt{T}, \mathtt{RAND}, [\mathtt{ID_i}|\mathtt{CERT_i}], \mathtt{SP}, \mathtt{DH_i}, \mathtt{SIGN_i}$
  $\mathtt{Init} \leftarrow \mathtt{Resp}: \quad \mathtt{HDR}, \mathtt{T}, [\mathtt{ID_r}|\mathtt{CERT_r}], \mathtt{ID_i}, \mathtt{DH_r}, \mathtt{DH_i}, \mathtt{SIGN_r}$

  Here $\mathtt{DH_i}, \mathtt{DH_r}$ stand for $\mathtt{g}^{\mathtt{x_i}}$ and $\mathtt{g}^{\mathtt{x_r}}$, where $\mathtt{x_i}, \mathtt{x_r}$ are randomly chosen by the initiator and responder, respectively. The derived key is $\mathtt{g}^{\mathtt{x_i}\cdot\mathtt{x_r}}$. DH group parameters are chosen by the initiator and signaled to the responder.

2.4 Secure media transport: SRTP

SRTP [5] defines a profile of the Real-time Transport Protocol (RTP) which aims to provide confidentiality, message authentication, and replay protection to RTP data and control traffic. SRTP uses a single *master key* to derive keying material via a cryptographically secure hash function. First, some notation. A *master key identifier* is a tag used by the key management protocol to identify the master key from which session keys are derived for the SRTP session. An *authentication tag* is a message authentication code computed over the RTP header and the encrypted portion of the RTP payload using the authentication key. A *cryptographic context* refers to the cryptographic state information maintained by the sender and receiver for the media stream. This includes the master key, session keys, identifiers for encryption and message authentication algorithms, lifetime of session keys, and a rollover counter (ROC).

Each RTP packet consists of a 16-bit sequence number (SEQ) which is monotonically increasing. The rollover counter is maintained by the receiver and is incremented by 1 every time the sequence number wraps around. For a multicast stream with multiple senders, a synchronization source identifier (SSRC) uniquely identifies a sender within a session. The only requirement on SSRC is that the SSRC must be unique for every sender within a session. A cryptographic context for SRTP is identified uniquely by the triple (SSRC, destination network address, destination port).

For data encryption, SRTP uses a single cipher, Advanced Encryption Standard (AES), in one of the following two modes: (i) Segmented Integer Counter mode, or (ii) f-8 mode. The input to AES is the triple (key,SSRC,SEQ), where "key" is the encryption key (explained below), SSRC is synchronization source identifier and SEQ is the sequence number of the packet. Instead of using AES as a block cipher, SRTP uses it as if it were a stream cipher and encrypts datagrams by `xor`'ing them with the output of AES applied to (key,SSRC,SEQ).

SRTP key derivation. SRTP uses a cryptographically secure pseudo-random function (PRF) for generating encryption and authentication session keys from the master key, master salt and the packet sequence number. The sequence number of the packet is chosen by the sender. Both master key and master salt are derived *deterministically* by applying HMAC, keyed with the material received during the key exchange protocol, to a known plaintext (as defined by the key exchange protocol). Session key derivation is defined in terms of a $\mathtt{label}$ (8-bit constant), master salt, $\mathtt{m_s}$ and key derivation rate, as determined in the cryptographic context and the index (48-bit ROC||SEQ). Let $||$ denote string concatenation. Let $\mathtt{x} = (\langle\mathtt{label}\rangle||\mathtt{r})\ \mathtt{xor}\ \mathtt{m_s}$, where $\mathtt{r}$ is the integer quotient obtained by dividing index by the key derivation rate. Let $\mathtt{m_k}$ denote the master key and $\mathtt{PRF}(\mathtt{k}, \mathtt{x})$ denote a pseudorandom function family such that for the secret random key $\mathtt{k}$, given m-bit $\mathtt{x}$, the output is an n-bit string computationally indistinguishable from random n-bit string. The session keys

are generated as $\mathtt{PRF}(\mathtt{m_k}, \mathtt{x})$ by substituting different labels for encryption, authentication and salting keys, respectively.

An important point to note is that there is *no receiver-generated randomness* in the session key derivation process. This is a potential weakness because security of the stream cipher-like encryption used in SRTP depends critically on the keystream never repeating.

For the keystream never to repeat, the PRF output (used as input into AES) must never repeat, and this means that the *PRF input must be unique for every session*. Uniqueness of SSRC and SEQ is not guaranteed. Moreover, both values are public and can be eavesdropped by the attacker. Therefore, the master key and master salt combination must be unique for each session. Both are derived deterministically from the key material received during the key exchange protocol. If the attacker ever succeeds in tricking an SRTP session into re-using previously used key material, the keystream will repeat. In section 3.2, we will describe how the attacker can force a VoIP server implementing SRTP in conjunction with SDES key exchange to repeat the keystream, completely breaking encryption of the data stream.

