AVTCORE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 23, 2015

J. Mattsson, Ed.
Ericsson
D. McGrew
D. Wing
F. Andreasen
Cisco

October 20, 2014

Encrypted Key Transport for Secure RTP
draft-ietf-avtcore-srtp-ekt-03

Abstract

Encrypted Key Transport (EKT) is an extension to Secure Real-time Transport Protocol (SRTP) that provides for the secure transport of SRTP master keys, Rollover Counters, and other information. This facility enables SRTP to work for decentralized conferences with minimal control.

This note defines EKT, and also describes how to use it with SDP Security Descriptions, DTLS-SRTP, and MIKEY. With EKT, these other key management protocols provide an EKT key to everyone in a session, and EKT coordinates the SRTP keys within the session.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 23, 2015.

Copyright Notice

Table of Contents

1. Introduction

RTP is designed to allow decentralized groups with minimal control to establish sessions, such as for multimedia conferences. Unfortunately, Secure RTP (SRTP [RFC3711]) cannot be used in many minimal-control scenarios, because it requires that SSRC values and other data be coordinated among all of the participants in a session. For example, if a participant joins a session that is already in progress, that participant needs to be told the SRTP keys (and SSRC, ROC and other details) of the other SRTP sources.

The inability of SRTP to work in the absence of central control was well understood during the design of the protocol; the omission was considered less important than optimizations such as bandwidth conservation. Additionally, in many situations SRTP is used in conjunction with a signaling system that can provide most of the central control needed by SRTP. However, there are several cases in which conventional signaling systems cannot easily provide all of the coordination required. It is also desirable to eliminate the layer violations that occur when signaling systems coordinate certain SRTP parameters, such as SSRC values and ROCs.

This document defines Encrypted Key Transport (EKT) for SRTP, an extension to SRTP that fits within the SRTP framework and reduces the amount of external signaling control that is needed in an SRTP session. EKT securely distributes the SRTP master key and other information for each SRTP source (SSRC), using SRTCP or SRTP to transport that information. With this method, SRTP entities are free

to choose SSRC values as they see fit, and to start up new SRTP sources (SSRC) with new SRTP master keys (see Section 2.2) within a session without coordinating with other entities via external signaling or other external means. This fact allows to reinstate the RTP collision detection and repair mechanism, which is nullified by the current SRTP specification because of the need to control SSRC values closely. An SRTP endpoint using EKT can generate new keys whenever an existing SRTP master key has been overused, or start up a new SRTP source (SSRC) to replace an old SRTP source that has reached the packet-count limit. However, EKT does not allow SRTP's ROC to rollover; that requires re-keying outside of EKT (e.g., using MIKEY or DTLS-SRTP). EKT also solves the problem in which the burst loss of the N initial SRTP packets can confuse an SRTP receiver, when the initial RTP sequence number is greater than or equal to 2^16 - N. These features can simplify many architectures that implement SRTP.

EKT provides a way for an SRTP session participant, either a sender or receiver, to securely transport its SRTP master key and current SRTP rollover counter to the other participants in the session. This data, possibly in conjunction with additional data provided by an external signaling protocol, furnishes the information needed by the receiver to instantiate an SRTP/SRTCP receiver context.

EKT does not control the manner in which the SSRC is generated; it is only concerned with their secure transport. Those values may be generated on demand by the SRTP endpoint, or may be dictated by an external mechanism such as a signaling agent or a secure group controller.

EKT is not intended to replace external key establishment mechanisms such as SDP Security Descriptions [RFC4568], DTLS-SRTP [RFC5764], or MIKEY [RFC3830][RFC4563]. Instead, it is used in conjunction with those methods, and it relieves them of the burden of tightly coordinating every SRTP source (SSRC) among every SRTP participant.

1.1. History

[[RFC Editor Note: please remove this section prior to publication as an RFC.]]

A substantial change occurred between the EKT documents draft-ietf-avt-srtp-ekt-03 and draft-ietf-avtcore-srtp-ekt-00. The change makes it possible for the EKT data to be removed from a packet without affecting the ability of the receiver to correctly process the data that is present in that packet. This capability facilitates interoperability between SRTP implementations with different SRTP key management methods. The changes also greatly simplify the EKT

processing rules, and makes the EKT data that must be carried in SRTP and/or SRTCP packets somewhat larger.

