Security Analysis of Voice-over-IP Protocols

Prateek Gupta
VMWare, Inc.

Vitaly Shmatikov
The University of Texas at Austin

Abstract

The 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 allows the attacker to convince the communicating parties that they have lost their shared secret. If they are using VoIP devices without displays and thus cannot execute the "human authentication" procedure, they are forced to communicate insecurely, or not communicate at all, i.e.*, this becomes a denial of service attack. 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. Our objective in this paper is to present a structured analysis of protocol inter-operation in the VoIP stack, and to demonstrate how even a subtle mismatch between the assumptions made by a protocol at one layer about the protocol at another layer can lead to catastrophic security breaches, including complete removal of transport-layer encryption.

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*. This division is quite natural, since each layer is typically implemented by a separate protocol.

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) [27], H.323 and MGCP. Session description protocols such as SDP [20] are used for initiating multimedia and other sessions, and often include key exchange as a sub-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. This is the fundamental building block in secure session establishment. Security of the media transport layer—the layer in which the 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 sent and received messages must *match* (*e.g.*, see the notion of "matching conversations" in [8]). Key exchange protocols for VoIP sessions include SDP's Security DEscriptions for Media Streams (SDES) [1], Multimedia Internet KEYing (MIKEY) [2] and ZRTP [31]. We will analyze all three in this paper.

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

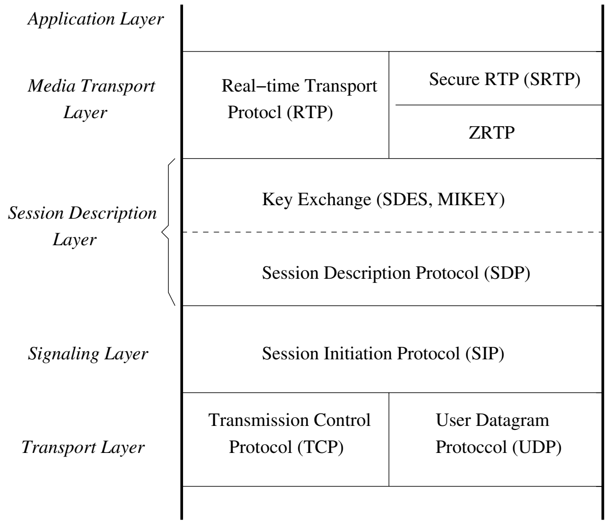
Application Layer
Media Transport Layer
Real–time Transport Protocl (RTP)
Secure RTP (SRTP)
ZRTP
Session Description Layer
Key Exchange (SDES, MIKEY)
Session Description Protocol (SDP)
Signaling Layer
Session Initiation Protocol (SIP)
Transport Layer
Transmission Control Protocol (TCP)
User Datagram Protoccol (UDP)

Figure 1. Voice-over-IP protocol stack

will be detected by the recipient. An example of a secure media transport protocol used on VoIP communications is Secure Real-time Transport Protocol (SRTP) [6], which is a profile of Real-time Transport Protocol (RTP) [28].

Our contributions. We analyze security of VoIP protocols at all layers of the VoIP stack. In particular, we focus on the 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 the transport-layer SRTP protocol to repeat the keystream used for datagram encryption. This enables the attacker to obtain the `xor` of plaintext datagrams or even to 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 the keystream that it used before, with devastating consequences. This attack is confirmed by our analysis of the `libsrtp` implementation.

- We show an attack on the ZRTP key exchange protocol that allows the attacker to convince ZRTP session participants that they have lost their shared secret. 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. If the attacker deliberately chooses values that cause verification to fail, A will decide—following ZRTP specification—that B has “forgotten” the shared secret.

  The ZRTP specification explicitly says that the protocol may proceed even if the set of shared secrets is empty, in which case the attacker ends up sharing a key with A who thinks she shares this key with B. Even if the participants stop the protocol after losing their shared secrets, but are using VoIP devices without displays, they cannot confirm the computed key by voice and must stop communicating. In this case, the attack becomes a simple and effective denial of service. 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.*, [12]). 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. Moreover, even hashing the Diffie-Hellman value does not allow the formal proof of security to go through in this case, since the hash function does not take any random inputs apart from the Diffie-Hellman value and cannot be viewed as a randomness extractor in the proof. (This observation does not immediately lead to any attacks.)