3 Attacks and Vulnerabilities

3.1 Attacks on SIP

Denial of service. A denial of service attack focuses on rendering a network of service unavailable, usually by directing a high volume of traffic towards the service thereby denying it to legitimate clients. A distributed denial of service allows a single network user to cause multiple network hosts to flood the target host. SIP architecture makes it particularly easy to launch a distributed denial of service attack. Attackers can launch bogus requests with a spoofed source IP address of the identified victim and send it to large number of SIP elements (or proxies), thereby causing unknowing SIP UAs and proxies to generate denial of service traffic aimed at the target. Similarly, attackers can use spoofed `Route header` fields in a request that identify the target host and send such messages to forking proxies who will amplify messages sent to the target. A wide variety of denial of service attacks become possible if REGISTER requests are not properly authenticated and authorized by registrars. If a malicious user is able to de-register some or all other users in the network and register his own device on their behalf then he can easily deny access to any of those users/services. Attackers can also try to deplete storage resources of the registrar by creating a huge number of bindings.

Authentication. Authentication is particularly difficult to achieve in SIP, since there are a number of intermediate elements such as proxies which possibly modify the contents of a message before it reaches the desired destination. All such intermediate elements must be trusted. Also, registration requests must be authenticated by the registrar, or else they open the door to many attacks, including denial of service.

SIP registration does not require the `From` field of a message to be the same as `To` header field of the request, allowing third parties to change address-of-record bindings on behalf of another user. If the attacker can successfully impersonate a party authorized to change contacts on behalf of a user, he can arbitrarily modify the address-of-record bindings for the associated `To` address. Since SIP authentication relies implicitly on authenticity of the server and intermediate proxies, the attacker who is able to successfully impersonate a server or a proxy can do arbitrary damage including denying service to the client or launching a (distributed) denial of service attack. This requires the existence of some methodology for the client to authenticate the server and/or the proxy. Unfortunately, no such mechanism is specified in the SIP RFC.

Another important security vulnerability in SIP is that BYE requests to terminate sessions are not authenticated since they are not acknowledged. Instead, a BYE request is implicitly authenticated if it is received from the same network element (on the same path) as a previous INVITE. A third-party attacker

can thus observe the parameters of an eavesdropped INVITE message, and then insert a BYE request into the session. Once the BYE request is received by the target, the session would be torn down permanently. Similar attacks can be launched on re-INVITE messages used to change session parameters.

3.2 Attack on SDES/SRTP

Figure 4 shows an attack on SRTP when used in combination with SDES key exchange. Suppose two legitimate users, Alice and Bob, previously carried out a successful VoIP session, which the attacker was able to passively eavesdrop, without learning the session key and thus not being able to decrypt the data streams. Suppose Bob was the initiator in this session, and SDES was used to transport SRTP key material. To provide confidentiality for the SDES message, S/MIME was used to encrypt the payload. S/MIME, in general, is preferred over TLS for protecting SDP messages because (i) S/MIME provides end-to-end integrity and confidentiality protection, and (ii) S/MIME does not require the intermediate proxies to be trusted.

S/MIME does not provide any anti-replay protection. After the original session has been torn down, the attacker can replay Bob's original INVITE message to Alice, containing an S/MIME-encrypted SDP attachment with the SDES key transfer message. Since Alice does not maintain any state for SDP, she will not be able to detect the replay. Using the old session's key material as her HMAC key, she will derive exactly the same master key and master salt as in the original session. Since SSRC and sequence number are the same, the resulting session encryption key will be the same as in the previous session, and the keystream generated by applying AES to the (key,SSRC,SEQ) triple will be the same as in the original session.

Encryption in SRTP is simply the `xor` of the data stream with the keystream. If Alice now sends a datagram in the new session that she thinks she is establishing with Bob, the attacker can `xor` the encrypted data stream with the data stream he eavesdropped in the original session. The keystream will cancel out, and the result will be the `xor` of two data streams. If data streams contain enough redundancy or the attacker can guess parts of either stream, he will be able to completely or partially reconstruct the data of both streams. In any case, encryption has been completely removed. This is similar to the famous attack on 802.11b WEP [9], where the keystream was re-used due to exhaustion of initialization vectors for the stream cipher.

The most important observation underlying our attack is that SRTP does not use any randomness on the responder side when the responder derives session keys. Instead, SRTP completely relies on the key exchange protocol to ensure that the key material is fresh for every session. Some key exchange protocols, such as MIKEY, do provide anti-replay defenses to prevent re-use of old key transfer messages. Unfortunately, SRTP is designed to be used with a wide variety of key exchange protocols, including S/MIME-encrypted SDES, which does *not* ensure key material freshness, leading to a devastating attack.

To prevent keystream re-use, SRTP responder should use its own fresh randomness as part of the key derivation process, *e.g.*, as input to HMAC used in session key derivation. This randomness need not be secret. It can be publicly communicated to the sender as part of SRTP session establishment to ensure that the sender derives the same set of session keys.