In draft-ietf-avtcore-srtp-ekt-02, SRTP master keys have to be always generated randomly and not re-used, MKI is no longer allowed with EKT (as MKI duplicates some of EKT's functions), and text clarifies that EKT must be negotiated during call setup. Some text was consolidated and re-written, notably Section 2.6 ("Timing and Reliability"). Support for re-directing the DTLS-SRTP handshake to another host was removed, as it needed NAT traversal support.

In draft-ietf-avtcore-srtp-ekt-03, the SRTCP compound packet problem is discussed. Updates and clarifications were made to the SDESC and MIKEY sections.

1.2. Conventions Used In This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Encrypted Key Transport

In EKT, an SRTP master key is encrypted with a key encrypting key and the resulting ciphertext is transported in selected SRTCP packets or in selected SRTP packets. The key encrypting key is called an EKT key. A single such key suffices for a single SRTP session, regardless of the number of participants in that session. However, there can be multiple EKT keys used within a particular session.

EKT defines a new method of providing SRTP master keys to an endpoint. In order to convey the ciphertext of the SRTP master key, and other additional information, an additional EKT field is added to SRTP or SRTCP packets. When added to SRTCP, the EKT field appears at the end of the packet, after the authentication tag, if that tag is present, or after the SRTCP index otherwise. When added to SRTP, The EKT field appears at the end of the SRTP packet, after the authentication tag (if that tag is present), or after the ciphertext of the encrypted portion of the packet otherwise.

EKT MUST NOT be used in conjunction with SRTP's MKI (Master Key Identifier) or with SRTP's <From, To> [RFC3711], as those SRTP features duplicate some of the functions of EKT.

2.1. EKT Field Formats

The EKT Field uses one of the two formats defined below. These two formats can always be unambiguously distinguished on receipt by examining the final bit of the EKT Field, which is also the final bit of the SRTP or SRTCP packet. The first format is the Full EKT Field (or Full_EKT_Field), and the second is the Short EKT Field (or Short_EKT_Field). The formats are defined as

```
EKT_Plaintext = SRTP_Master_Key || SSRC || ROC || ISN

EKT_Ciphertext = EKT_Encrypt(EKT_Key, EKT_Plaintext)

Full_EKT_Field = EKT_Ciphertext || SPI || '1'

Short_EKT_Field = Reserved || '0'
```

Figure 1: EKT data formats

Here || denotes concatenation, and '1' and '0' denote single one and zero bits, respectively. These fields and data elements are defined as follows:

EKT_Plaintext: The data that is input to the EKT encryption operation. This data never appears on the wire, and is used only in computations internal to EKT.

EKT_Ciphertext: The data that is output from the EKT encryption operation, described in Section 2.3. This field is included in SRTP and SRTCP packets when EKT is in use. The length of this field is variable, and is equal to the ciphertext size N defined in Section 2.3. Note that the length of the field is inferable from the SPI field, since the particular EKT cipher used by the sender of a packet can be inferred from that field.

SRTP_Master_Key: On the sender side, the SRTP Master Key associated with the indicated SSRC. The length of this field depends on the cipher suite negotiated during call setup for SRTP or SRTCP.

SSRC: On the sender side, this field is the SSRC for this SRTP source. The length of this field is fixed at 32 bits.

Rollover Counter (ROC): On the sender side, this field is set to the current value of the SRTP rollover counter in the SRTP context associated with the SSRC in the SRTP or SRTCP packet. The length of this field is fixed at 32 bits.

Initial Sequence Number (ISN): If this field is nonzero, it indicates the RTP sequence number of the initial RTP packet that is protected using the SRTP master key conveyed (in encrypted form) by the EKT Ciphertext field of this packet. When this field is present in an RTCP packet it indicates the RTP sequence number of the first RTP packet encrypted by this master key. If the ISN field is zero, it indicates that the initial RTP/RTCP packet protected using the SRTP master key conveyed in this packet preceded, or was concurrent with, the last roll-over of the RTP sequence number, and thus should be used as the current master key for processing this packet. The length of this field is fixed at 16 bits.

Security Parameter Index (SPI): This field is included in SRTP and SRTCP packets when EKT is in use. It indicates the appropriate EKT key and other parameters for the receiver to use when processing the packet. It is an "index" into a table of possibilities (which are established via signaling or some other out-of-band means), much like the IPsec Security Parameter Index [RFC4301]. The length of this field is fixed at 15 bits. The parameters identified by this field are:

* The EKT key used to process the packet.

* The EKT cipher used to process the packet.