While we demonstrate several real, exploitable vulnerabilities in VoIP security protocols, our main contribution is to highlight the importance of analyzing protocols in context rather than in isolation. Specifications of VoIP protocols tend to be a mixture of informal prose and pseudocode, with some assumptions—especially those about the protocols operating at the other layers of the VoIP stack—are left implicit and vague. Therefore, our study has important lessons for the design and analysis of security protocols in general.

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) [27] 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* (UA) 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 the 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. SIP also supports *forking proxies*, which receive a single request and forward it to multiple recipients (as we show in section 3.1, this makes SIP potentially vulnerable to denial of service attacks). In the 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 the 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.

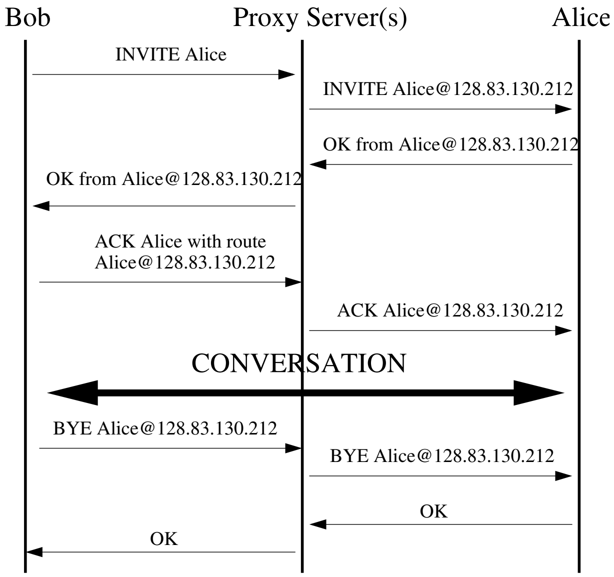

Figure 2. SIP protocol exchange

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) [15]. 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 [20]. A multimedia *session* is a set of multimedia senders and receivers and the data streams flowing between them; 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, ZRTP, and MIKEY

Unlike session initiation and description, key exchange is a fundamental *security* mechanism. Therefore, we describe the key exchange protocols specific to VoIP in more detail. It is essential to understand what security guarantees they provide, because, as we show below, a mismatch between the expectations of the transport-layer protocol and the security properties actually ensured by the key exchange protocol can be a source of serious vulnerabilities.

SDES. SDES [1] is the key transport extension of the SDP protocol. It 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 `crypto` attribute for SRTP is defined as: $\mathtt{a} = \mathtt{crypto} : \langle\mathtt{tag}\rangle\langle\mathtt{crypto} - \mathtt{suite}\rangle\langle\mathtt{key} - \mathtt{params}\rangle[\langle\mathtt{session} - \mathtt{params}\rangle]$. The most important component is $\mathtt{key}{-}\mathtt{params}$, which specifies one or more cryptographic keys as $\langle\mathtt{key} - \mathtt{method}\rangle : \langle\mathtt{key} - \mathtt{info}\rangle$. The only method supported for key exchange is `inline` :, which specifies that the key itself must be included in plaintext. In other words, the key is embedded directly in the SDP attachment of a SIP message. Therefore, protection of the key depends solely on SIP.

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 [15] (if the transport layer is TCP), or S/MIME [25]. The use of TLS is deprecated because TLS 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 [18].

Note that S/MIME does *not* provide any replay protection. Hence, if S/MIME is used to protect SDP payload (which includes the key in plaintext), 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 [31] describes an extension header for Real-time Transport Protocol (RTP) to establish a session key for SRTP sessions using authenticated Diffie-Hellman key exchange. An implementation of ZRTP is available as Zfone [30]. The main distinguishing feature of ZRTP is that it does not require prior shared secrets or the existence of a separate public-key infrastructure (PKI). This is an important consideration since it eliminates the need for a trusted certificate server.

Because 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 key confirmation. The communicating parties confirm the established key verbally over the phone, by looking at their respective phone displays and reading the displayed SAS values to each other. After that, they rely on key chaining: the shared Diffie-Hellman secrets cached from the previous sessions are used to authenticate the current session.