3.3 Attacks on ZRTP

Denial of service. ZRTP is potentially vulnerable to denial of service attacks caused by attackers simply sending spurious HELLO messages to end points and forcing the latter to keep state for every half-open connection. Eventually, end points will run out of storage or memory, and subsequent requests from legitimate clients will be refused.

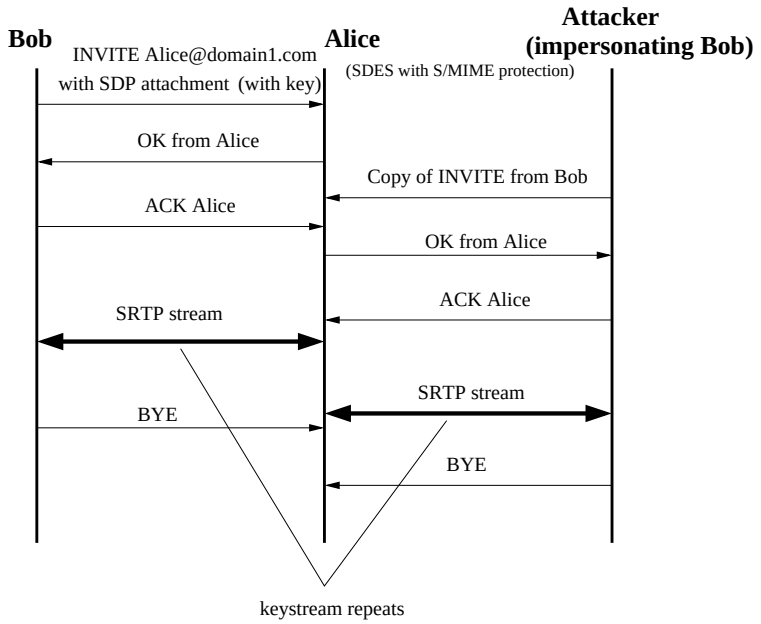

Figure 4: Attack on SRTP using SDES key exchange

Authentication. The main advantage of ZRTP is that it avoids the need for global trust associated with a PKI infrastructure. ZRTP aims to achieve this with the help of Short Authentication String (SAS), which is essentially a (keyed) cryptographic hash of Diffie-Hellman values along with other pre-shared secrets. After shared secrets have been used for authentication in one session, they are updated as described in section 2.3 and kept by participants to be used for authentication in the next session.

To authenticate the party on the other end of a VoIP session, the SAS value is read aloud over the voice connection. However, authentication based on SAS requires that some sort of GUI or display be available to the user. This is a serious problem for many secure VoIP devices, *e.g.*, those that implement VoIP via a local network proxy and lack a display. Therefore, we will focus upon security of ZRTP in the situation where the user cannot explicitly verify SAS over the voice connection.

Authentication in ZRTP is based on the assumption that, in order to launch a successful man-in-the-middle attack on a pair of participants who already conducted several sessions, the attacker must be present on every session starting from the very first one. The reasoning goes as follows. Each ZRTP user retains shared secrets `rs1` and `rs2` (see section 2.3) for users with whom he previously communicated. When initiating a new session, the user sends his ZID, which is used by the recipient to retrieve the set of shared secrets associated with this ZID. The session key is computed by hashing the joint Diffie-Hellman value concatenated with the shared secrets. Therefore, even if the DH exchange is compromised, the attacker still cannot compute the session key because he does not know the shared secrets. Because the shared secrets are re-computed after each session, the attacker must be present in every session starting from the very first one, in which there was no shared secret.

Unfortunately, this reasoning is fallacious. The main problem with the protocol is that ZIDs, which are used by recipients to look up shared secrets, are not authenticated early enough in the protocol exchange. Consider a passive attacker who eavesdrops on a session between Alice and Bob and learns Bob's ZID. He then stages a man-in-the-middle attack as shown in figure 5.

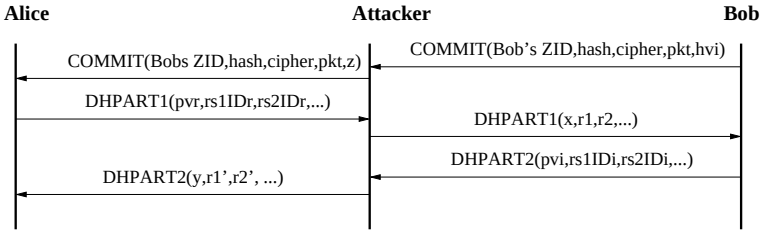

Figure 5: A man in the middle attack on the ZRTP protocol

The attacker chooses random exponents $\mathtt{x}', \mathtt{y}'$ and computes $\mathtt{x} = \mathtt{g}^{\mathtt{x}'} \bmod \mathtt{p}$ and $\mathtt{y} = \mathtt{g}^{\mathtt{y}'} \bmod \mathtt{p}$, respectively. z is the hash of x concatenated with the set of algorithms chosen by Bob for the ZRTP session. The attacker also replaces all shared-secret IDs with random numbers. When Alice receives the DHPART2 message from Bob, she retrieves the set of secrets that she shares with Bob and computes the set of expected IDs. Since the attacker has replaced all IDs with random numbers, they will not match.

