CHAPTER 7

KEY MANAGEMENT MECHANISMS

Key management is a fundamental part of protecting Internet multimedia applications such as VoIP. At the same time, key management protocols are difficult to design, especially for multimedia applications that require group participation (for example, videoconferencing, broadcasting or multicast audio, video or file transfer). Until recently, various key-exchange mechanisms such as IKE were proven to support asynchronous communications (that is, file transfer) but were not suitable for group or multicast Internet multimedia applications. Therefore, a distinct effort has been initiated within the IETF to establish such capability. The IETF RFC 4046, "MSEC Group Key Management Architecture," defines an architecture that consists of abstractions and design principles for developing key management protocols. The MSEC architecture defines a set of requirements for developing key management protocols.[1] These requirements discuss the properties and principles that key management protocols should exhibit for scalability, group security policy, associations (encryption key, lifetime, and so on), group membership, rekeying, attack deterrence, and recovery from compromise.

Multimedia communications such as VoIP require key negotiation protocols that can provide robust and extensible capabilities for multicast and unicast communications. For example, protocols such as TLS and IKE do not provide such capabilities. Group key management protocols can be used to protect multicast and unicast communications between users, user groups, and subgroups (through the group security association). In addition, they have to demonstrate resistance to attacks from external and internal sources (that is, impersonations, DoS).

Within the MSEC architecture, a multicast or group security architecture is defined in which key negotiation and key management are components. Negotiation of keying material is one of the most challenging topics

1. *M. Baugher, et al. Multicast Security (MSEC) Group Key Management Architecture. IETF RFC 4046, April 2005.*

for VoIP (and, generally, Internet multimedia applications). Those who want to maintain confidentiality and integrity of their communication need a robust and secure mechanism to reliably exchange cryptographic keys. Primarily, there are two methods of exchanging keying messages:

- Integrated keying, through the session establishment protocol, such as SIP[2]. This approach requires fewer messages to be exchanged, and thus minimizes associated delays introduced by message exchange.
- Native key exchange through a distinct process. This approach requires more messages to be generated between end points, and thus increases the risk of associated delays introduced by message exchange. Furthermore, a device cannot determine in advance whether the remote end point can support a particular key-exchange mechanism. For example, Bob sends an INVITE to Alice, but Bob doesn't know whether Alice's device can support MIKEY[3] because the initial exchange of messages does not contain any corresponding information. At this point, Bob can't determine whether his call will be encrypted or there is a delay in setting up the encryption unless his phone has the capability to alert him.

Cryptographic functions are computationally intensive because of the mathematical computations they must perform to derive the corresponding product (for example, Message Authentication Code or cryptographic keys). Therefore, it is important to define a set of requirements when designing key negotiation protocols, particularly when they are used in conjunction with real-time streaming applications that are time sensitive. When designing key-exchange protocols, you must consider the following:

- **Computational resource consumption—**Key negotiation mechanisms are resource intensive and impact both processing and storage resources (that is, CPU, memory), which also consume power (for example, battery life). In the case of multimedia applications such as VoIP, where processing of media streams is also computational intensive, it is critical to maintain stringent requirements for

2. *J. Rosenberg, et al. SIP: Session Initiation Protocol. IETF RFC 3261, June 2002.*
3. *J. Arkko, et. al. MIKEY: Multimedia Internet KEYing. IETF RFC 3830, August 2004.*

low resource consumption, especially for mobile devices such as phones and PDAs. Establish a careful balance between the amount of processing required by the cryptographic functions and the resource capabilities of the respective device. A typical question that helps guide the decision is "how much security does this device provide?"

- **Session establishment delay—**In multimedia applications (and naturally, VoIP), key negotiation adds another layer of messages to be exchanged to establish a secure session between two or more parties. This added layer can introduce delays in establishing a session, which may impact the QoS. Therefore, it is necessary to be as conservative as possible and minimize the number of messages required to negotiate keys.
- **Implosion avoidance—**An important consideration when designing a key management protocol is the case of implosion, where a network element can be overloaded by an overwhelming number of legitimate messages. There are two variations of this condition: out-of-sync and feedback implosion. The out-of-sync implosion refers to the simultaneous attempt of legitimate participants to update their security associations or rekey, which will result in overwhelming the key server with update request messages. The feedback implosion refers to the reliable delivery of rekey messages. Typically, reliable multicast protocols are designed to retransmit packets when packet loss occurs. Therefore, many group members may simultaneously transmit feedback messages (that is, NACK or ACK) to the key server, and thus overwhelm the server.

Currently, there are several existing and emerging key management standards, including MIKEY, and MIKEYv2, SRTP Security Descriptions, ZRTP,[4] and GDOI[5] (group key management) for establishing SRTP cryptographic context. This chapter focuses on MIKEY, ZRTP, and SRTP Security Descriptions because they are currently deployed in VoIP environments and supported by VoIP vendors.

4. *P. Ziemmermann. ZRTP: Extensions to RTP for Diffie-Hellman Key Agreement for SRTP. IETF draft, www.ietf.org/Internet-drafts/draft-zimmermann-avt-zrtp-02.txt.*

5. *M. Baugher, et al. The Group Domain of Interpretation. IETF RFC 3547.*

MIKEY

Internet multimedia applications such as VoIP have demanding performance and QoS requirements. Latency[6] is one[7] property that requires careful consideration to improve or maintain QoS in multimedia communications. Key exchange adds an additional processing burden for edge devices, especially the ones with limited processing power and memory capacity (for example, handhelds). Although memory and processing power have dramatically improved for handheld devices, encryption remains a resource-intensive task that requires consideration when designing protocols. Therefore, MIKEY was developed with the intention to minimize latency when exchanging cryptographic keys between small interactive groups that reside in heterogeneous networks. In addition, the following properties were considered when developing the protocol:

- The protocol should maintain simplicity for ease of implementation, performance, and security.
- Minimize message exchange. The negotiation of key material should be accomplished in one round trip.
- Support secure end-to-end key management.
- Protocol integration. Allow transport of messages within other protocols (that is, SDP).
- Protocol independence. Maintain independence from any security functionality imposed by the underlying transport.
- Low bandwidth consumption and low computational workload.

The MIKEY (Multimedia Internet KEYing) protocol is defined in the IETF RFC 3830 and was developed to support key negotiation for security protocols such as SRTP (Secure Real-time Time Protocol) and IPSec. Although SRTP is currently the only protocol directly supported by MIKEY, IPSec/ESP can also be supported by developing the corresponding profile. The standard describes mechanisms for negotiating keys between two or more parties who want to establish a secure channel of communication. The protocol can be used in the following modes:

6. *Delay of packet delivery. ITU-T G.114 recommends a maximum of a 150ms one-way latency.*
7. *Packet loss and jitter are other factors that impact multimedia communications, and there is always a constant effort for improvement.*

- Peer to peer (unicast)
- Simple one to many (multicast)
- Many to many, without a centralized control unit

An additional mode is supported: many to many, with centralized control (applicable to a larger user group that requires the coordination of key exchange). To transport and exchange keying material, three methods are supported, as follows:

- **Pre-shared secret key (PSK)—**The PSK is used to derive subkeys for encryption and integrity. Although it is not scalable for group communications, it is the most efficient way to handle key transport.
- **Public key encryption (PKE)—**The originating user generates a random encryption key, which is then sent to the remote user using the public key to encrypt it. This method requires somewhat more computational resources as compared to PSK, but it supports key negotiation for group communications and provides better scalability in an environment where a central repository for public keys is available (that is, a PKI infrastructure).
- **Diffie-Hellman (DH) key exchange—**Optional to implement. This method is the most resource intensive, but it is the only one of the three listed here that provides Perfect Forward Secrecy (PFS).[8] This method can be used only for peer-to-peer key negotiation, and it requires the existence of a PKI infrastructure.

As you can understand from the preceding list, vendors that implement MIKEY in their products are required to support pre-shared and public key encryption methods to interoperate with other implementations.

MIKEY Protocol Definitions and Constructs

To understand the operations of the MIKEY protocol, it is necessary to understand some of the abstract protocol constructs. The following definitions represent some of the fundamental constructs used in MIKEY.[9]

8. *In an authenticated key agreement protocol that uses public key cryptography, Perfect Forward Secrecy (PFS) is the property that disclosure of the long-term secret keying material that is used to derive an agreed ephemeral key does not compromise the secrecy of agreed keys from earlier runs (definition provided by wikipedia.org).*

9. *Some of these definitions are originally captured in the Multicast Security (MSEC) Group Key Management Architecture document RFC 4046.*

- **Data security protocol—**The security protocol used to protect the actual data traffic, such as IPSec and SRTP.
- **Data security association (data SA or SA)—**Information for the security protocol, including a TEK and a set of parameters/policies.
- **TEK-generation key (TGK)—**A bit string agreed upon by two or more parties, associated with the crypto session bundle (defined in this list). From the traffic-generation key, traffic-encrypting keys can then be generated without needing further communication.
- **Traffic-encrypting key (TEK)—**The key used by the security protocol to protect the CS. (This key may be used directly by the security protocol or may be used to derive further keys depending on the security protocol.) The TEKs are derived from the CSB's TGK.
- **Crypto session (CS)—**Uni- or bidirectional data stream(s) protected by a single instance of a security protocol. For example, when SRTP is used, the CS will often contain two streams, an RTP stream and the corresponding RTCP, which are both protected by a single SRTP cryptographic context; that is, they share key data and the bulk of security parameters in the SRTP cryptographic context (default behavior in SRTP). In the case of IPSec, a CS would represent an instantiation of an IPSec SA. A CS can be viewed as a data SA (as defined in GKMARCH) and could therefore be mapped to other security protocols if necessary.
- **Crypto session bundle (CSB)—**Collection of one or more CSs, which can have common traffic-generation keys and security parameters.
- **Crypto session ID.** Unique identifier for the CS within a CSB.
- **Crypto session bundle ID (CSB ID)—**Unique identifier for the CSB.