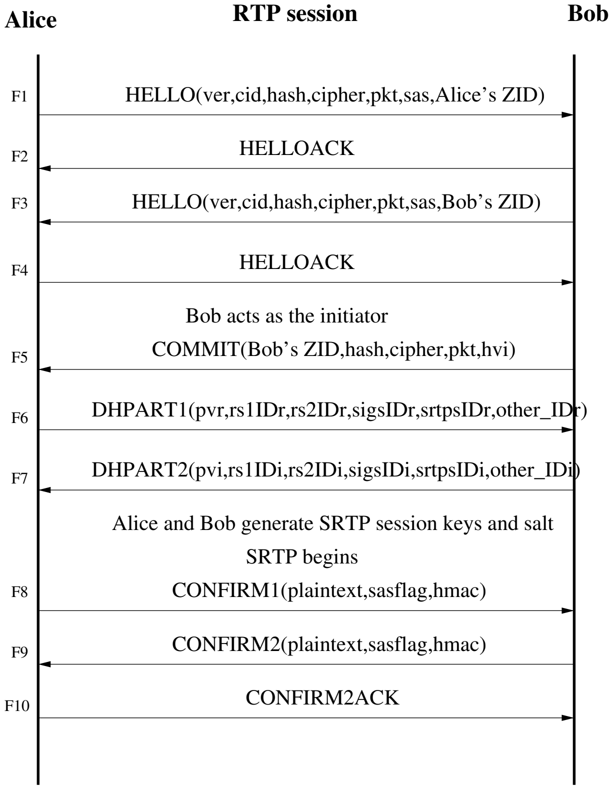
Alice
RTP session
Bob
F1
HELLO(ver,cid,hash,cipher,pkt,sas,Alice's ZID)
F2
HELLOACK
F3
HELLO(ver,cid,hash,cipher,pkt,sas,Bob's ZID)
F4
HELLOACK
Bob acts as the initiator
F5
COMMIT(Bob's ZID,hash,cipher,pkt,hvi)
F6
DHPART1(pvr,rs1IDr,rs2IDr,sigsIDr,srtpsIDr,other_IDr)
F7
DHPART2(pvi,rs1IDi,rs2IDi,sigsIDi,srtpsIDi,other_IDi)
Alice and Bob generate SRTP session keys and salt
SRTP begins
F8
CONFIRM1(plaintext,sasflag,hmac)
F9
CONFIRM2(plaintext,sasflag,hmac)
F10
CONFIRM2ACK

Figure 3. Establishment of SRTP session key using ZRTP

Figure 3 shows a ZRTP key exchange between users Alice and Bob. 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. The HELLO and HEL-

LOACK 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. In the COMMIT message, `hash`, `cipher` and `pkt` describe the hash, encryption and public key algorithms, respectively, chosen by Bob from the intersection of algorithms in the sent and received HELLO messages. Bob chooses a random exponent `svi` and computes the value $\mathtt{pvi} = \mathtt{g}^{\mathtt{svi}} \bmod \mathtt{p}$, 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`. Each secret already shared between Alice and Bob has an ID, which is the `HMAC` of the string "`Responder`" computed using this secret as the key. Alice uses Bob's ZID to retrieve their shared secrets `rs1` and `rs2`. Bob's behavior in response to Alice's DHPART1 message is similar.

Upon receipt of the DHPART2 message, Alice checks that Bob's public DH value is not equal to `1` or $\mathtt{p} - \mathtt{1}$. (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 it.) 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 the IDs in the intersection are stored as set `D`, sorted in the ascending order. The specification explicitly allows this set of shared secrets to be empty [31, p.12].

The final session key is computed as the hash of the joint Diffie-Hellman secret concatenated with the set `D` of shared secrets. Finally, cached shared secrets `rs1` and `rs2` are updated as $\mathtt{rs2} = \mathtt{rs1}$ and $\mathtt{rs1} = \mathtt{HMAC}(\mathtt{session\,key}, \text{“known plaintext”})$ 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.

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 authenticated Diffie-Hellman (DH) key exchange. A later extension provides for a DH exchange in the pre-shared key mode [16]. These modes are described in detail in appendix B.

An advantage of MIKEY is that it allows the key to be negotiated 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 either prior shared secrets, or a separate PKI, with all attendant problems such as certificate dispersal, revocation, and so on.