According to the protocol specification, Alice should now assume that Bob has forgotten their shared secrets. She computes the joint Diffie-Hellman value as $\mathtt{y}^{\mathtt{svr}} \bmod \mathtt{p}$ $(= \mathtt{g}^{\mathtt{y}'.\mathtt{svr}} \bmod \mathtt{p})$. The session key is now computed as the hash of the joint Diffie-Hellman value *alone* because Alice believes that she doesn't have any shared secrets with Bob anymore.

Similarly, Bob computes the session key as the hash of the Diffie-Hellman value $\mathtt{g}^{\mathtt{x}'.\mathtt{svi}} \bmod \mathtt{p}$. The attacker knows both values. Therefore, he can compute SRTP master key and salt, and completely break SRTP encryption.

A possible fix to this protocol design flaw is to have Alice reject messages from Bob if the shared-secret IDs it contains do not match those expected by Alice. This is a poor solution, however, because it is not clear how parties can re-establish communication after they have truly lost their shared secrets.

Formal analysis of ZRTP in AVISPA. To support our analysis of ZRTP, we constructed a formal model of the protocol in the High Level Protocol Specification Language HLPSL [11] and used the automated AVISPA model checker [3] to carry out formal analysis.

Formal verification of ZRTP with AVISPA presents an interesting challenge because the model must capture the "multi-session" nature of authentication in ZRTP. Authentication in a ZRTP session depends on information exchanged in *previous* sessions. HLPSL, however, does not allow state to be retained across sessions.

To get our model to work, we had to assume that, for a given session, initiator and responder agree on the value of their shared secrets at the start of the session. This allows us to model the protocol by passing the shared secrets as arguments to the role specification of the initiator and responder roles. Also, we assume that there are no other shared secrets between the participants. The protocol specification in HLPSL is given in appendix B. We only model the relevant fields in message bodies. To simply presentation, we show the specification in the "Alice-Bob" notation:

$\mathtt{Init} \rightarrow \mathtt{Resp} : \mathtt{H}(\mathtt{g}^{\mathtt{x}})$

$\mathtt{Init} \leftarrow \mathtt{Resp} : \mathtt{g}^{\mathtt{y}}, \mathtt{rs1IDr}, \mathtt{rs2IDr}$ | Diffie-Hellman exchange

$\mathtt{Init} \rightarrow \mathtt{Resp} : \mathtt{g}^{\mathtt{x}}, \mathtt{rs1IDi}, \mathtt{rs2IDi}$

$\mathtt{Init} \leftarrow \mathtt{Resp} : \mathtt{MAC}(\mathtt{K}, \mathtt{c1})$ | K is shared key

$\mathtt{Init} \rightarrow \mathtt{Resp} : \mathtt{MAC}(\mathtt{K}, \mathtt{c2})$ | Authentication part

Here K is the shared session key calculated as described in section 2.3, H() is a cryptographic hash function and MAC(k, text) is a keyed message authentication code computed over text using authentication key k. rs1IDi, rs2IDi (rs1IDr, rs2IDr) are the keyed HMACs of the strings "Initiator" and "Responder" computed using the shared secrets rs1, rs2, respectively. c1, c2 are public constants.

Our specification of the authentication property is based on [6]. Intuitively, authentication holds for a particular session between Alice and Bob if the following condition holds: at the end of a successfully completed session, if Alice believes that she is talking to Bob, then she is indeed talking to Bob and their respective records of messages sent and received during the protocol execution "match." The condition for Bob is similar.

The attack on authentication described above was successfully discovered by the AVISPA tool. The attack is shown in appendix C.

3.4 Analysis of MIKEY

Secrecy. The goal of a cryptographically secure key exchange protocol is to establish a session key which is indistinguishable from a random bitstring by anyone other than the participants [10]. It is easy to see that MIKEY does not satisfy this requirement when executed in the Diffie-Hellman mode. The shared key is derived as $g^{x_i \cdot x_r}$, *i.e.*, the joint Diffie-Hellman value is used directly as the key. In many Diffie-Hellman groups, *e.g.*, in the group of squares modulo a large prime, testing group membership is not a computationally hard problem. Therefore, it is easy to tell the difference between a random bitstring and the key.

This does not necessarily lead to any exploitable weaknesses, although it does preclude a rigorous proof of security from going through. Moreover, encryption schemes in which the derived key is intended to be used typically require that the key be indistinguishable from a random value. This yet another example of how assumptions made by one layer of the VoIP protocol stack (transport layer in this case) are not met by the other layer (key exchange layer in this case).

There is a simple, standard solution. To derive the key from the joint Diffie-Hellman value, MIKEY participants should use a *randomness extractor*, *e.g.*, a universal hash function [20] with public randomness generated by one of the participants.