Establishing a Session

Each key-exchange mechanism (PSK, PKE, and Diffie-Hellman) defined in MIKEY is using the same approach of sending and receiving messages, but the message attributes (that is, headers, payloads, and values) differ from method to method. Ultimately, the objective in each method is to transport the appropriate key material and establish a crypto session. The two important pieces of information contained in the initial message, which is originated by the initiator, is the TGKs (one or more) and the security policies associated with the respective CS. The typical attributes of a message include the following:

- **HDR—**The general MIKEY header, which includes MIKEY CSB-related data (for example, CSB ID) and information mapping to the specific security protocol used.
- ***T*—**The timestamp, used mainly to prevent replay attacks.
- **ID*x*—**The identity of entity *x* (IDi = initiator, IDr = responder).
- **RAND—**Random/pseudo-random byte string, which is always included in the first message from the initiator. RAND is used as a freshness value for the key generation. It is not included in update messages of a CSB.
- **SP—**The security policies for the data security protocol.

When PKE exchange is used between two parties, the initiator sends an I_MESSAGE request, which carries the KEMAC and the desired SP. Figure 7.1 demonstrates this message exchange.

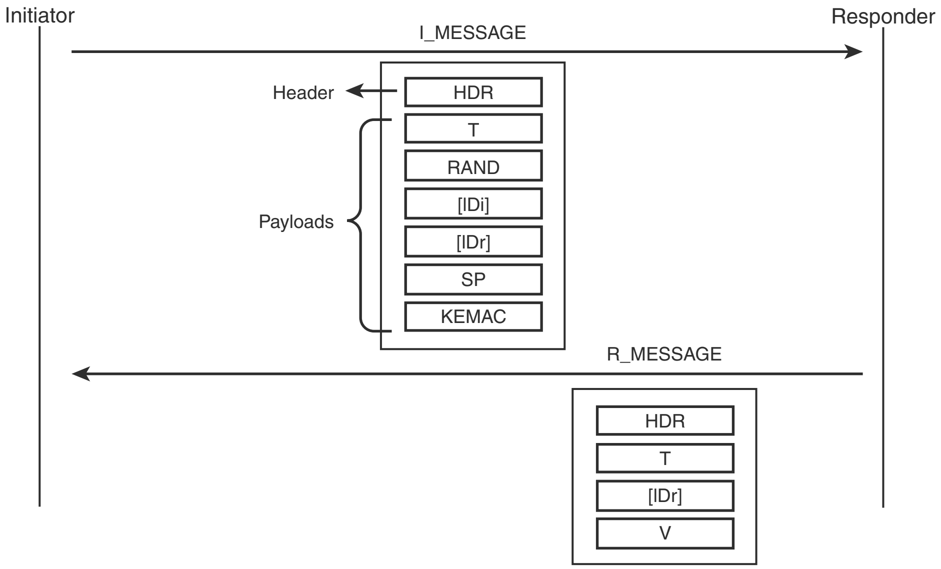


Figure 7.1 PSK message exchange in MIKEY.

The main objective of the initiator's message is to transport one or more TGKs and security parameters to the responder in a secure manner. In this case, the KEMAC is computed by encrypting all the TGK and using the predetermined MAC algorithm to provide integrity. The computation is as follows:

```
KEMAC = E(encr_key, {TGK}) || MAC
```

The main objective of the verification message (in the R_MESSAGE) from the responder is to obtain mutual authentication. The verification message, V, is a MAC that is computed over the responder's entire message, the timestamp (the same as the one that was included in the initiator's message), and the two parties' identities, using the authentication key. In the case where PKE is used, the initiator's message contains three additional payloads to support associated certificate information: CHASH, PKE, and SIGNi. The responder's message is the same as in the previous case. Figure 7.2 depicts the message structure when PKE is used.

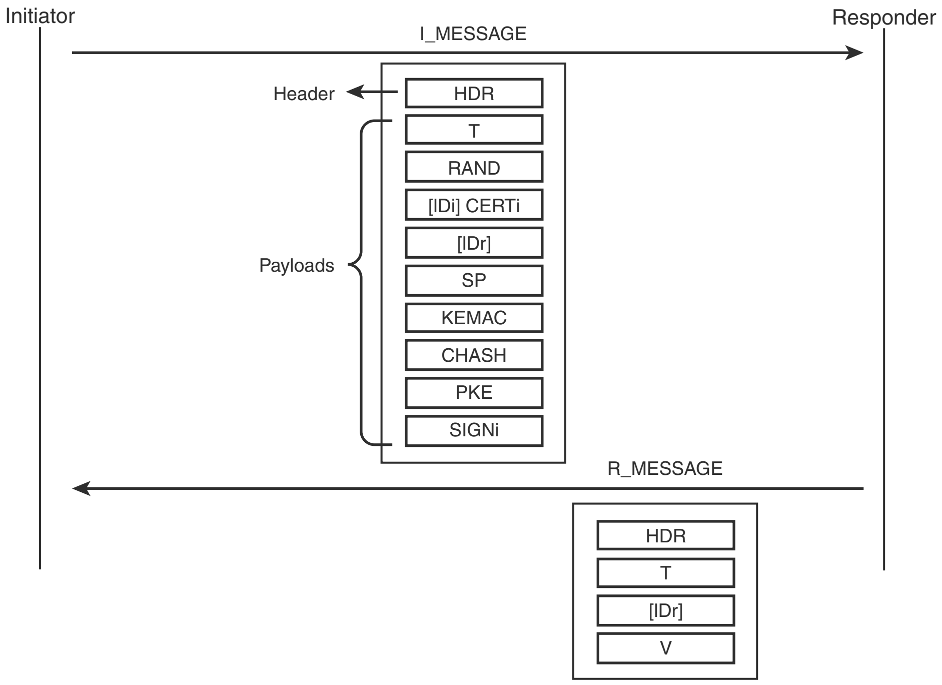


FIGURE 7.2 PKE message in MIKEY.

In this instance, the KEMAC is computed using the following:

```
KEMAC = E(encr_key, IDi || {TGK}) || MAC
```

The KMAC contains a set of encrypted subpayloads and a MAC, as described earlier with regard to the PSK exchange, but the encrypted payload contains the TGK and the identity of the initiator IDi. The IDi does not represent a certificate, but it can be the same ID as the one in the initiator's certificate. Finally, in the Diffie-Hellman exchange, the payloads for KEYMAC, CHASH, and PKE are not used, but a payload DHi that holds the initiator's Diffie-Hellman information is introduced. The main objective of the initiator's message is to communicate securely the security protocol parameters and provide the responder with its DH value (DHi) g^{xi}, where xi *must* be pseudo-randomly and secretly chosen.

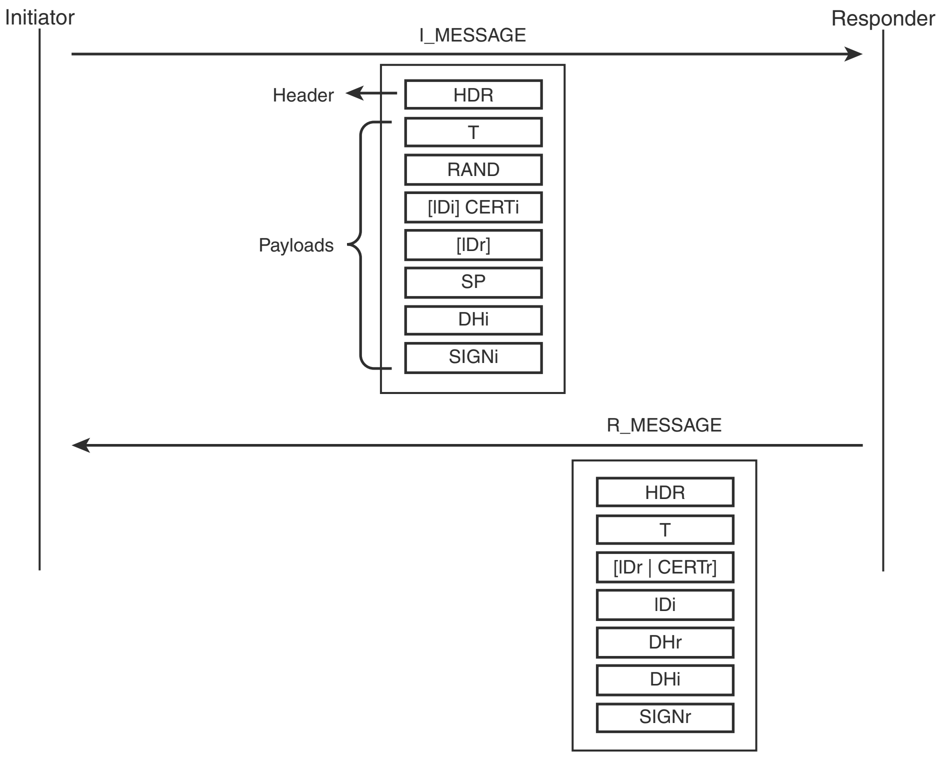


FIGURE 7.3 MIKEY Diffie-Hellman exchange.

In this case, the responder's message is very different compared to the other two response messages. Specifically, the message includes a payload that captures the responder's ID and certificate information, the initiator's ID, and the DH values for the initiator and receiver. Both parties calculate the TGK, g(xi×xr) from the exchanged DH values. The SIGNr is a signature covering the responder's MIKEY message, R_MESSAGE, using the responder's signature key.

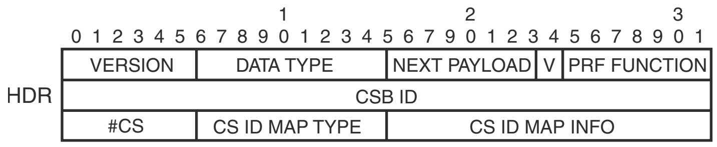
Protocol Syntax and Message Creation

Creating a MIKEY message consists of the following steps:

1. Create an initial MIKEY message starting with the Common Header payload.
2. Concatenate necessary payloads of the MIKEY message.
3. As a last step (for messages that must be authenticated, this also includes the verification message), create and concatenate the MAC/signature payload without the MAC/signature field filled in. (If a NEXT PAYLOAD field is included in this payload, it is set to LAST PAYLOAD.)
4. Calculate the MAC/signature over the entire MIKEY message, except the MAC/Signature field, and add the MAC/signature in the field. In the case of the verification message, the Identity_i || Identity_r || Timestamp *must* directly follow the MIKEY message in the Verification MAC calculation. Note that the added identities and timestamp are identical to those transported in the ID and T payloads.

The common header payload must be included at the beginning of each MIKEY message (request and response) because it provides necessary information about the CS and CSB with which it is associated. Figure 7.4 depicts the fields that comprise the header.