For the purposes of our analysis, we focus on the Diffie-Hellman mode of MIKEY, described in detail in appendix B. In this mode, the initiator and the responder exchange their respective Diffie-Hellman values $\mathtt{g}^{\mathtt{x_i}}$ and $\mathtt{g}^{\mathtt{x_r}}$. Both values are signed to ensure authentication. The derived key is $\mathtt{g}^{\mathtt{x_i} \cdot \mathtt{x_r}}$.

2.4 Secure transport layer: SRTP

VoIP datagrams are usually transported using the Real-time Transport Protocol (RTP). SRTP [6] is a profile of 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.

In SRTP, 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. A cryptographic context for SRTP is identified 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 the 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) to generate 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). As we show in section 3.2, the determinism of key derivation is a fatal flaw since it makes an unwarranted assumption about the key exchange protocol used to create the master key.

Session key derivation involves an 8-bit `label`, master salt $\mathtt{m_s}$, the key derivation rate, as determined by 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 `r` is the integer quotient obtained by dividing the 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 `k`, given m-bit `x`, the output is an n-bit string computationally indistinguishable from a truly random n-bit string. The session keys are generated as $\mathtt{PRF}(\mathtt{m_k}, \mathtt{x})$, using 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 will allow us to break the protocol because security of the stream cipher-like encryption used in SRTP depends critically on the keystream never repeating. This is emphasized several times in the SRTP specification [6], which warns about the dangers of "two-time pad" (a colloquial term for keystream reuse).

For the keystream never to repeat, the PRF output (used as input into AES) must never repeat, because the other inputs into AES—SSRC and SEQ—are not required to be globally unique and can repeat from session to session (SSRC must be unique within a session, but may and will repeat from session to session if the same sender is involved). Moreover, both values are public and can be eavesdropped by the attacker. This means that the *PRF input must be unique for every session.*

Therefore, the master key and master salt must be unique in each session. Indeed, according to the SRTP specification, "a master key MUST NOT be shared among different RTP sessions." Recall that both the master key and master salt 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 master key 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 master key, completely breaking the transport-layer 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. An attacker can put the victim's IP address into a spoofed `Router header` request, and send it to forking proxies, who will greatly amplify the number of messages returned to the victim.

Reflection is an another way to stage a denial of service attack. An attacker can send spoofed requests to a large number of SIP elements and proxies, putting the victim's IP address into the source field. Each of the recipients will generate a response, overwhelming the victim.

A limited protection against spoofed SIP requests can be provided by IPsec, but end-to-end IPsec is challenging to deploy in a typical VoIP environment where end points are dynamic, and it is not clear from the specification how SIP inter-operates with IPsec (see appendix A).

Another important 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.

A wide variety of denial of service attacks also become possible if registration 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, 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.

SIP registration does not require the `From` field of a message to be the same as the `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 the 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.

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. (Fig. 4 shows the sessions running concurrently, but the attack need not be adaptive; one session can be executed after the other.) 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 the 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.

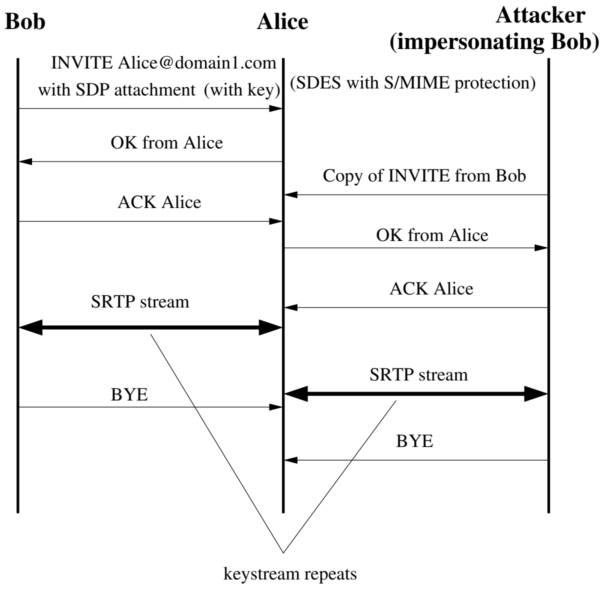

Figure 4. Attack on SRTP using SDES key exchange

In any case, encryption has been completely removed, and the attacker obtains a bitwise xor of two payloads. Since VoIP datagrams are highly redundant, and the payloads of at least the initial datagrams are very predictable (*e.g.*, most phone conversations start with a "Hello", whose digital encoding can be predicted, even accounting for variations in accent and pronunciation), this should be considered a complete security breach. Detailed analysis of redundancy in network packets and the implications for stream cipher security can be found in [23].