Finally, we observe that MIKEY in the pre-shared key mode obviously doesn't satisfy perfect forward secrecy because the compromise of the pre-shared secret leads to the compromise of all previous sessions.

Denial of service. MIKEY offers very limited protection against DoS attacks. In the public-key DH mode, the responder only performs CPU-intensive modular exponentiation after verifying the message digest of the initiator's message. The attacker can still flood the responder with multiple copies of the same message. This will cause the responder to perform digest verifications and may exhaust memory resources.

4 Related work

The two VoIP protocols that have attracted most attention in the research literature are SIP and Skype. SIP, in particular, has been the subject of several comprehensive studies [19, 22, 16, 25]. All of them focused solely on the signaling layer. Skype, which is a closed-end system based on a proprietary peer-to-peer protocol, has been the subject of several analyses [7, 4] and reverse engineering attempts [8].

To the best of our knowledge, the VoIP protocol stack, including key exchange and transport layer security protocols, has not been analyzed before in its entirety. We'd like to emphasize the need to analyze not only individual layers in isolation, but also assumptions and guarantees made by layers when interacting

with each other. As our study shows, "misunderstanding" between protocols at different layers is a common source of security vulnerabilities. We hope that this work will serve as the first step towards developing a comprehensive security assessment of the entire VoIP protocol stack.

5 Conclusions

We have presented a structured security analysis of the entire Voice-over-IP protocol stack, including signaling protocols such as SIP, key exchange protocols such as SDES, ZRTP and MIKEY, and transport-layer security protocols such as SRTP. Our analysis uncovered several serious vulnerabilities. The first is a replay attack on SDES key exchange which causes SRTP to use the same keystream in multiple sessions, thus allowing the attacker to remove encryption from SRTP-protected data streams. The second is an attack on ZRTP caused by unauthenticated user IDs, which allows the attacker to disable authentication mechanisms and trick a ZRTP participant into establishing a shared key with the attacker.

Our study illustrates the importance of thorough analysis of protocol specifications. This is especially critical in applications such as Voice-over-IP, where multiple protocols, operating at different layers in the protocol stack, have to make assumptions about each other to achieve end-to-end security. When these assumptions are not justified — such as the assumption made by SRTP that the key exchange protocol always ensures freshness of the key material — the result is a security vulnerability.

References

[1] F. Andreasen, M. Baugher, and D. Wing. Session Description Protocol (SDP) Security Descriptions for Media Streams. IETF RFC 4568, July 2006.

[2] J. Arkko, E. Carrara, F. Lindholm, M. Naslund, and K. Norrman. MIKEY: Multimedia Internet KEYing. IETF RFC 3830, August 2004.

[3] AVISPA. The AVISPA project. `http://www.avispa-project.org`, 2006.

[4] S. Baset and H. Schulzrinne. An analysis of the Skype peer-to-peer internet telephony protocol. In *Proc. 24th INFOCOM*. IEEE, 2006.

[5] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman. The Secure Real-time Transport Protocol (SRTP). IETF RFC 3711, March 2004.

[6] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Proc. Advances in Cryptology – CRYPTO 1993*, volume 773 of *LNCS*, pages 232–249. Springer, 1993.

[7] T. Berson. Skype security evaluation. `http://www.anagram.com/berson/abskyeval.html`, October 2005.

[8] P. Biondi and F. Desclaux. Silver needle in the Skype. `http://blackhat.com/presentations/bh-europe-06/bh-eu-06-biondi/bh-eu-06%-biondi-up.pdf`, March 2006.

[9] N. Borisov, I. Goldberg, and D. Wagner. Intercepting mobile communications: the insecurity of 802.11. In *Proc. 7th Annual Conference on Mobile Computing and Networking (MOBICOM)*, pages 180–189. ACM, 2001.

[10] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Proc. Advances in Cryptology - EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474. Springer, 2001.

[11] Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, J. Mantovani, S. Mödersheim, and L. Vigneron. A high level protocol specification language for industrial security-sensitive protocols. In *Proc. Workshop on Specification and Automated Processing of Security Requirements (SAPS)*, 2004.

[12] T. Dierks and C. Allen. The TLS Protocol Version 1.0. IETF RFC 2246, January 1999.

[13] M. Euchner. HMAC-Authenticated Diffie-Hellman for Multimedia Internet KEYing (MIKEY). IETF RFC 4650, September 2006.

[14] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. IETF RFC 2617, June 1999.

[15] J. Galvin, S. Murphy, S. Crocker, and N. Freed. Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted. IETF RFC 1847, October 1995.

[16] D. Geneiatakis, G. Kambourakis, T. Dagiuklas, C. Lambrinoudakis, and S. Gritzalis. SIP security mechanisms: A state-of-the-art review. In *Proc. 5th International Network Conference (INC)*, pages 147–155. ACM, 2005.

[17] M. Handley and V. Jacobson. SDP: Session Description Protocol. IETF RFC 2327, April 1998.