MIKEY defines several payloads to support the three key exchange methods and the corresponding architectural scenarios (that is, peer to peer, simple one to many [multicast] many to many, without a centralized control unit), as follows:



VERSION	8 – bits, the version number of MIKEY, currently 0x01 as defined in RFC 3830
DATA TYPE	8 – bits, describes the type of message (e.g. public key transport, verification, error message)
NEXT PAYLOAD	8 – bits, indentifies the payload that is added after this payload
V	1 – bit, flag to indicate whether a verification message is expected or not. Typically this is set by the initiator of a message only.
PRF FUNCTION	7 – bits, indicates the PRF function that has been (or will be) used for key derivation
CSB ID	32 – bits, identifies the CSB
#CS	8 – bits, indicates the number of Crypto Sessions that will be handled within the CSB. Although it is possible to have 255 CS's it is not likely that will occur in a single CSB. The number 0 indicates that no CS is included.
CS ID MAP TYPE	8 – bits, specifies the method of uniquely mapping Crypto Sessions to the security protocol sessions
CS ID MAP INFO	16 – bits, identifies the crypto session(s) for which the SA should be created. Currently the defined map is SRTP-ID.

FIGURE 7.4 MIKEY header.

- **Key data transport payload (KEMAC)—**Contains encrypted key data subpayloads.
- **Envelope data payload (PKE)—**Contains the encrypted envelope key that is used in the public key transport to protect the data in the key data transport payload.
- **DH data payload (DH)—**Contains the DH value and indicates the DH group used.
- **Signature payload (SIGN)—**Contains the signature and its related data.
- **Timestamp payload (T)—**Carries the timestamp information.
- **ID payload (ID).** The ID payload carries a uniquely defined identifier.
- **Certificate payload (CERT)—**The certificate payload contains an indicator of the certificate provided as well as the certificate data.
- **Cert hash payload (CHASH)—**The Cert hash payload contains the hash of the certificate used.

- **Ver msg payload (V)—**The Ver msg payload contains the calculated verification message in the pre-shared key and the public key transport methods.
- **Security policy payload (SP)—**The security policy payload defines a set of policies that apply to a specific security protocol.
- **SRTP policy.** This field specifies the parameters for SRTP and SRTCP.
- **RAND payload (RAND)—**The RAND payload consists of a (pseudo-)random bit string.
- **Error payload (ERR)—**The error payload is used to specify the error(s) that may have occurred.
- **Key data subpayload—**The key data payload contains key material (for example, TGKs).
- **Key validity data—**The key validity data is not a standalone payload, but part of either the key data payload or the DH payload.
- **General extensions payload—**The general extensions payload is included to allow possible extensions to MIKEY without the need for defining a completely new payload each time.

MIKEY messages can be transported using various signaling protocols including SIP, RTSP, and H.323.

Generating a Crypto Session

MIKEY provides the ability to support multiple crypto sessions for several security protocols or multiple instances of the same security protocol. This notion is represented using a CSB. Figure 7.5 shows a logical representation of the process for establishing a crypto session.

The CSB maintains the traffic-generation key and the security policies associated with each CS. The CSB can facilitate the management of one or more crypto sessions, which in turn represent a distinct communication channel (for example, a phone call, file transfer, video stream). It should be noted that the CSs in a CSB can use the same traffic-generation key mechanism, but each CS inherits a distinct TEK. Each CS is applied to the corresponding data stream through the associated security protocol (that is, SRTP).

The data security association (data SA) is used by the corresponding security protocol (that is, SRTP). The information within the data SA includes the parameters and policies to be used with the corresponding security protocol (that is, encryption algorithms, encryption key size, lifetime of keys, and so on) and a TEK. The TEK can also be used by the respective security protocol to derive additional keys. Figure 7.6 shows a logical representation for the key-derivation process.

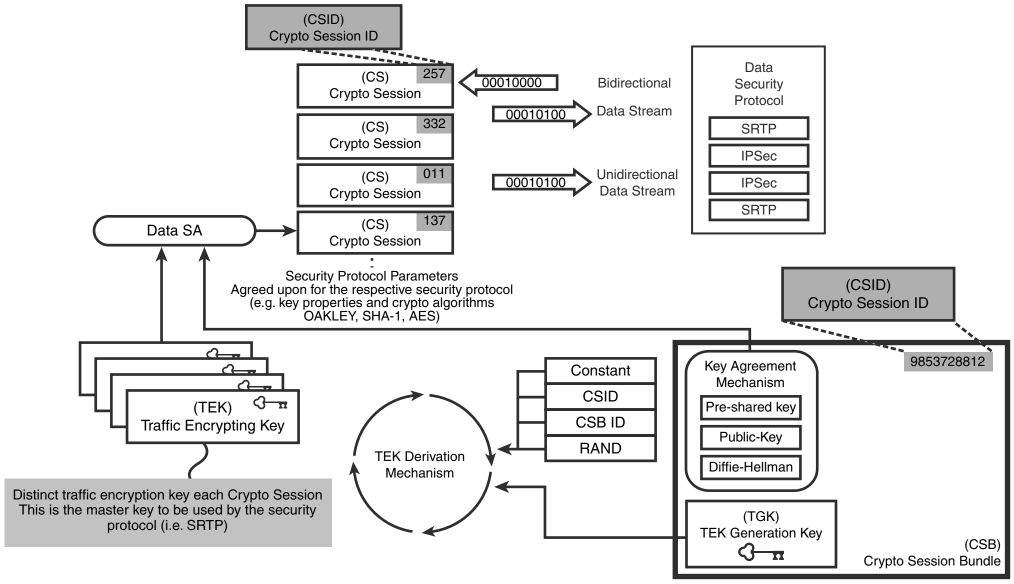


FIGURE 7.5 MIKEY creation of a crypto session.

The TEK is generated using a pseudo-random function (PRF) with the following as input (|| indicates concatenation):

```
inkey : TGK
inkey_len : bit length of TGK
label : constant || cs_id || csb_id || RAND
outkey_len : bit length of the output key
```

The 32-bit *constant* values are taken from the decimal digits of number *e* (2. 718281828…) in consecutive chunks, where each constant consists of nine decimal digits (for example, the first nine decimal digits 718281828 = 0x2AD01C64). The cs_id is an 8-bit unsigned integer of the corresponding CS. Similarly, the csb_id is a 32-bit unsigned integer of the corresponding CSB. Finally, RAND is a 128-bit pseudo-random sent by the initiator in the initial message exchange.

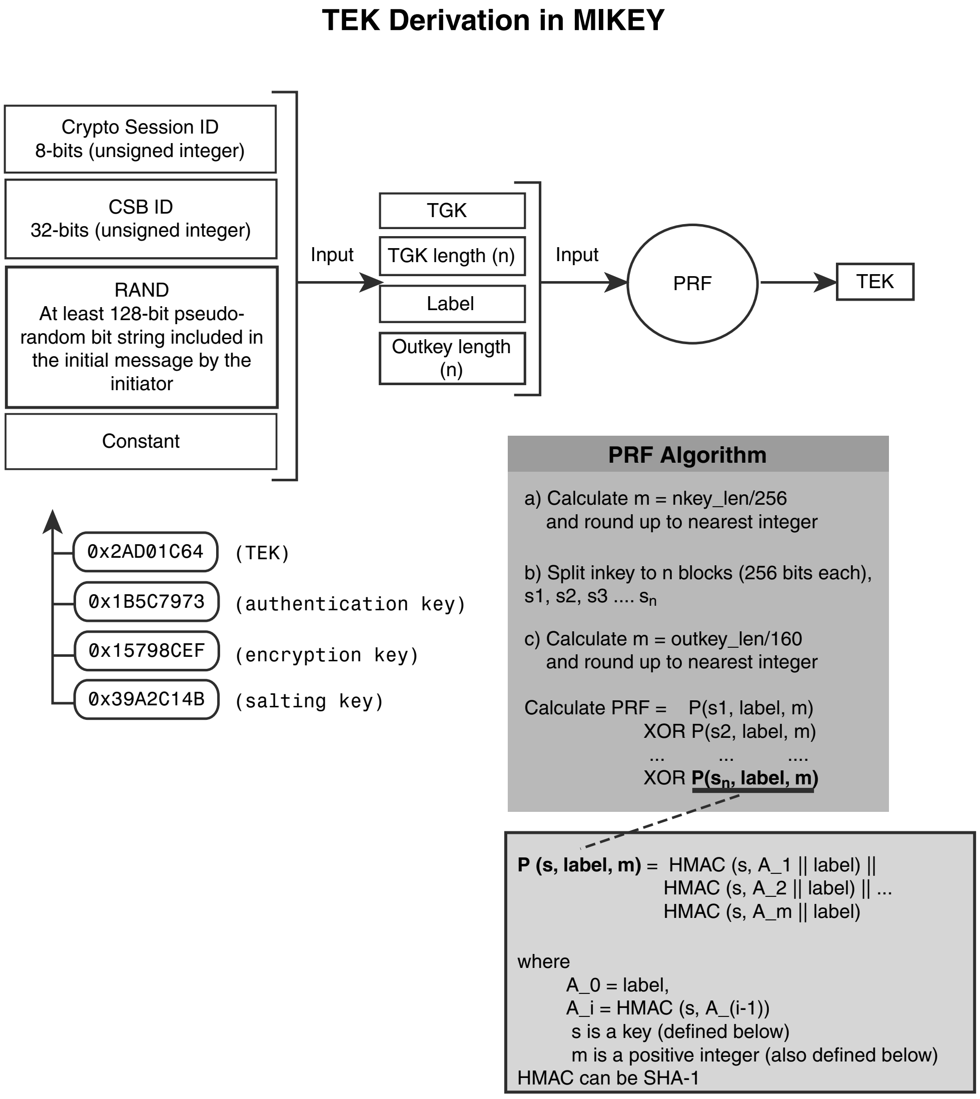

TEK Derivation in MIKEY
Crypto Session ID 8-bits (unsigned integer)
CSB ID 32-bits (unsigned integer)
RAND At least 128-bit pseudo-random bit string included in the initial message by the initiator
Constant
Input
TGK
TGK length (n)
Label
Outkey length (n)
Input
PRF
TEK
0x2AD01C64 (TEK)
0x1B5C7973 (authentication key)
0x15798CEF (encryption key)
0x39A2C14B (salting key)
PRF Algorithm
a) Calculate m = nkey_len/256 and round up to nearest integer
b) Split inkey to n blocks (256 bits each), s1, s2, s3 .... sn
c) Calculate m = outkey_len/160 and round up to nearest integer
Calculate PRF = P(s1, label, m) XOR P(s2, label, m) ... ... XOR P(sn, label, m)
P (s, label, m) = HMAC (s, A_1 || label) || HMAC (s, A_2 || label) || ... HMAC (s, A_m || label)
where
A_0 = label,
A_i = HMAC (s, A_(i-1))
s is a key (defined below)
m is a positive integer (also defined below)
HMAC can be SHA-1

FIGURE 7.6 MIKEY TEK derivation.