* The Secure RTP parameters associated with the SRTP Master Key carried by the packet and the SSRC value in the packet. Section 8.2. of [RFC3711] summarizes the parameters defined by that specification.

* The Master Salt associated with the Master Key. (This value is part of the parameters mentioned above, but we call it out for emphasis.) The Master Salt is communicated separately, via signaling, typically along with the EKT key.

Together, these data elements are called an EKT parameter set. Within each SRTP session, each distinct EKT parameter set that may be used MUST be associated with a distinct SPI value, to avoid ambiguity.

Reserved: The length of this field is 7 bits. MUST be all zeros on transmission, and MUST be ignored on reception.

The Full_EKT_Field and Short_EKT_Field formats are shown in Figure 2 and Figure 3, respectively. These figures show the on-the-wire data. The Ciphertext field holds encrypted data, and thus has no apparent inner structure.

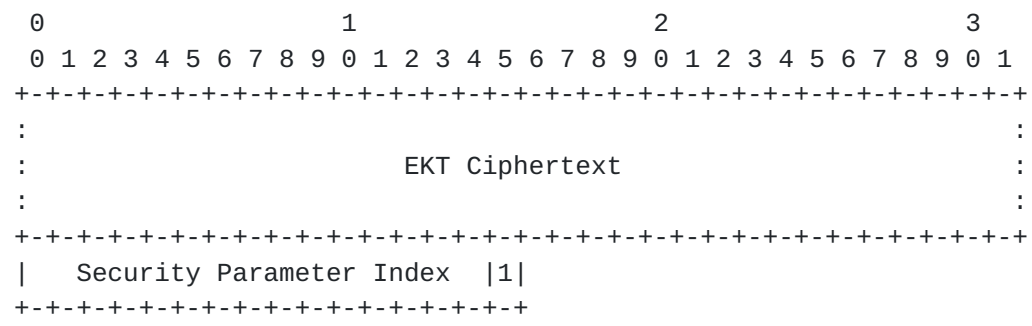

Figure 2: Full EKT Field format

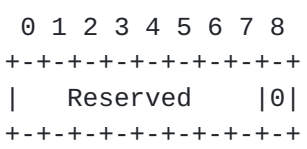

Figure 3: Short EKT Field format

2.2. Packet Processing and State Machine

At any given time, each SRTP/SRTCP source (SSRC) has associated with it a single EKT parameter set. This parameter set is used to process all outbound packets, and is called the outbound parameter set. There may be other EKT parameter sets that are used by other SRTP/SRTCP sources in the same session, including other SRTP/SRTCP sources on the same endpoint (e.g., one endpoint with voice and video might have two EKT parameter sets, or there might be multiple video sources on an endpoint each with their own EKT parameter set). All of these EKT parameter sets SHOULD be stored by all of the participants in an SRTP session, for use in processing inbound SRTP and SRTCP traffic.

All SRTP master keys MUST NOT be re-used, MUST be randomly generated according to [RFC4086], and MUST NOT be equal to or derived from other SRTP master keys.

2.2.1. Outbound Processing

See Section 2.6 which describes when to send an EKT packet and describes if a Full EKT Field or Short EKT Field is sent.

When an SRTP or SRTCP packet is to be sent, the EKT field for that packet is created as follows, or uses an equivalent set of steps. The creation of the EKT field MUST precede the normal SRTP or SRTCP packet processing. The ROC used in EKT processing MUST be the same as the one used in the SRTP processing.

If the Short format is used, an all-zero octet is appended to the packet. Otherwise, processing continues as follows.

The Rollover Counter field in the packet is set to the current value of the SRTP rollover counter (represented as an unsigned integer in network byte order).

The Initial Sequence Number field is set to zero, if the initial RTP packet protected using the current SRTP master key for this source preceded, or was concurrent with, the last roll-over of the RTP sequence number. Otherwise, that field is set to the value of the RTP sequence number of the initial RTP packet that was or will be protected by that key. See "rekey" in Section 2.6. The rekeying event MUST NOT change the value of ROC (otherwise, the current value of the ROC would not be known to late joiners of existing sessions). This means rekeying near the end of sequence number space (e.g., 100 packets before sequence number 65535) is not possible because ROC needs to roll over.

The Security Parameter Index field is set to the value of the Security Parameter Index that is associated with the outbound parameter set.

The EKT_Plaintext field is computed from the SRTP Master Key, SSRC, ROC, and ISN fields, as shown in Figure 1.