[18] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). IETF RFC 2409, November 2006.

[19] D. Richard Kuhn, T. Walsh, and S. Fries. Security considerations for Voice Over IP systems. NIST SP 800-58, January 2005.

[20] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proc. 22nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 427–437. ACM, 1990.

[21] B. Ramsdell. S/MIME Version 3 Message Specification. IETF RFC 2633, June 1999.

[22] J. Rosenberg and H. Schulzrinne. SIP traversal through residential and enterprise NATs and firewalls. `http://www.jdrosen.net/sip_entfw.html`, July 2001.

[23] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. IETF RFC 3261, June 2002.

[24] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. IETF RFC 3550, July 2003.

[25] D. Sisalem, S. Ehlert, D. Geneiatakis, G. Kambourakis, T. Dagiuklas, J. Markl, M. Rokos, O. Botron, J. Rodriguez, and J. Liu. Towards a secure and reliable VoIP infrastructure. `http://www.snocer.org/Paper/COOP-005892-SNOCER-D2-1.pdf`, May 2005.

[26] Zfone. The Zfone Project. `http://zfoneproject.com/`, 2006.

[27] P. Zimmermann. ZRTP: Extensions to RTP for Diffie-Hellman Key Agreement for SRTP. `http://www.philzimmermann.com/docs/draft-zimmermann-avt-zrtp-01.html`, March 2006.

A SIP security mechanisms

SIP security mechanisms can be broadly divided into those devoted to authentication, data integrity, and confidentiality.

Authentication. SIP supports *HTTP basic authentication* and *HTTP digest authentication* [14]. HTTP basic authentication requires username and matching password to be sent in plaintext as part of the HTTP header request. This has serious security risks, and HTTP basic authentication has been deprecated in SIP version 2 (SIPv2) [23].

HTTP digest authentication is based on a simple challenge-response paradigm. The digest authentication scheme challenges the remote user with a random *nonce*. A valid response consists of an MD5 or SHA-1 *digest* of the secret password, the nonce value and some other parameters including the requested URI. Although HTTP digest authentication improves upon the basic authentication by not sending the password in the clear, it is still prone to offline dictionary attacks based on intercepted hash values if short or weak passwords are used. The digest authentication scheme is also prone to computational denial-of-service attacks since it requires the challenger to compute the digest for any received hash value.

Confidentiality. SIP itself does not provide confidentiality for media data. SIP messages include MIME bodies and the MTME standard provides mechanisms for ensuring data integrity and confidentiality [15]. SIP may use Secure MIME (S/MIME) [21] for distribution of certificates, authentication, confidentiality and data integrity. Distribution of S/MIME certificates, however, requires the existence of a trusted server. Authenticating MIME payloads is not a problem since each end point has its own private signing key and certificates may be forwarded along with the signature. Confidentiality, on the other hand, poses a serious problem since it requires prior knowledge of the recipient's public key. This key must be fetched from a central authority or obtained from a peer via special SIP messages. To be able to protect SIP headers as well, tunneling of SIP messages inside MIME bodies is supported. Tunnelled packets may be large, and it is suggested to use TCP as the transport layer protocol to avoid problems with UDP fragmentation.

Another option is to use secure SIP (SIPS) URI, which is very similar to secure http (https) and employs Transport Layer Security (SSL/TLS). Since we wish to protect SIP headers and each hop may add routing information to the SIP message header, protection is on a hop-by-hop basis along each segment of the path. The use of TLS also requires the use of TCP as a transport protocol and depends on the existence of a PKI infrastructure.

The third option is IPSec. IPsec provides two mechanisms for authentication and, in the case of ESP, confidentiality: Authentication Header (AH) and Encapsulating Security Payload (ESP). Since each proxy server on the path may add or change information in the SIP header, both ESP and AH must be applied on a hop-by-hop basis. IPsec security mechanisms can also be established on a permanent basis between the end points without active involvement of the UAs themselves. SIP RFC does not specify which IPsec service may be used or how key management is realized. One commonly accepted key establishment protocol for IPSec Internet Key Exchange (IKE) [18]. IKE may be used with pre-shared secrets or PKI infrastructure. Since the UAs are mostly dynamic, IKE Main Mode will not work with pre-shared secrets and IKE Aggressive Mode is fraught with problems such as man in the middle attacks, offline dictionary attacks, *etc.*.

Data integrity. For data integrity, either S/MIME or SIPS URI or IPSec may be used.