Using MIKEY with SIP

MIKEY messages can be exchanged through the signaling protocol that the multimedia application is using. This section describes how MIKEY messages are exchanged using SIP/SDP, but similar approaches are used by signaling protocols such as H.235 and RTSP.

Integrating MIKEY messages within SIP minimizes the number of messages sent between end points to exchange keys. In other words, end points are not required to send separate MIKEY messages and SIP messages to establish a secure session. Therefore, it is desirable to integrate MIKEY messages within the application protocol.

Figure 7.7 shows how MIKEY key exchange is performed using SIP. Bob sends an INVITE to Alice that contains the MIKEY initiator message (I_MESSAGE). If Alice answers the phone, a 200 OK response will be sent back to Bob's phone that contains a MIKEY responder message (R_MESSAGE). Note that the MIKEY R_MESSAGE is not sent in the provisional response 180 Ringing to avoid performing the key exchange prematurely and thus executing cryptographic computations unnecessarily in case the called user does not respond. If the MIKEY R_MESSAGE is included in the 180 Ringing response, an attacker can take advantage of this configuration to perform various DoS attacks.

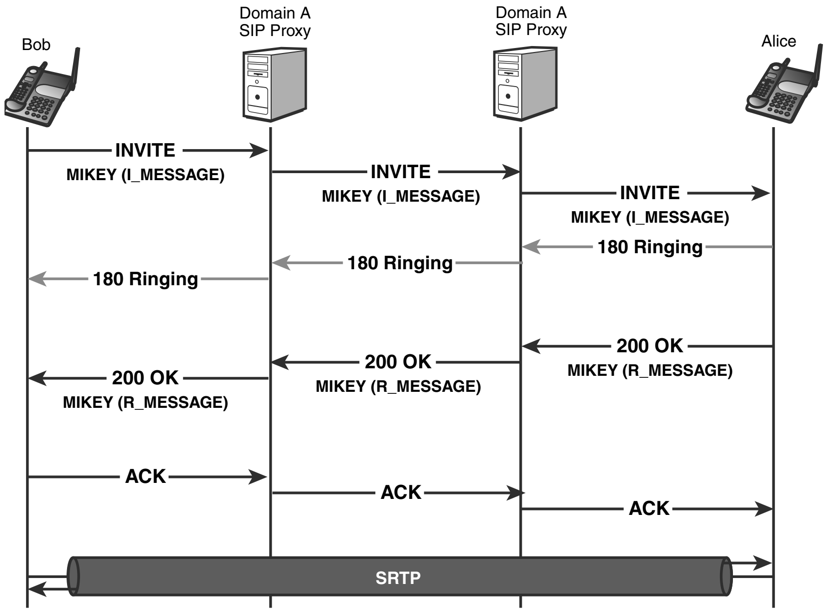
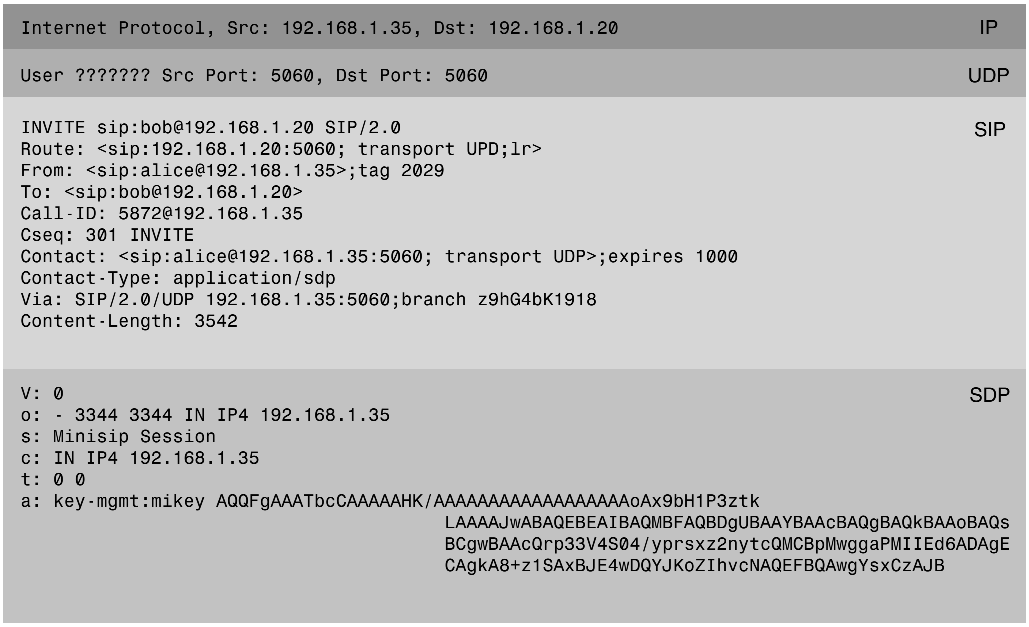


FIGURE 7.7 MIKEY key exchange with SIP.

When Bob's phone receives the 200 OK response, it sends an ACK to Alice's phone and prepares to initiate the media exchange. Each device derives a TEK for the corresponding session based on the key that was negotiated during the session setup. The TEK key is used with SRTP to protect the media. Figure 7.8 illustrates the use of a MIKEY initiation message in a SIP INVITE.



```
Internet Protocol, Src: 192.168.1.35, Dst: 192.168.1.20                                IP

User ??????? Src Port: 5060, Dst Port: 5060                                            UDP

INVITE sip:bob@192.168.1.20 SIP/2.0                                                    SIP
Route: <sip:192.168.1.20:5060; transport UPD;lr>
From: <sip:alice@192.168.1.35>;tag 2029
To: <sip:bob@192.168.1.20>
Call-ID: 5872@192.168.1.35
Cseq: 301 INVITE
Contact: <sip:alice@192.168.1.35:5060; transport UDP>;expires 1000
Contact-Type: application/sdp
Via: SIP/2.0/UDP 192.168.1.35:5060;branch z9hG4bK1918
Content-Length: 3542

V: 0                                                                                   SDP
o: - 3344 3344 IN IP4 192.168.1.35
s: Minisip Session
c: IN IP4 192.168.1.35
t: 0 0
a: key-mgmt:mikey AQQFgAAATbcCAAAAAHK/AAAAAAAAAAAAAAAAAAoAx9bH1P3ztk
                                        LAAAAJwABAQEBEAIBAQMBFAQBDgUBAAYBAAcBAQgBAQkBAAoBAQs
                                        BCgwBAAcQrp33V4S04/yprsxz2nytcQMCBpMwggaPMIIEd6ADAgE
                                        CAgkA8+z1SAxBJE4wDQYJKoZIhvcNAQEFBQAwgYsxCzAJB
```

Figure 7.8 Using MIKEY with SIP.

The MIKEY parameters are captured in the SDP portion of the SIP INVITE using the *key-mgmt* attribute.

According to RFC 3830, "MIKEY is mainly intended to be used for peer-to-peer, simple one-to-many, and small size (interactive) groups." Therefore, one area that needs to be addressed is whether MIKEY can support key distribution in large groups that require multimedia services (that is, video multicasting for millions of subscribers). This is a theoretical limitation because there haven't been any substantial case studies that use MIKEY for communications in large distributed groups.

Another area that requires attention when implementing MIKEY is selecting the appropriate transport protocol to be used: TCP versus UDP. RFC 3261 mandates that SIP messages that are larger than 1300 bytes must be transmitted using congestion-controlled transport such as TCP. Therefore, in cases where MIKEY requires the exchange of PKI certificates and the use of Diffie-Hellman, TLS must be used. On the other hand, use of TCP impacts the performance of setting up a call that traverses multiple hops because TLS operates over TCP and TCP connections require more messages to set up compared to UDP (for example, three-way TCP handshake versus single UDP packet). If the TLS sessions has already been established, there is no impact.

Regardless of what transport protocol is used to exchange MIKEY messages, it has to provide confidentiality and integrity to protect the keys from being intercepted by an unauthorized party. Therefore, most implementations use TLS, with some experimental adoption of DTLS.

SRTP Security Descriptions

SRTP Security Descriptions[10] is not considered a key management protocol such as MIKEY but rather a mechanism to negotiate cryptographic keys among users in unicast sessions using the SRTP transport (for example, RTP/SAVP or RTP/SAVPF). The Security Descriptions mechanism does not support multicast media streams or multipoint unicast streams.

To communicate the keying material, the *crypto* field is used within SDP (Session Description Protocol). Figure 7.9 shows where the crypto attribute is defined in the SDP portion of a SIP INVITE message.