Our attack is similar in spirit to the famous attack on 802.11b WEP [11], which also allowed the attacker to obtain an xor of wireless packets. In the case of WEP, the keystream was re-used due to exhaustion of initialization vectors for the stream cipher.

We emphasize that our attack on SDES/SRTP is not a theoretical exercise. We tested libsrtp, an open-source implementation of SRTP, and confirmed that it generates the same master key if the same key material is supplied by the key exchange protocol, and the replay of an S/MIME-protected message will result in exactly the same key material being supplied to the SRTP implementation.

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, even though the designers of SRTP were clearly aware of the dangers of master key re-use [6]. The SRTP specification emphasizes the need to use *automatic* key management mechanisms, since manual key management is more prone to result in key re-use. Among the automatic key management protocols compatible with SRTP are MIKEY, SDES, and ZRTP.

Even though MIKEY was specifically designed as the key exchange protocol for SRTP, many VoIP implementations use SDES instead. As of April 2007, products that rely on the SDES/SRTP combination include software PBX (Private Branch Exchange) from `pbxnsip`, VoIP session border controllers and SIP firewalls from Covergence and Ingate Systems, and the eyeBeam software phone from CounterPath.

While MIKEY contains built-in anti-replay protection and thus appears suitable for establishing SRTP keys, our analysis demonstrates that SDES is not. Some SRTP implementations may take additional measures to prevent key re-use, but the `libsrtp` implementation relies completely (and disastrously) on the key exchange protocol to ensure freshness of the key material.

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.

Overall, SRTP is a well-designed protocol, and there are good practical reasons why AES in counter mode has been chosen as the keystream generator in SRTP [7]. Nevertheless, we believe that our analysis contains an important lesson for the designers of other protocols that use stream-like ciphers in counter modes. As the authors of SRTP re-iterate several times in the protocol specification, it is critical that the counter never repeat. For the protocol to be secure, however, AES key/salt combinations must be unique even across multiple sessions. In SRTP, this responsibility is *implicitly* delegated to the key exchange protocol. Even the SRTP FAQ says only that keys "can be provided via signaling, and can be expressed using the SDP Security Descriptions (if the signaling is cryptographically protected) or MIKEY [. . .]" [7]. Unfortunately, cryptographic protection in the absence of replay protection is not enough to guarantee uniqueness of the keys across multiple sessions, resulting in a potentially vulnerable combination of key exchange and stream cipher.

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. In response to each HELLO, a ZRTP endpoint creates a half-open connection, and keeps its parameters in memory. Eventually, it will run out of storage or memory, and subsequent requests from legitimate clients will be refused.

Authentication. The main advantage of ZRTP is that it avoids the need for global trust associated with a public-key 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 the participants 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.

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.

The protocol specification explicitly allows the set of shared secrets to be empty: "the final shared secret, s0,

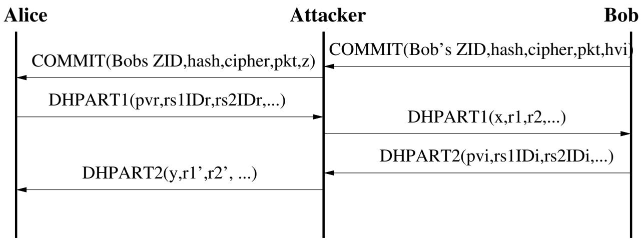
Alice
Attacker
Bob
COMMIT(Bobs ZID,hash,cipher,pkt,z)
COMMIT(Bob's ZID,hash,cipher,pkt,hvi
DHPART1(pvr,rs1IDr,rs2IDr,...)
DHPART1(x,r1,r2,...)
DHPART2(pvi,rs1IDi,rs2IDi,...)
DHPART2(y,r1',r2', ...)