B Formalization of ZRTP in HLPSL

```
role zrtp_Init (A,B: agent,
                G: nat,
```

```
               Hash: hash_func,
               RS1,RS2: nat,
               Snd,Rcv: channel(dy))
played_by A
def=
  local State          : nat,
        X              : text,
        DH, T          : nat,
        K              : symmetric_key,
        ID             : nat,
        Confirm1       : nat,
        Confirm2       : nat,
        EY             : nat,
        RS1IDr,RS2IDr  : nat,
        C,Init,Resp    : nat

  const sec_k1     : protocol_id

  init  State := 0
        /\ ID := 1
        /\ Confirm1 := 2
        /\ Confirm2 := 3
        /\ Init := 4
        /\ Resp := 5

  transition
   1. State = 0
      /\ Rcv(start)
      =|>
      State' := 1
      /\ X' := new()
      /\ Snd(Init.Hash(exp(G,X')))
   2. State = 1
      /\ Rcv(EY'.RS1IDr'.RS2IDr')
      /\ not(RS1IDr' = Hash(RS1.Resp))
      /\ not(RS2IDr' = Hash(RS2.Resp))
      =|>
      State' := 2
      /\ Snd(exp(G,X).Hash(RS1.Init).Hash(RS2.Init))
      /\ DH' := exp(EY',X)
      /\ K'  := Hash(Hash(DH'))
   3. State = 1
      /\ Rcv(EY'.RS1IDr'.RS2IDr')
      /\ RS1IDr' = Hash(RS1.Resp)
      /\ not(RS2IDr' = Hash(RS2.Resp))
      =|>
      State' := 2
      /\ Snd(exp(G,X).Hash(RS1.Init).Hash(RS2.Init))
      /\ DH' := exp(EY',X)
      /\ K'  := Hash(Hash(DH').RS1)
   4. State = 1
      /\ Rcv(EY'.RS1IDr'.RS2IDr')
      /\ not(RS1IDr' = Hash(RS1.Resp))
      /\ RS2IDr' = Hash(RS2.Resp)
     =|>
      State' := 2
      /\ Snd(exp(G,X).Hash(RS1.Init).Hash(RS2.Init))
      /\ DH' := exp(EY',X)
      /\ K'  := Hash(Hash(DH').RS2)
   5. State = 1
      /\ Rcv(EY'.RS1IDr'.RS2IDr')
      /\ RS1IDr' = Hash(RS1.Resp)
      /\ RS2IDr' = Hash(RS2.Resp)
      =|>
      State' := 2
```

```
      /\ Snd(exp(G,X).Hash(RS1.Init).Hash(RS2.Init))
      /\ DH' := exp(EY',X)
      /\ K'  := Hash(Hash(DH').RS1.RS2)
   6. State = 2
      /\ Rcv(C')
      /\ C' = Hash(K.Confirm2)
      =|>
      State' := 3
      /\ RS2' := RS1
      /\ RS1' := Hash(K.ID)
      /\ Snd(Hash(K.Confirm1))
      /\ secret(K,sec_k1,{A,B})
      /\ witness(A,B,na,Hash(K.Confirm1))
      /\ request(A,B,nb,C')
end role
```

```
role zrtp_Resp (A,B: agent,
                G: nat,
                Hash: hash_func,
                RS1,RS2: nat,
                Snd,Rcv: channel(dy))
played_by B
def=
  local State          : nat,
        Y              : text,
        DH, T          : nat,
        K              : symmetric_key,
        HVI            : nat,
        ID             : nat,
        Confirm1       : nat,
        Confirm2       : nat,
        EX             : nat,
        RS1IDi,RS2IDi  : nat,
        C,Init,Resp    : nat

  const sec_k2     : protocol_id

  init  State := 0
        /\ ID := 1
        /\ Confirm1 := 2
        /\ Confirm2 := 3
        /\ Init := 4
        /\ Resp := 5

  transition
   1. State = 0
      /\ Rcv(Init.HVI')
      =|>
      State' := 1
      /\ Y'  := new()
      /\ Snd(exp(G,Y').Hash(RS1.Resp).Hash(RS2.Resp))
   2. State = 1
      /\ Rcv(EX',RS1IDi',RS2IDi')
      /\ not(RS1IDi' = Hash(RS1.Init))
      /\ not(RS2IDi' = Hash(RS2.Init))
      /\ HVI = Hash(EX')
      =|>
      State' := 2
      /\ DH' := exp(EX',Y)
      /\ K'  := Hash(Hash(DH'))
      /\ Snd(Hash(K'.Confirm2))
      /\ witness(B,A,nb,Hash(K'.Confirm2))
   3. State = 1
      /\ Rcv(EX',RS1IDi',RS2IDi')
      /\ RS1IDi' = Hash(RS1.Init)
```

```
      /\ not(RS2IDi' = Hash(RS2.Init))
      /\ HVI = Hash(EX')
      =|>
      State' := 2
      /\ DH' := exp(EX',Y)
      /\ K'  := Hash(Hash(DH').RS1)
      /\ Snd(Hash(K'.Confirm2))
      /\ witness(B,A,nb,Hash(K'.Confirm2))
   4. State = 1
      /\ Rcv(EX',RS1IDi',RS2IDi')
      /\ not(RS1IDi' = Hash(RS1.Init))
      /\ RS2IDi' = Hash(RS2.Init)
      /\ HVI = Hash(EX')
      =|>
      State' := 2
      /\ DH' := exp(EX',Y)
      /\ K'  := Hash(Hash(DH').RS2)
      /\ Snd(Hash(K'.Confirm2))
      /\ witness(B,A,nb,Hash(K'.Confirm2))
   5. State = 1
      /\ Rcv(EX',RS1IDi',RS2IDi')
      /\ RS1IDi' = Hash(RS1.Init)
      /\ RS2IDi' = Hash(RS2.Init)
      /\ HVI = Hash(EX')
      =|>
      State' := 2
      /\ DH' := exp(EX',Y)
      /\ K'  := Hash(Hash(DH').RS1.RS2)
      /\ Snd(Hash(K'.Confirm2))
      /\ witness(B,A,nb,Hash(K'.Confirm2))
   6. State = 2
      /\ Rcv(C')
      /\ C' = Hash(K.Confirm1)
      =|>
      State' := 3
      /\ RS2' := RS1
      /\ RS1' := Hash(K.ID)
      /\ secret(K,sec_k2,{A,B})
      /\ request(B,A,na,C')
end role
```

```
role session(A,B: agent,
             G: nat,
             H: hash_func,
             RS1, RS2: nat)
def=
  local SA, RA, SB, RB: channel (dy)

  composition
     zrtp_Init(A,B,G,H,RS1,RS2,SA,RA)
  /\ zrtp_Resp(A,B,G,H,RS1,RS2,SB,RB)

end role
```

```
role environment()
def=
  const a, b     : agent,
        rs1,rs2  : nat,
        na, nb   : protocol_id,
        g        : nat,
        h        : hash_func

  intruder_knowledge={a,b,g,h}

  composition
```

```
      session(a,b,g,h,rs1,rs2)
   /\ session(b,a,g,h,rs1,rs2)

end role
```

```
goal

 % Confidentiality
 secrecy_of sec_k1, sec_k2

 % Message authentication
 authentication_on na

 % Message authentication
 authentication_on nb

end goal
```

```
environment()
```