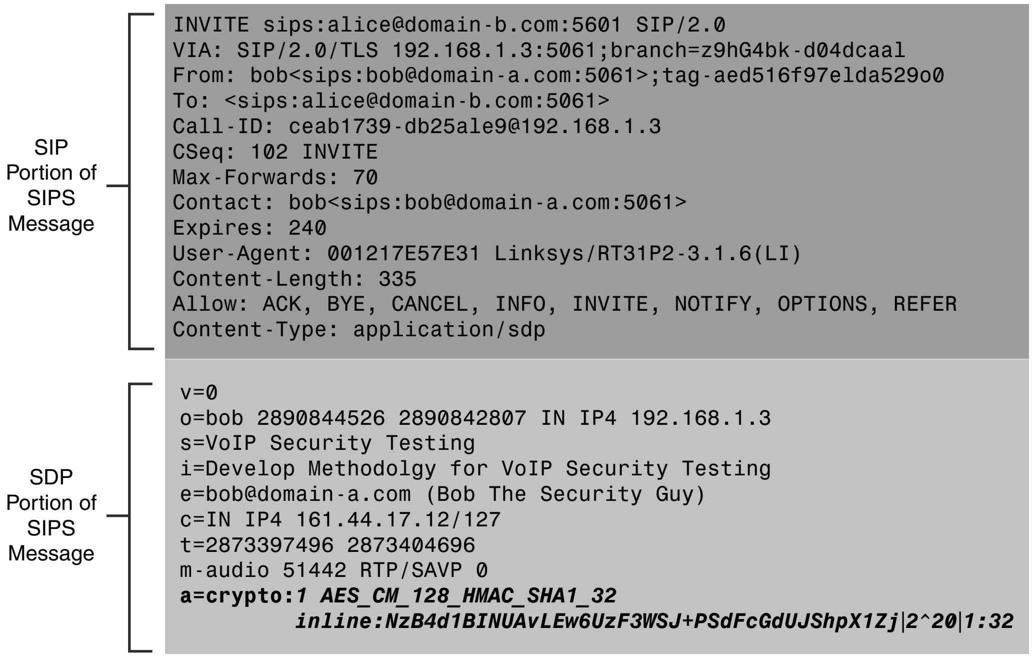
The format of the *crypto* attribute is as follows:

```
a=crypto:<tag> <crypto-suite> <key-params> [<session-params>]
```

In Figure 7.9, the *crypto suite* is AES_CM_128_HMAC_SHA1_32, *key params* is defined by the text starting with "inline:", and *session params* is implementation dependent.

The *tag* field is a decimal number and is used as part of the offer/answer model to distinguish the crypto attributes chosen by the participants for each media stream in a session. For example, Alice may offer two or more crypto suites to Bob during the initial offer (for example, AES_CM_128_HMAC_SHA1_80, AES_CM_128_HMAC_SHA1_32, and f8_128_HMAC_SHA1_80). Bob can respond to Alice by selecting the option f8_128_HMAC_SHA1_80 as the cryptographic transformation to protect the respective media stream.

10. *F. Andreasen, M. Baugher, and D. Wing. Session Description Protocol Security Descriptions for Media Streams. IETF RFC 4568.*



```
INVITE sips:alice@domain-b.com:5601 SIP/2.0
VIA: SIP/2.0/TLS 192.168.1.3:5061;branch=z9hG4bk-d04dcaal
From: bob<sips:bob@domain-a.com:5061>;tag-aed516f97elda529o0
To: <sips:alice@domain-b.com:5061>
Call-ID: ceab1739-db25ale9@192.168.1.3
CSeq: 102 INVITE
Max-Forwards: 70
Contact: bob<sips:bob@domain-a.com:5061>
Expires: 240
User-Agent: 001217E57E31 Linksys/RT31P2-3.1.6(LI)
Content-Length: 335
Allow: ACK, BYE, CANCEL, INFO, INVITE, NOTIFY, OPTIONS, REFER
Content-Type: application/sdp
```

```
v=0
o=bob 2890844526 2890842807 IN IP4 192.168.1.3
s=VoIP Security Testing
i=Develop Methodolgy for VoIP Security Testing
e=bob@domain-a.com (Bob The Security Guy)
c=IN IP4 161.44.17.12/127
t=2873397496 2873404696
m-audio 51442 RTP/SAVP 0
a=crypto:1 AES_CM_128_HMAC_SHA1_32
         inline:NzB4d1BINUAvLEw6UzF3WSJ+PSdFcGdUJShpX1Zj|2^20|1:32
```

Figure 7.9 SIP and SDescriptions.

The *crypto-suite* field is an identifier that describes the encryption and authentication algorithms (for example, AES_CM_128_HMAC_SHA1_80).

The *key-params* field provides one or more sets of keying material for the *crypto-suite* and consists of a method, in this case "inline," which indicates that the actual keying material (master key and salt) is provided in the *key-info* field itself. Additional information includes the associated policy of the master key such as its lifetime and use of MKI (master key identifier). The MKI is used to associate SRTP packets with a master key in a multimedia session. Based on the IETF Security Descriptions standard, each key follows this format:

```
"inline:" <key||salt> ["|" lifetime] ["|" MKI ":" length]
```

The syntax of the key is as follows:

- key||salt is the concatenated master key and salt encoded in base64 format.
- lifetime indicates the lifetime of the master key.
- MKI:length: indicates the MKI and length of the MKI field in SRTP packets.

The lifetime and MKI parameters may not be present in some implementations because they are defined as *optional* by the standard. Figure 7.10 shows an example of a key without lifetime or MKI values.

```
Media Attribute (a): crypto: 1 AES_CM_128_HMAC_SHA1_32 inline:UlrbLlfNTNw3blKHQVLGze6oHsyFdjGj3NheKoYx

Media Attribute (a): crypto:

    1 ------------------------------------------------> Tag
    AES_CM_128_HMAC_SHA1_32 -------------------------> Crypto-suite
    inline: ------------------------------------------> Method
    UlrbLlfNTNw3blKHQVLGze6oHsyFdjGj3NheKoYx ---> Master key with salt encoded using base 64
```

FIGURE 7.10 Security Descriptions without lifetime or MKI values.

Figure 7.11 displays the case where the lifetime attribute and MKI are present.

```
Media Attribute (a): crypto:
    1 ------------------------------------------------> Tag
    AES_CM_128_HMAC_SHA1_32 -------------------------> Crypto-suite
    inline: ------------------------------------------> Method
    UlrbLlfNTNw3blKHQVLGze6oHsyFdjGj3NheKoYx ---> Master key with salt encoded using  base 64
    |2^20 --------------------------------------------> Master key lifetime (optional)
    |1:4   -------------------------------------------> Master Key Identifier (optional)
```

FIGURE 7.11 Security Descriptions with lifetime and MKI values.

The notation |2^20 (for example, 2 to the power of 20) indicates the lifetime value of the master key measured in packets (for example, the maximum number of SRTP packets that should be encrypted using this particular key).

The notation |1:4 indicates the MKI and its length. This parameter is also optional. The identifier is 1 (one) and its length is 4 bytes long. Another example is this:

```
inline: UlrbLlfNTNw3blKHQVLGze6oHsyFdjGj3NheKoYx |1024:4
```

where the key identifier is 1024 with a length of 4 bytes.

The session parameters [<session-params>] that can be included in an offer/answer interaction are as follows (as defined by the RFC):

- **KDR—**The SRTP key-derivation rate is the rate that a PRF is applied to a master key.
- **UNENCRYPTED_SRTP—**SRTP messages are not encrypted.
- **UNENCRYPTED_SRTCP—**SRTCP messages are not encrypted.
- **UNAUTHENTICATED_SRTP—**SRTP messages are not authenticated.
- **FEC_ORDER—**Order of forward error correction (FEC) relative to SRTP services.
- **FEC_KEY—**Master key for FEC when the FEC stream is sent to a separate address/port.
- **WSH—**Window size hint, which is used to protect against replay attacks.
- **Extensions—**Extension parameters can be defined.

Note that Security Descriptions are defined within SDP, which is typically encapsulated in protocols such as SIP or MGCP. Therefore, it is expected that the underling transport protocol (for example, TLS, IPSec) will provide authentication and confidentiality to protect the keying material from attacks such as eavesdropping, replaying, and message modification.

ZRTP

ZRTP is another key agreement protocol that can be used to support SRTP. The fundamental difference between ZRTP and other existing key-exchange mechanisms is that cryptographic keys are negotiated through the media stream (RTP) over the same UDP port instead of using the signaling path as it is done with MIKEY or SDescriptions. Therefore, the key negotiation is performed directly between peers without requiring the use of intermediaries such as SIP proxies to relay the keying material. If necessary, however, the ZRTP design also provides the option to exchange keying material through signaling messages. Primarily, the protocol uses ephemeral DH (Diffie-Hellman) keys to establish a shared secret between peers, but it does not require a PKI, which makes the protocol an attractive alternative for organizations that do not maintain a PKI. As of this writing, ZRTP is labeled "draft," but it is expected to be ratified as an RFC by the IETF because it has been implemented by vendors.

ZRTP Key Negotiation

Key negotiation in ZRTP is performed using the media path (RTP), and there are two key agreement modes: Diffie-Hellman and pre-shared secret. When Diffie-Hellman mode is used, the key agreement process is performed using five steps to announce support for ZRTP between peers and initiate, manage, and terminate the key exchange, as shown in Figure 7.12.

In pre-shared mode, the end points omit the DH calculation because it is assumed that the shared secret is known from a previous session, but the DHPart1 and DHPart2 messages are still exchanged to determine which shared keys should be used. Instead of DH values (hvi and pvr), the end points use nonces along with the retained secret keys to derive the key material.

In Step 1a, Bob's phone sends a ZRTP Hello message that contains a ZRTP ID (ZID) value, the protocol version, and options to be used with ZRTP. The ZID is a 96-bit random string generated one time during installation of the software shim that implements ZRTP. The options include a hash, cipher, authentication method and tag length, key agreement type, and supported algorithms for SAS (Short Authentication String).

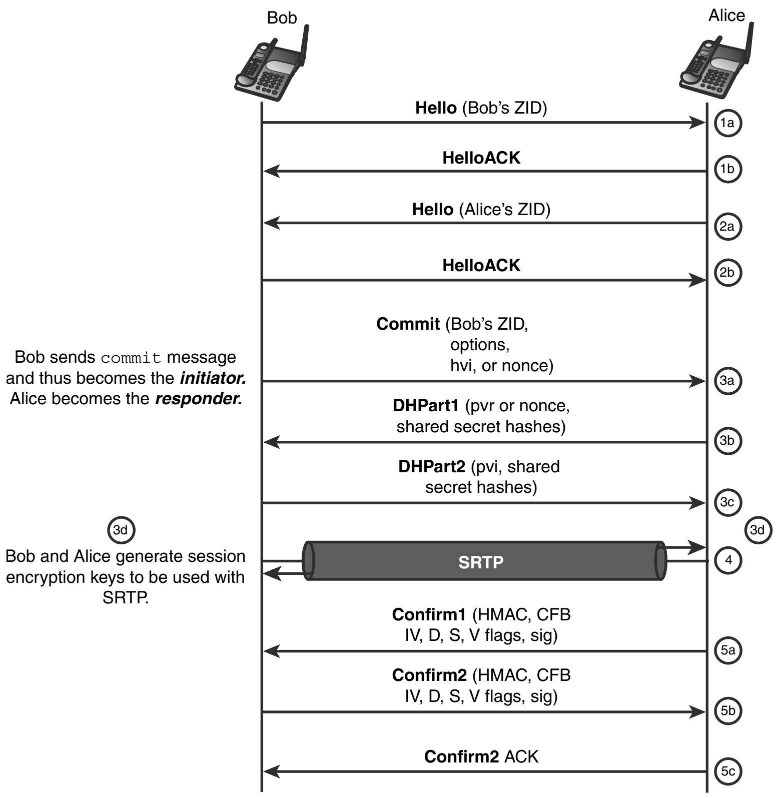

Bob
Alice
Hello (Bob's ZID)
1a
HelloACK
1b
Hello (Alice's ZID)
2a
HelloACK
2b
Commit (Bob's ZID, options, hvi, or nonce)
3a
Bob sends commit message and thus becomes the initiator. Alice becomes the responder.
DHPart1 (pvr or nonce, shared secret hashes)
3b
DHPart2 (pvi, shared secret hashes)
3c
3d
Bob and Alice generate session encryption keys to be used with SRTP.
SRTP
4
Confirm1 (HMAC, CFB IV, D, S, V flags, sig)
5a
Confirm2 (HMAC, CFB IV, D, S, V flags, sig)
5b
Confirm2 ACK
5c

Figure 7.12 ZRTP key negotiation using Diffie-Hellman mode.