The EKT_Ciphertext field is set to the ciphertext created by encrypting the EKT_Plaintext with the EKT cipher, using the EKT Key as the encryption key. The encryption process is detailed in Section 2.3. Implementations MAY cache the value of this field to avoid recomputing it for each packet that is sent.

Implementation note: Because of the format of the Full EKT Field, a packet containing the Full EKT Field MUST be sent when the ROC changes (i.e., every 2^16 packets).

2.2.2. Inbound Processing

When an SRTP or SRTCP packet containing a Full EKT Field or Short EKT Field is received, it is processed as follows or using an equivalent set of steps. Inbound EKT processing MUST take place prior to the usual SRTP or SRTCP processing. Implementation note: the receiver may want to have a sliding window to retain old master keys for some brief period of time, so that out of order packets can be processed. The following steps show processing as packets are received in order.

1. The final bit is checked to determine which EKT format is in use. If the packet contains a Short EKT Field then the Short EKT Field

is removed and normal SRTP or SRTCP processing is applied. If the packet contains a Full EKT Field, then processing continues as described below.

2. The Security Parameter Index (SPI) field is checked to determine which EKT parameter set should be used when processing the packet. If multiple parameter sets have been defined for the SRTP session, then the one that is associated with the value of the SPI field in the packet is used. This parameter set is called the matching parameter set below. If there is no matching SPI, then the verification function MUST return an indication of authentication failure, and the steps described below are not performed.

3. The EKT_Ciphertext is decrypted using the EKT_Key and EKT_Cipher in the matching parameter set, as described in Section 2.3. If the EKT decryption operation returns an authentication failure, then the packet processing halts with an indication of failure. Otherwise, the resulting EKT_Plaintext is parsed as described in Figure 1, to recover the SRTP Master Key, SSRC, ROC, and ISN fields.

4. The SSRC field output from the decryption operation is compared to the SSRC field from the SRTP header if EKT was received over SRTP, or from the SRTCP header if EKT was received over SRTCP. If the values of the two fields do not match, then packet processing halts with an indication of failure. Otherwise, it continues as follows.

5. If an SRTP context associated with the SSRC in the previous step already exists and the ROC from the EKT_Plaintext is less than the ROC in the SRTP context, then EKT processing halts and the packet is processed as an out-of-order packet (if within the implementation's sliding window) or dropped (as it is a replay). Otherwise, the ROC in the SRTP context is set to the value of the ROC from the EKT_Plaintext, and the SRTP Master Key from the EKT_Plaintext is accepted as the SRTP master key corresponding to the SSRC indicated in the EKT_Plaintext, beginning at the sequence number indicated by the ISN (see next step).

6. If the ISN from the EKT_Plaintext is less than the RTP sequence number of an authenticated received SRTP packet, then EKT processing halts (as this is a replay). If the Initial Sequence Number field is nonzero, then the initial sequence number for the SRTP master key is set to the packet index created by appending that field to the current rollover counter and treating the result as a 48-bit unsigned integer. The initial sequence number for the master key is equivalent to the "From" value of the

<From, To> pair of indices (Section 8.1.1 of [RFC3711]) that can be associated with a master key.

7. The newly accepted SRTP master key, the SRTP parameters from the matching parameter set, and the SSRC from the packet are stored in the crypto context associated with the SRTP source (SSRC). The SRTP Key Derivation algorithm is run in order to compute the SRTP encryption and authentication keys, and those keys are stored for use in SRTP processing of inbound packets. The Key Derivation algorithm takes as input the newly accepted SRTP master key, along with the Master Salt from the matching parameter set.

8. At this point, EKT processing has successfully completed, and the normal SRTP or SRTCP processing takes place.

   Implementation note: the value of the EKT Ciphertext field is identical in successive packets protected by the same EKT parameter set and the same SRTP master key, ROC, and ISN. This ciphertext value MAY be cached by an SRTP receiver to minimize computational effort by noting when the SRTP master key is unchanged and avoiding repeating Steps 2 through 6.

2.3. Ciphers

EKT uses an authenticated cipher to encrypt the EKT Plaintext, which is comprised of the SRTP master keys, SSRC, ROC, and ISN. We first specify the interface to the cipher, in order to abstract the interface away from the details of that function. We then define the cipher that is used in EKT by default. The default cipher described in Section 2.3.1 MUST be implemented, but another cipher that conforms to this interface MAY be used, in which case its use MUST be coordinated by external means (e.g., key management).