NOTE: We use $Hash$ instead of $HMAC$ since AVISPA does not support keyed MAC's.

C AVISPA trace of ZRTP attack

```
SUMMARY
  UNSAFE

DETAILS
  ATTACK_FOUND
  UNTYPED_MODEL

PROTOCOL
  ./ZRTP.if

GOAL
  Authentication attack on (b,a,nb,{{{exp(g,n19(Y)*n37(X))}_h.rs1.rs2}_h.3}_h)

BACKEND
  CL-AtSe

STATISTICS

  Analysed   : 451 states
  Reachable  : 115 states
  Translation: 0.21 seconds
  Computation: 0.25 seconds


ATTACK TRACE
 i -> (a,3):  start
 (a,3) -> i:  4.{exp(g,n1(X))}_h

 i -> (a,3):  EY(2).RS1IDr(2).RS2IDr(2)
              & RS2IDr(2)<>{rs2.5}_Hash(2);  RS1IDr(2)<>{rs1.5}_Hash(2);
 (a,3) -> i:  exp(g,n1(X)).{rs1.4}_h.{rs2.4}_h

 i -> (a,7):  4.{EX(68)}_h
 (a,7) -> i:  exp(g,n55(Y)).{rs1.5}_h.{rs2.5}_h

 i -> (a,7):  EX(68)
 (a,7) -> i:  {{{exp(EX(68),n55(Y))}_h.rs1.rs2}_h.3}_h
              & Witness(a,b,nb,{{{exp(EX(68),n55(Y))}_h.rs1.rs2}_h.3}_h);
```

```
i -> (b,6):  start
(b,6) -> i:  4.{exp(g,n37(X))}_h

i -> (b,4):  4.{exp(g,n37(X))}_h
(b,4) -> i:  exp(g,n19(Y)).{rs1.5}_h.{rs2.5}_h

i -> (b,6):  exp(g,n19(Y)).{rs1.5}_h.{rs2.5}_h
(b,6) -> i:  exp(g,n37(X)).{rs1.4}_h.{rs2.4}_h

i -> (b,4):  exp(g,n37(X))
(b,4) -> i:  {{{exp(g,n19(Y)*n37(X))}_h.rs1.rs2}_h.3}_h
             & Witness(b,a,nb,{{{exp(g,n19(Y)*n37(X))}_h.rs1.rs2}_h.3}_h);

i -> (b,6):  {{{exp(g,n37(X)*n19(Y))}_h.rs1.rs2}_h.3}_h
(b,6) -> i:  {{{exp(g,n37(X)*n19(Y))}_h.rs1.rs2}_h.2}_h
             & Secret({{exp(g,n37(X)*n19(Y))}_h.rs1.rs2}_h,set_119);
             & Witness(b,a,na,{{{exp(g,n37(X)*n19(Y))}_h.rs1.rs2}_h.2}_h);
             & Request(b,a,nb,{{{exp(g,n37(X)*n19(Y))}_h.rs1.rs2}_h.3}_h);
             & Add b to set_119;  Add a to set_119;
```