This initial message (Hello) is used to verify whether the remote end point supports ZRTP and announce the encryption algorithms that can be supported by the callee. The ZID is a unique identifier generated during installation of the ZRTP, and it is used to index cached shared secrets that have been accumulated from previous sessions and identify the corresponding end point's shared secret. This minimizes the need for additional key renegotiation if the key is already known by the end points.

In Step 1b, Alice sends a response to Bob acknowledging his initial ZRTP Hello message. This indicates to Bob's phone that Alice's phone supports ZRTP and announces her ZID if it is not known from a previous session. The HelloACK message can be omitted in cases where an end point wants to enter the negotiation mode immediately, and the commit message is sent instead.

In Step 2a, Alice sends a Hello message similar to Bob's, who in turn responds with a HelloACK (Step 2b).

In Step 3a, the end points can begin the key agreement when the exchange of initial Hello and HelloACK messages is complete. The first party who sends the commit message is considered the *initiator*; the other party becomes the *responder.* If both parties send the commit message simultaneously, the party that generated the highest hvi (hash value) assumes the initiator role. The hvi is computed as hvi = hash(pvi | responder's Hello message), and the pvi (DH public value) is computed as pvi = g^{svi} mod p. The svi (secret value) is a randomly generated string that is used as the exponent of base g (a number based on Diffie-Hellman cyclic group G). In addition to the ZID and hvi value, the commit message contains a set of options that consists of the ZRTP mode, hash value, cipher, at, keya, and SAS type.

In Step 3b, Alice (responder) sends a DHPart1 message to Bob. The message contains a pvr and shared secret hashes (HMACs) that were used in generating the ZRTP secret. There are five HMAC parameters: rs1IDr, rs2IDr, sigsIDr, srtpsIDr, and other_secretIDr.

In Step 3c, Bob sends a DHPart2 message that contains his public DH value and the calculated secret IDs, similar to DHPart1.

In Step 3d, each participant generates the SRTP master key and master salt using the exchanged shared secret. Note that there are two RTP streams in session, one from Bob to Alice and one from Alice to Bob. Therefore, each RTP stream is using different RTP keys and salts. Each end uses the srtpkey(i/r) and srtpsalt(i/r) to encrypt and decrypt the corresponding RTP stream.

In Steps 5a and 5b, the two end points exchange information about the shared secret key's life expectancy (cache expiration interval) using the confirm message. This message is sent only in response to a valid DHPart2 message when the key negotiation has been completed successfully. Part of the confirm message is encrypted using CFB (Cipher Feedback encryption mode) and protected for integrity using HMAC.

In Step 5c the Conf2ACK message is sent upon receipt of a valid Confirm2, and it is used to stop further retransmission of a Confirm2 message.

To terminate encrypting media, the GoClear message is used. The message does not terminate the session, but alters the state of the RTP stream from being encrypted to unencrypted.

Table 7.1 provides a description of the ZRTP header fields.

Table 7-1 ZRTP Key Negotiation Parameter Mapping

Variable	Description	Comments
ZID	Unique identifier of ZRTP end point	96-bit-long random string generated during initial installation.
hvi/r	Hash value initiator/responder	Computed as hvi = hash (pvi | responder's Hello message).
pvi/r	Public value initiator/responder	Computed as pvi = g^{svi} mod p (initiator). Computed as pvr = g^{svr} mod p (responder).
svi/r	Secret value initiator/responder	Random Diffie-Hellman value based on DH-4096 or DH-3072. The svi value is twice as long as the AES key length. For example, if AES key is 128 bits, svi should be 256 bits.
hash	Supported hash type block	S256; SHA-256 is the only one currently supported.
cipher	Supported cipher types	AES1; AES-CM with 128-bit keys, as defined in RFC 3711.
		AES2; AES-CM with 256-bit keys, as defined in RFC 3711.
at	Authentication tag	HS32; HMAC-SHA1 32-bit authentication tag, as defined in RFC 3711.
		HS80; HMAC-SHA1 80-bit authentication tag, as defined in RFC 3711.
keya	Key agreement types	DH3k; DH mode with p=3072-bit prime, as defined in RFC 3526.
		DH4k; DH mode with p=4096-bit prime, as defined in RFC 3526.
		Prsh; Pre-shared non-Diffie-Hellman mode uses shared secrets.
SAS	SAS type	B32; Short Authentication String using Base32 encoding.
		B256; Short Authentication String using Base256 encoding.
		The SAS value is calculated as the hash of the ZRTP messages (responder's Hello, commit, DHPart1, and DHPart2).

Variable	Description	Comments
rs1IDi/r	Retained secret ID	Computed as rs1IDi = HMAC(rs1, "Initiator"). Computed as rs1IDr = HMAC(rs1, "Responder").
rs2IDi/r	Retained secret ID	Computed as rs2IDi = HMAC(rs2, "Initiator"). Computed as rs2IDr = HMAC(rs2, "Responder").
sigsIDi/r	Signaling secret	The HMAC of the initiator's/responder's signaling shared secret. These values are exchanged using the signaling protocol (for example, SIP) and passed to ZRTP. Computed as sigsIDi = HMAC(sigs, "Initiator"). Computed as sigsIDr = HMAC(sigs, "Responder")
srtpsIDi/r	SRTP secret ID	The HMAC of the initiator's/responder's SRTP secret. Computed as srtpsIDi = HMAC(srtps, "Initiator"). Computed as srtpsIDr = HMAC(srtps, "Responder").
other_secretIDi/r	Other secret	HMAC of an additional shared secret in case multiple shared secrets are available. Computed as other_secretIDi = HMAC(other_secret, "Initiator"). Computed as other_secretIDr = HMAC(other_secret, "Responder").

srtpkeyi/r	SRTP key	The ZRTP initiator and responder generate this value using the following:
		srtpkeyi = HMAC(s0,"Initiator ZRTP key")
		srtpkeyr = HMAC(s0,"Responder SRTP master key")

srtpsalti/r	SRTP salt	The ZRTP initiator and responder generate this value using the following: rtpsalti = HMAC(s0,"Initiator HMAC key") rtpsaltr HMAC(s0,"Responder HMAC key")

(continues)

Table 7-1 ZRTP Key Negotiation Parameter Mapping *(continued)*

Variable	Description	Comments

hmackeyi/r	HMAC key	This value is used only with ZRTP but not SRTP. The ZRTP initiator and responder generate this value using the following:
		hmackeyi = HMAC(s0,"Initiator HMAC key")
		hmackeyr = HMAC(s0,"Responder HMAC key")
		This HMAC key is used to ensure that GoClear messages are unique and cannot be replayed by an attacker to force a connection to go in to unencrypted mode.

Using Zfone

The initial implementation of ZRTP is Zfone, which interfaces with existing soft phones such as X-Lite, Gizmo, and SJphone, but vendors have started to support the protocol; therefore, it is expected that it will gain additional acceptance. Figure 7.13 depicts Zfone with X-Lite.

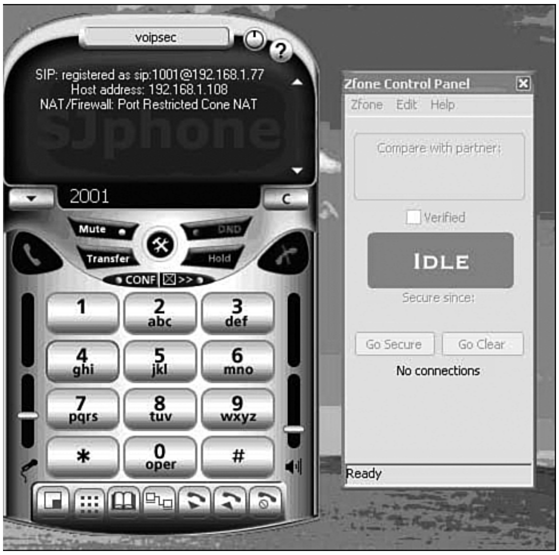


Figure 7.13 ZRTP interface.

During key negotiation, Zfone displays a message to the user indicating its current state, as shown in Figure 7.14.