The master salt length for the offered cipher suites MUST be the same. In practice the easiest way to achieve this is by offering the same crypto suite.

An EKT cipher consists of an encryption function and a decryption function. The encryption function E(K, P) takes the following inputs:

- a secret key K with a length of L bytes, and

- a plaintext value P with a length of M bytes.

The encryption function returns a ciphertext value C whose length is N bytes, where N is at least M. The decryption function D(K, C) takes the following inputs:

o a secret key K with a length of L bytes, and

o a ciphertext value C with a length of N bytes.

The decryption function returns a plaintext value P that is M bytes long, or returns an indication that the decryption operation failed because the ciphertext was invalid (i.e. it was not generated by the encryption of plaintext with the key K).

These functions have the property that D(K, E(K, P)) = P for all values of K and P. Each cipher also has a limit T on the number of times that it can be used with any fixed key value. For each key, the encryption function MUST NOT be invoked on more than T distinct values of P, and the decryption function MUST NOT be invoked on more than T distinct values of C.

The length of the EKT Plaintext is ten bytes, plus the length of the SRTP Master Key.

Security requirements for EKT ciphers are discussed in Section 8.

2.3.1. The Default Cipher

The default EKT Cipher is the Advanced Encryption Standard (AES) [FIPS197] Key Wrap with Padding [RFC5649] algorithm. It requires a plaintext length M that is at least one octet, and it returns a ciphertext with a length of N = M + 8 octets. It can be used with key sizes of L = 16, 24, and 32, and its use with those key sizes is indicated as AESKW_128, AESKW_192, and AESKW_256, respectively. The key size determines the length of the AES key used by the Key Wrap algorithm. With this cipher, T=2^48.

SRTP transform	EKT transform	length of EKT plaintext	length of EKT ciphertext	length of Full EKT Field
AES-128	AESKW_128 (m)	26	40	42
AES-192	AESKW_192	34	48	50
AES-256	AESKW_256	42	56	58
F8-128	AESKW_128	26	40	42
SEED-128	AESKW_128	26	40	42

Figure 4: AESKW Table

The mandatory to implement transform is AESKW_128, indicated by (m).

As AES-128 is the mandatory to implement transform in SRTP [RFC3711], AESKW_128 MUST be implemented for EKT.

For all the SRTP transforms listed in the table, the corresponding EKT transform MUST be used, unless a stronger EKT transform is negotiated by key management.

2.3.2. Other EKT Ciphers

Other specifications may extend this one by defining other EKT ciphers per Section 9. This section defines how those ciphers interact with this specification.

An EKT cipher determines how the EKT Ciphertext field is written, and how it is processed when it is read. This field is opaque to the other aspects of EKT processing. EKT ciphers are free to use this field in any way, but they SHOULD NOT use other EKT or SRTP fields as an input. The values of the parameters L, M, N, and T MUST be defined by each EKT cipher, and those values MUST be inferable from the EKT parameter set.

2.4. Synchronizing Operation

A participant in a session MAY opt to use a particular EKT parameter set to protect outbound packets after it accepts that EKT parameter set for protecting inbound traffic. In this case, the fact that one participant has changed to using a new EKT key for outbound traffic can trigger other participants to switch to using the same key.

If a source has its EKT key changed by key management, it MUST also change its SRTP master key, which will cause it to send out a new Full EKT Field. This ensures that if key management thought the EKT key needs changing (due to a participant leaving or joining) and communicated that in key management to a source, the source will also change its SRTP master key, so that traffic can be decrypted only by those who know the current EKT key.

The use of EKT MUST be negotiated during key management or call setup (e.g., using DTLS-SRTP, Security Descriptions, MIKEY, or similar).

2.5. Transport

EKT SHOULD be used over SRTP, and MAY be used over SRTCP. SRTP is preferred because it shares fate with transmitted media, because SRTP rekeying can occur without concern for RTCP transmission limits, and to avoid SRTCP compound packets with RTP translators and mixers.

This specification requires the EKT SSRC match the SSRC in the RTCP header, but Section 6.1 of [RFC3550] encourages creating SRTCP compound packets:

> It is RECOMMENDED that translators and mixers combine individual RTCP packets from the multiple sources they are forwarding into one compound packet whenever feasible in order to amortize the packet overhead (see Section 7).