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

is calculated by hashing the concatenation of the Diffie-Hellman shared secret (DHSS) followed by the (possibly empty) set of shared secrets that are actually shared between the initiator and responder" [31, p.12]. The specification does *not* require Alice to stop the protocol, and instead instructs her to compute 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.

An informal discussion in the preamble to the protocol specification does say that "If we ever lose this cached shared secret, it is no longer available for authentication of DH exchanges, so we would have to do a new SAS procedure and start over with a new cached shared secret" [31, p.3]. First, this is inconsistent with the actual specification, which permits the set of shared secrets to be empty during key derivation (see above). Second, it is not at all clear how to implement this, since the preamble goes on to say that "SAS is easiest to implement when a GUI or some sort of display is available, which raises the question of what to do when no display is available. We envision some products that implement secure VoIP via a local network proxy, which lacks a display in many cases."

Even if Alice stops communicating with Bob when the set of shared secrets is empty (and we emphasize once again that this is *not* what the specification prescribes), this attack turns into a very effective denial of service, which allows the attacker to break off any session conducted between VoIP devices without displays. In general, the claim in the specification that the attacker is forced to solve multiple problems, such as "stealing a shared secret from one of the parties, being present on the very first session and every subsequent session to carry out an active MitM attack, and solving the discrete log problem," does not appear to be borne out.

It is not clear whether the problem can be solved at all. In the absence of either PKI, or pre-shared secrets, or device support for "out-of-band" key confirmation by reading key hashes to each other, it is hard to see how the parties can carry out an authenticated key exchange. At the very least, the specification of ZRTP should not allow the key exchange to go forward when the set of shared secrets is empty.

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 [13] 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 C. We only model the relevant fields in message bodies. To simplify 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 $\mathtt{K}$ is the shared session key calculated as described in section 2.3, $\mathtt{H}()$ is a cryptographic hash function and $\mathtt{MAC}(\mathtt{k}, \mathtt{text})$ is a keyed message authentication code computed over $\mathtt{text}$ using authentication key $\mathtt{k}$. $\mathtt{rs1IDi}, \mathtt{rs2IDi}$ ($\mathtt{rs1IDr}, \mathtt{rs2IDr}$) are the keyed HMACs of the strings "Initiator" and "Responder" computed using the shared secrets $\mathtt{rs1}, \mathtt{rs2}$, respectively. $\mathtt{c1}, \mathtt{c2}$ are public constants.

Our specification of the authentication property is based on [8]. 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 D.

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 [12]. 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: it is sufficient to compute the Jacobi symbol. 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 is yet another example of how assumptions made by one layer of the VoIP protocol stack (transport layer in this case) are not met by another layer (key exchange layer in this case).

But wouldn't the joint Diffie-Hellman derived in MIKEY have to be hashed anyway before it can actually be used as a key? Yes, but this is not enough. A simple application of a deterministic hash function to the joint Diffie-Hellman value does *not* provably produce an output which is indistinguishable from random. By contrast, in protocols like TLS [15] and IKE [21], the key is derived by hashing the Diffie-Hellman value together with some (authenticated) *random values* generated by one or both participants. This use of randomness in key derivation, absent in MIKEY, is essential for the cryptographic proof of security to go through.

There is a simple, standard solution. To derive a key from the joint Diffie-Hellman value, MIKEY participants should follow the approach used in TLS and IKE, and use a *randomness extractor*, *e.g.*, a universal hash function [24] with public randomness generated by one or both participants.

Finally, we observe that MIKEY in the pre-shared key mode (see appendix B) 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 denial of service 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 or messages containing an incorrect digest. This will cause the responder to perform a large number 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 [22, 26, 19, 29]. 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 [9, 5] and reverse engineering attempts [10].

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 the individual layers in isolation, but also the assumptions and guarantees made by the layers when interacting with each other. As our study shows, "misunderstandings" 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 either trick a ZRTP participant into establishing a shared key with the attacker, or cause the protocol to terminate prematurely. The third is a "certificational" issue: due to the lack of proper randomness extraction in MIKEY key derivation, MIKEY cannot be proved cryptographically secure.

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.