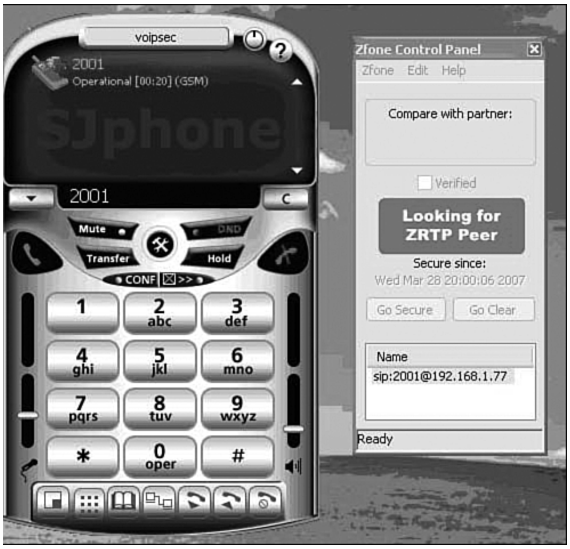
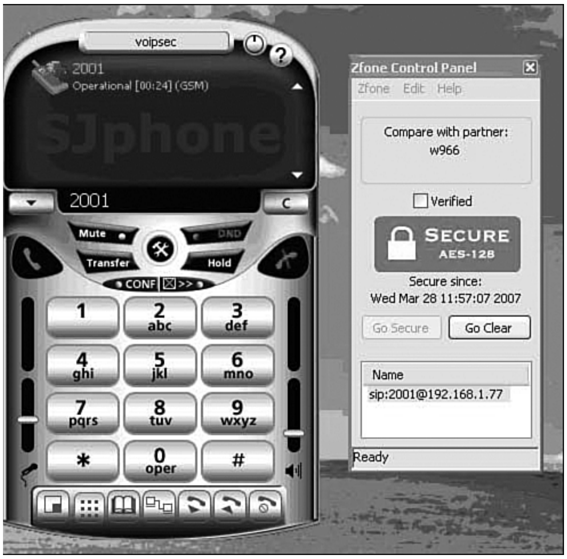


FIGURE 7.14 ZRTP key negotiation state indicator.

When the two parties establish the key exchange, their session is encrypted, as shown in Figure 7.15.

The ZRTP key exchange helps alleviate many of the complexities found in other key-exchange protocols that require the use of signaling messages, but it has its limitations. ZRTP works well in a peer-to-peer network in which RTP is used, but it cannot support calls that traverse between VoIP networks and PSTN. Therefore, another mechanism needs to be established to support such interconnection.



Compare with partner:
w966

☐ Verified

SECURE
AES-128

Secure since:
Wed Mar 28 11:57:07 2007

Name
sip:2001@192.168.1.77

Figure 7.15 ZRTP in secure mode.

ZRTP and Man in the Middle

Because DH is susceptible to *man-in-the-middle* attacks, the ZRTP design provides a Short Authentication String (SAS). The SAS is used by the participants to determine whether their key exchange has been compromised. The legitimate parties announce the SAS string to each other when the initial handshake is completed, and they also set the V flag (SAS verified). This is available only in implementations in which the user can set the SAS verified flag, such as a soft phone. If the option to set the SAS flag is not available to the user, it is possible to perform a man-in-the-middle attack.[11]

ZRTP DoS

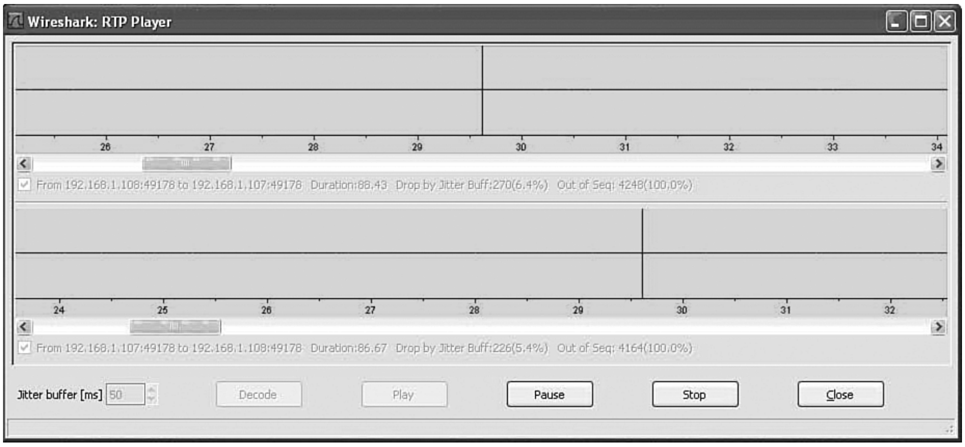
One common method to attack key-exchange protocols is by performing a DoS through resource consumption and exhaustion. In ZRTP, an attacker

11. P. Gupta, and V. Shmatikov. Security Analysis of Voice-over-IP Protocols. The University of Texas at Austin. Cryptology ePrint Archive, 2006.

may send spurious Hello messages to end points, thus forcing them to allocate resources and eventually causing them to degrade or terminate operation.[12] This attack requires that the exchange of signaling messages has preceded the ZRTP negotiation. The signaling messages may be an easier target for attack instead of waiting until the end points initiate RTP media exchange and exploit ZRTP.

ZRTP DTMF Disclosure

Although ZRTP is designed to protect the RTP stream, the current implementation of Zfone fails to protect DTMF (Dual Tone Multi Frequency) tones. RFC 2833 defines the payload format to transmit DTMF tones in RTP.[13] Many automated answering systems use DTMF tones to allow menu navigation and provide customer support and services. For example, when users dial their bank or health-care provider, they are prompted to enter their account number or Social Security number or other personal-identifiable information. It is possible for an attacker to capture a conversation between end points and retrieve credit card numbers, birthdates, PINS, Social Security numbers, or other confidential information. Figure 7.16 depicts a failed attempt to decode an RTP stream that is protected using ZRTP.



From 192.168.1.108:49178 to 192.168.1.107:49178 Duration:88.43 Drop by Jitter Buff:270(6.4%) Out of Seq: 4248(100.0%)

From 192.168.1.107:49178 to 192.168.1.108:49178 Duration:86.67 Drop by Jitter Buff:226(5.4%) Out of Seq: 4164(100.0%)

Figure 7.16 ZRTP and eavesdropping.

12. *P. Gupta, and V. Shmatikov. Security Analysis of Voice-over-IP Protocols. The University of Texas at Austin. Cryptology ePrint Archive 2006.*

13. *H. Sculzrinne, and S. Petrck. RFC 2833, "RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals."*

Figure 7.16 depicts a sniffer capture of the same RTP stream protected by ZRTP. In this figure, the user has pressed various numbers on his keypad that translated into DTMF tones. One of the DTMF tones is the number 2, which is clearly depicted in Figure 7.17.