These compound SRTCP packets might have an SSRC that does not match the EKT SSRC. To reduce the occasion of this occuring, EKT-aware RTP mixers and translators which are generating SRTCP compound packets SHOULD attempt to place an SRTCP payload containing an EKT tag at the front of the compound packet (so that the EKT receiver will process it), and MAY be even more robust and implement more sophisticated algorithms to ensure all EKT tags from different senders are sent at the front of the compound packet. However, no robust algorithm exists which ensures robust EKT delivery in conjunction with SRTCP compound packets. This impact to RTP translators and mixers, and the inability to realibly determine an RTP translator or mixer might be involved in an RTP session, provides further incentive to send EKT over RTP.

The packet processing, state machine, and Authentication Tag format for EKT over SRTP are nearly identical to that for EKT over SRTCP. Differences are highlighted in Section 2.2.1 and Section 2.2.2.

The Full EKT Field is appended to the SRTP or SRTCP payload and is 42, 50, or 58 octets long for AES-128, AES-192, or AES-256, respectively. This length impacts the maximum payload size of the SRTP (or SRTCP) packet itself. To remain below the network path MTU, senders SHOULD constrain the SRTP (or SRTCP) payload size by this length of the Full EKT Field.

EKT can be transported over SRTCP, but some of the information that it conveys is used for SRTP processing; some elements of the EKT parameter set apply to both SRTP and SRTCP. Furthermore, SRTCP packets can be lost and both SRTP and SRTCP packets may be delivered out of order. This can lead to various race conditions if EKT is transported over SRTCP but not SRTP, which we review below.

The ROC signaled via EKT over SRTCP may be off by one when it is received by the other party(ies) in the session. In order to deal with this, receivers should simply follow the SRTP packet index estimation procedures defined in Section 3.3.1 [RFC3711].

2.6. Timing and Reliability Consideration

A system using EKT has the SRTP master keys distributed with EKT, rather than with call signaling. A receiver can immediately decrypt an SRTP (or SRTCP packet) using that new key, provided the SRTP packet (or SRTCP packet) also contains a Full EKT Field.

This section describes how to reliably and expediently deliver new SRTP master keys to receivers.

There are three cases to consider. The first case is a new sender joining a session which needs to communicate its SRTP master key to all the receivers. The second case is a sender changing its SRTP master key which needs to be communicated to all the receivers. The third case is a new receiver joining a session already in progress which needs to know the sender's SRTP master key.

New sender: A new sender SHOULD send a packet containing the Full EKT Field as soon as possible, always before or coincident with sending its initial SRTP packet. To accommodate packet loss, it is RECOMMENDED that three consectutive packets contain the Full EKT Field be transmitted. Inclusion of that Full EKT Field can be stopped early if the sender determines all receivers have received the new SRTP master key by receipt of an SRTCP receiver report or explicit ACK for a sequence number with the new key.

Rekey: By sending EKT over SRTP, the rekeying event shares fate with the SRTP packets protected with that new SRTP master key. To avoid sending large SRTP packets (such as video key frames) with the Full EKT Field, it can be advantageous to send smaller SRTP packets with the Full EKT Field with the Initial Sequence Number prior to the actual rekey event, but this does eliminate the benefits of fate-sharing EKT with the SRTP packets with the new SRTP master key, which increases the chance a new receiver won't have seen the new SRTP master key.

New receiver: When a new receiver joins a session it does not need to communicate its sending SRTP master key (because it is a receiver). When a new receiver joins a session the sender is generally unaware of the receiver joining the session. Thus, senders SHOULD periodically transmit the Full EKT Field. That interval depends on how frequently new receivers join the session, the acceptable delay before those receivers can start processing SRTP packets, and the acceptable overhead of sending the Full EKT Field. The RECOMMENDED frequency is the same as the key frame frequency if sending video or every 5 seconds. When joining a session it is likely that SRTP or SRTCP packets might be received before a packet containing the Full EKT Field is received. Thus, to avoid doubling the authentication

effort, an implementation joining an EKT session SHOULD buffer received SRTP and SRTCP packets until it receives the Full EKT Field packet and use the information in that packet to authenticate and decrypt the received SRTP/SRTCP packets.