Much attention has been devoted to the development of formal protocol analysis methods that provide *compositional* security guarantees (*e.g.*, see [4, 14]). An interesting topic of future research is whether attacks that exploit "misunderstandings" between protocol layers can be discovered automatically by analyzing protocol specifications. This requires formal tools that are capable of inferring and modeling inter-protocol assumptions such as "protocol A assumes that the key provided by protocol B will be fresh in every session."

Acknowledgements. This material is based upon work supported by the National Science Foundations under grants CNS-0509033 and IIS-0534198, and by the Department of Defense under Contract No. H98230-05-C-1650.

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] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *Proc. 10th ACM Conference on Computer and Communications Security (CCS)*, pages 220–230. ACM, 2003.

[5] S. Baset and H. Schulzrinne. An analysis of the Skype peer-to-peer Internet telephony protocol. In *Proc. 25th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*. IEEE, 2006.

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

[7] M. Baugher, D. McGrew, M. Naslund, and K. Norrman. Secure RTP Frequently Asked Questions. `http://srtp.sourceforge.net/faq.html`, October 2005.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] Y. Chevalier, L. Compagna, J. Cuellar, P. H. 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.

[14] A. Datta, A. Derek, J. Mitchell, and D. Pavlovic. A derivation system and compositional logic for security protocols. *J. Computer Security*, 13(3):423–482, 2005.

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

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

[17] 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.

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

[19] 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.

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

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

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

[23] D. McGrew and S. Fluhrer. Attacks on additive encryption of redundant plaintext and implications on internet security. In *Proc. 7th Annual International Workshop on Selected Areas in Cryptography (SAC)*, volume 2012 of *LNCS*, pages 14–28. Springer, 2000.

[24] 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.

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

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

[27] 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.

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

[29] 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.

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

[31] P. Zimmermann. ZRTP: Extensions to RTP for Diffie-Hellman Key Agreement for SRTP. `http://www1.tools.ietf.org/html/draft-zimmermann-avt-zrtp-01`, 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* [17]. 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) [27].

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 [18]. SIP may use Secure MIME (S/MIME) [25] 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) [21]. 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 Modes of MIKEY

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 $\mathtt{s}$ and a random value using a cryptographically secure hash function. Let $\mathtt{ID_i}$ and $\mathtt{ID_r}$ be the identities of the initiator and responder, respectively. Message $\mathtt{m}$ is defined as $\mathtt{m} := \mathtt{HDR}, \mathtt{T}, \mathtt{RAND}, [\mathtt{ID_i}], [\mathtt{ID_r}], \{\mathtt{SP}\}, \mathtt{KEMAC}$, where $\mathtt{T}$ is the timestamp (used for replay protection), $\mathtt{RAND}$ is a random number used for generating encryption key ($\mathtt{Encr_k}$)

and authentication key ($\mathtt{Auth_k}$) from the shared secret $\mathtt{s}$, $\mathtt{SP}$ is a set of security policies and $\mathtt{KEMAC} = \mathtt{E}(\mathtt{Encr_k}, \{\mathtt{TGK}\})||\mathtt{MAC}$. $\mathtt{E}(\mathtt{key}, \mathtt{text})$ denotes the encryption of $\mathtt{text}$ with the encryption key $\mathtt{key}$ and $||$ denotes string concatenation. $\mathtt{MAC}$ is a keyed message authentication code computed over the entire message $\mathtt{m}$ using the authentication key $\mathtt{Auth_k}$. It is assumed that $\mathtt{TGK}$ 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 $\mathtt{HDR}$, timestamp $\mathtt{T}$, the initiator and responder identities $\mathtt{ID_i}$, $\mathtt{ID_r}$, 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

  $$\mathtt{m} := \mathtt{HDR}, \mathtt{T}, \mathtt{RAND}, [\mathtt{ID_i}|\mathtt{CERT_i}], [\mathtt{ID_r}], \{\mathtt{SP}\}, \mathtt{KEMAC}, \mathtt{PKE}, \mathtt{SIGN_i}$$

  Here $\mathtt{CERT_i}$ stands for the initiator's certificate. In this mode, the encryption and authentication keys are derived from an *envelope key* ($\mathtt{Env_k}$) chosen by the initiator at random. $\mathtt{PKE}$ is the encryption of $\mathtt{Env_k}$ 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 g. Authenticated Diffie-Hellman key exchange is shown below:

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

  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.

C 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.

D 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;
```