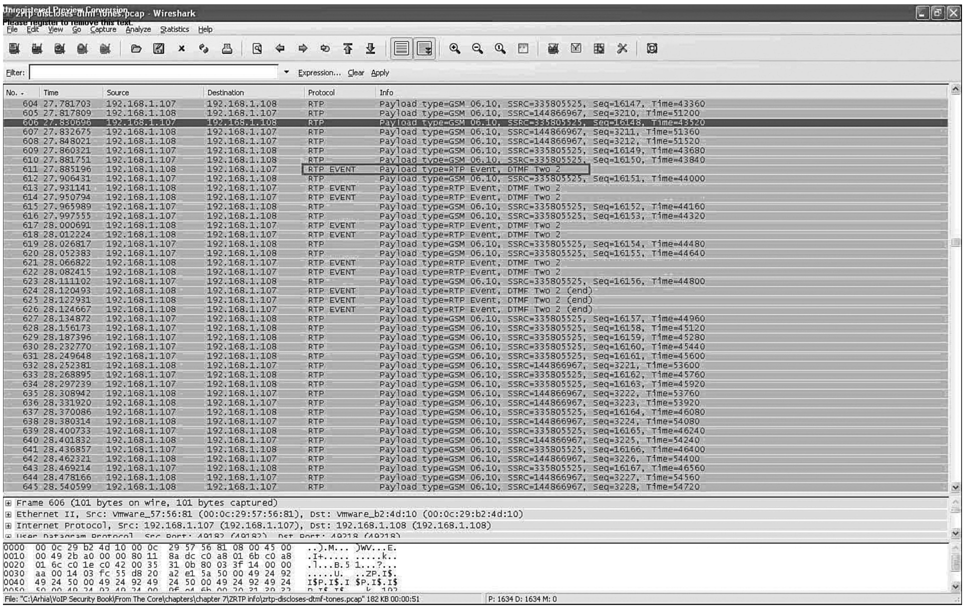
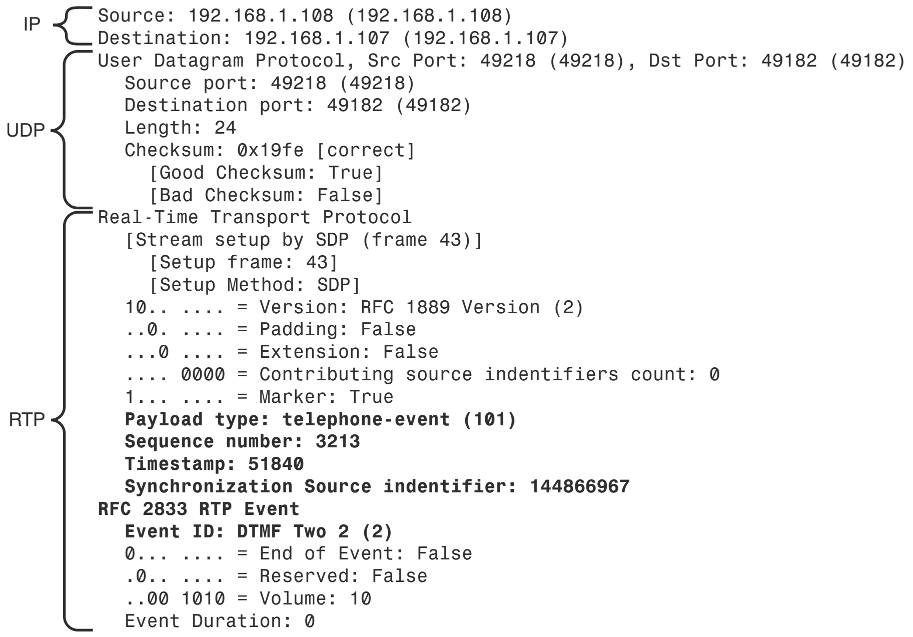

```
Unregistered Preview Conversion
zrtp-discloses-dtmf-tones.pcap - Wireshark
Please register to remove this text.
File  Edit  View  Go  Capture  Analyze  Statistics  Help
Filter:                                   Expression...  Clear  Apply

No. -  Time       Source         Destination    Protocol   Info
  604 27.781703   192.168.1.107  192.168.1.108  RTP        Payload type=GSM 06.10, SSRC=335805525, Seq=16147, Time=43360
  605 27.817809   192.168.1.108  192.168.1.107  RTP        Payload type=GSM 06.10, SSRC=144866967, Seq=3210, Time=51200
  606 27.830696   192.168.1.107  192.168.1.108  RTP        Payload type=GSM 06.10, SSRC=335805525, Seq=16148, Time=43520
  607 27.832675   192.168.1.108  192.168.1.107  RTP        Payload type=GSM 06.10, SSRC=144866967, Seq=3211, Time=51360
  608 27.848021   192.168.1.108  192.168.1.107  RTP        Payload type=GSM 06.10, SSRC=144866967, Seq=3212, Time=51520
  609 27.860321   192.168.1.107  192.168.1.108  RTP        Payload type=GSM 06.10, SSRC=335805525, Seq=16149, Time=43680
  610 27.881751   192.168.1.107  192.168.1.108  RTP        Payload type=GSM 06.10, SSRC=335805525, Seq=16150, Time=43840
  611 27.885196   192.168.1.108  192.168.1.107  RTP EVENT  Payload type=RTP Event, DTMF Two 2
  612 27.906431   192.168.1.107  192.168.1.108  RTP        Payload type=GSM 06.10, SSRC=335805525, Seq=16151, Time=44000
  613 27.931141   192.168.1.108  192.168.1.107  RTP EVENT  Payload type=RTP Event, DTMF Two 2
  614 27.950794   192.168.1.108  192.168.1.107  RTP EVENT  Payload type=RTP Event, DTMF Two 2
  615 27.965989   192.168.1.107  192.168.1.108  RTP        Payload type=GSM 06.10, SSRC=335805525, Seq=16152, Time=44160
  616 27.997555   192.168.1.107  192.168.1.108  RTP        Payload type=GSM 06.10, SSRC=335805525, Seq=16153, Time=44320
  617 28.000691   192.168.1.108  192.168.1.107  RTP EVENT  Payload type=RTP Event, DTMF Two 2
  618 28.012224   192.168.1.108  192.168.1.107  RTP EVENT  Payload type=RTP Event, DTMF Two 2
  619 28.026817   192.168.1.107  192.168.1.108  RTP        Payload type=GSM 06.10, SSRC=335805525, Seq=16154, Time=44480
  620 28.052383   192.168.1.107  192.168.1.108  RTP        Payload type=GSM 06.10, SSRC=335805525, Seq=16155, Time=44640
  621 28.066822   192.168.1.108  192.168.1.107  RTP EVENT  Payload type=RTP Event, DTMF Two 2
  622 28.082415   192.168.1.108  192.168.1.107  RTP EVENT  Payload type=RTP Event, DTMF Two 2
  623 28.111102   192.168.1.107  192.168.1.108  RTP        Payload type=GSM 06.10, SSRC=335805525, Seq=16156, Time=44800
  624 28.120493   192.168.1.108  192.168.1.107  RTP EVENT  Payload type=RTP Event, DTMF Two 2 (end)
  625 28.122931   192.168.1.108  192.168.1.107  RTP EVENT  Payload type=RTP Event, DTMF Two 2 (end)
  626 28.124667   192.168.1.108  192.168.1.107  RTP EVENT  Payload type=RTP Event, DTMF Two 2 (end)
  627 28.134872   192.168.1.107  192.168.1.108  RTP        Payload type=GSM 06.10, SSRC=335805525, Seq=16157, Time=44960
  628 28.156173   192.168.1.107  192.168.1.108  RTP        Payload type=GSM 06.10, SSRC=335805525, Seq=16158, Time=45120
  629 28.187396   192.168.1.107  192.168.1.108  RTP        Payload type=GSM 06.10, SSRC=335805525, Seq=16159, Time=45280
  630 28.232770   192.168.1.107  192.168.1.108  RTP        Payload type=GSM 06.10, SSRC=335805525, Seq=16160, Time=45440
  631 28.249648   192.168.1.107  192.168.1.108  RTP        Payload type=GSM 06.10, SSRC=335805525, Seq=16161, Time=45600
  632 28.252381   192.168.1.108  192.168.1.107  RTP        Payload type=GSM 06.10, SSRC=144866967, Seq=3221, Time=53600
  633 28.268895   192.168.1.107  192.168.1.108  RTP        Payload type=GSM 06.10, SSRC=335805525, Seq=16162, Time=45760
  634 28.297239   192.168.1.107  192.168.1.108  RTP        Payload type=GSM 06.10, SSRC=335805525, Seq=16163, Time=45920
  635 28.308942   192.168.1.108  192.168.1.107  RTP        Payload type=GSM 06.10, SSRC=144866967, Seq=3222, Time=53760
  636 28.331920   192.168.1.108  192.168.1.107  RTP        Payload type=GSM 06.10, SSRC=144866967, Seq=3223, Time=53920
  637 28.370086   192.168.1.107  192.168.1.108  RTP        Payload type=GSM 06.10, SSRC=335805525, Seq=16164, Time=46080
  638 28.380314   192.168.1.108  192.168.1.107  RTP        Payload type=GSM 06.10, SSRC=144866967, Seq=3224, Time=54080
  639 28.400733   192.168.1.107  192.168.1.108  RTP        Payload type=GSM 06.10, SSRC=335805525, Seq=16165, Time=46240
  640 28.401832   192.168.1.108  192.168.1.107  RTP        Payload type=GSM 06.10, SSRC=144866967, Seq=3225, Time=54240
  641 28.436857   192.168.1.107  192.168.1.108  RTP        Payload type=GSM 06.10, SSRC=335805525, Seq=16166, Time=46400
  642 28.462321   192.168.1.108  192.168.1.107  RTP        Payload type=GSM 06.10, SSRC=144866967, Seq=3226, Time=54400
  643 28.469214   192.168.1.107  192.168.1.108  RTP        Payload type=GSM 06.10, SSRC=335805525, Seq=16167, Time=46560
  644 28.478166   192.168.1.108  192.168.1.107  RTP        Payload type=GSM 06.10, SSRC=144866967, Seq=3227, Time=54560
  645 28.540599   192.168.1.108  192.168.1.107  RTP        Payload type=GSM 06.10, SSRC=144866967, Seq=3228, Time=54720

⊞ Frame 606 (101 bytes on wire, 101 bytes captured)
⊞ Ethernet II, Src: Vmware_57:56:81 (00:0c:29:57:56:81), Dst: Vmware_b2:4d:10 (00:0c:29:b2:4d:10)
⊞ Internet Protocol, Src: 192.168.1.107 (192.168.1.107), Dst: 192.168.1.108 (192.168.1.108)
⊞ User Datagram Protocol, Src Port: 49182 (49182), Dst Port: 49218 (49218)

0000  00 0c 29 b2 4d 10 00 0c  29 57 56 81 08 00 45 00   ..).M... )WV...E.
0010  00 49 2b a0 00 00 80 11  8a dc c0 a8 01 6b c0 a8   .I+..... .....k..
0020  01 6c c0 1e c0 42 00 35  31 0b 80 03 3f 14 00 00   .l...B.5 1...?...
0030  aa 00 14 03 fc 55 d8 20  a2 e1 5a 50 00 49 24 92   .....U.  ..ZP.I$.
0040  49 24 50 00 49 24 92 49  24 50 00 49 24 92 49 24   I$P.I$.I $P.I$.I$
0050  50 00 49 24 92 49 24 00  0f e4 6b 00 20 31 39 32   P.I$.I$. ..k. 192
File: "C:\Arhia\VoIP Security Book\From The Core\chapters\chapter 7\ZRTP info\zrtp-discloses-dtmf-tones.pcap" 182 KB 00:00:51    P: 1634 D: 1634 M: 0
```

Figure 7.17 DTMF disclosure in ZRTP.

Figure 7.18 depicts the captured message.

This vulnerability can have great impact in VoIP implementations in which sensitive information is exchanged through DTMF tones. One approach to address this weakness is to introduce a capability in the design of ZRTP to extend its protection of DTMF tones (for example, encrypting the RTP payload format of named events).



```
Source: 192.168.1.108 (192.168.1.108)
Destination: 192.168.1.107 (192.168.1.107)
User Datagram Protocol, Src Port: 49218 (49218), Dst Port: 49182 (49182)
  Source port: 49218 (49218)
  Destination port: 49182 (49182)
  Length: 24
  Checksum: 0x19fe [correct]
    [Good Checksum: True]
    [Bad Checksum: False]
Real-Time Transport Protocol
  [Stream setup by SDP (frame 43)]
    [Setup frame: 43]
    [Setup Method: SDP]
  10.. .... = Version: RFC 1889 Version (2)
  ..0. .... = Padding: False
  ...0 .... = Extension: False
  .... 0000 = Contributing source indentifiers count: 0
  1... .... = Marker: True
  Payload type: telephone-event (101)
  Sequence number: 3213
  Timestamp: 51840
  Synchronization Source indentifier: 144866967
RFC 2833 RTP Event
  Event ID: DTMF Two 2 (2)
  0... .... = End of Event: False
  .0.. .... = Reserved: False
  ..00 1010 = Volume: 10
  Event Duration: 0
```

Figure 7.18 ZRTP and DTMF disclosure: example packet.

Summary

Key management is a *must* to protect Internet multimedia applications such as VoIP, video on demand, conferencing, and others. This chapter covered two methods, MIKEY and SRTP Security Descriptions, currently implemented by vendors to support security requirements to provide authentication, confidentiality, and integrity of media streams. In addition, this chapter discussed ZRTP, which is currently an IETF "draft" but is likely to become a viable solution for peer-to-peer confidentiality. The MIKEY protocol provides the scalability and flexibility to support unicast and multicast communications, but it can be more complex to implement compared to SRTP Security Descriptions. Nevertheless, both approaches provide the ability to exchange cryptographic material and support the SRTP protocol to adequately protect the media streams between participants. ZRTP provides a level of transparency compared to MIKEY or

SDescriptions because it is signaling protocol independent and it requires changes on the peer software but not the core VoIP components such as a SIP proxy or an H.323 gatekeeper. One limitation that all key-exchange protocols suffer is that they cannot extend their properties to calls that traverse between VoIP networks and PSTN. Forking and media clipping are additional issues that require further research and need to be addressed by any key-exchange mechanism or protocol.[14] Currently, the IETF is working on several options, including EKT and redesigning MIKEY (MIKEYv2), to provide additional mechanism for key management.

14. D. Wing, et al., Media Security Requirements. IETF draft, www.ietf.org/Internet-drafts/draft-wing-media-security-requirements-00.txt, October 2006.