3. Use of EKT with SDP Security Descriptions

The SDP Security Descriptions (SDESC) [RFC4568] specification defines a generic framework for negotiating security parameters for media streams negotiated via the Session Description Protocol with the "crypto" attribute and the Offer/Answer procedures defined in [RFC3264]. In addition to the general framework, SDESC also defines how to use that framework specifically to negotiate security parameters for Secure RTP. Below, we first provide a brief recap of the crypto attribute when used for SRTP and we then explain how it is complementary to EKT. In the rest of this Section, we provide extensions to the crypto attribute and associated offer/answer procedures to define its use with EKT.

3.1. SDP Security Descriptions Recap

The SRTP crypto attribute defined for SDESC contains a tag followed by three types of parameters (refer to [RFC4568] for details):

- Crypto-suite. Identifies the encryption and authentication transform.
- Key parameters. SRTP keying material and parameters.
- Session parameters. Additional (optional) SRTP parameters such as Key Derivation Rate, Forward Error Correction Order, use of unencrypted SRTP, and other parameters defined by SDESC.

The crypto attributes in the example SDP in Figure 5 illustrate these parameters.

```
v=0
o=sam 2890844526 2890842807 IN IP4 192.0.2.5
s=SRTP Discussion
i=A discussion of Secure RTP
u=http://www.example.com/seminars/srtp.pdf
e=marge@example.com (Marge Simpson)
c=IN IP4 192.0.2.12
t=2873397496 2873404696
m=audio 49170 RTP/SAVP 0
a=crypto:1 AES_CM_128_HMAC_SHA1_80
   inline:WVNfX19zZW1jdGwgKCkgewkyMjA7fQp9CnVubGVz|2^20
   FEC_ORDER=FEC_SRTP
a=crypto:2 F8_128_HMAC_SHA1_80
   inline:MTIzNDU2Nzg5QUJDREUwMTIzNDU2Nzg5QUJjiiKt|2^20
   FEC_ORDER=FEC_SRTP
```

Figure 5: SDP Security Descriptions example

For legibility the SDP shows line breaks that are not present on the wire.

The first crypto attribute has the tag "1" and uses the crypto-suite AES_CM_128_HMAC_SHA1_80. The "inline" parameter provides the SRTP master key and salt and the master key lifetime (number of packets). Finally, the FEC_ORDER session parameter indicates the order of Forward Error Correction used (FEC is applied before SRTP processing by the sender of the SRTP media).

The second crypto attribute has the tag "2", the crypto-suite F8_128_HMAC_SHA1_80, a SRTP master key, and its associated salt. Finally, the FEC_ORDER session parameter indicates the order of Forward Error Correction used.

3.2. Relationship between EKT and SDESC

SDP Security Descriptions [RFC4568] define a generic framework for negotiating security parameters for media streams negotiated via the Session Description Protocol by use of the Offer/Answer procedures defined in [RFC3264]. In addition to the general framework, SDESC also defines how to use it specifically to negotiate security parameters for Secure RTP.

EKT and SDP Security Descriptions are complementary. SDP Security Descriptions can negotiate several of the SRTP security parameters (e.g., cipher and use of Master Key Identifier) as well as SRTP master keys. SDESC, however, does not negotiate SSRCs and their associated Rollover Counter (ROC). Instead, SDESC relies on a so-called "late binding", where a newly observed SSRC will have its

crypto context initialized to a ROC value of zero. Clearly, this does not work for participants joining an SRTP session that has been established for a while and hence has a non-zero ROC. It is impossible to use SDESC to join an SRTP session that is already in progress. In this case, EKT on the endpoint running SDESC can provide the additional signaling necessary to communicate the ROC (Section 6.4.1 of [RFC4568]). The use of EKT solves this problem by communicating the ROC associated with the SSRC in the media plane.

SDP Security Descriptions negotiates different SRTP master keys in the send and receive direction. The offer contains the master key used by the offerer to send media, and the answer contains the master key used by the answerer to send media. Consequently, if media is received by the offerer prior to the answer being received, the offerer does not know the master key being used. Use of SDP security preconditions can solve this problem, however it requires an additional round-trip as well as a more complicated state machine. EKT solves this problem by simply sending the master key used in the media plane thereby avoiding the need for security preconditions.

If multiple crypto-suites were offered, the offerer also will not know which of the crypto-suites offered was selected until the answer is received. EKT solves this problem by using a correlator, the Security Parameter Index (SPI), which uniquely identifies each crypto attribute in the offer.

One of the primary call signaling protocols using offer/answer is the Session Initiation Protocol (SIP) [RFC3261]. SIP uses the INVITE message to initiate a media session and typically includes an offer SDP in the INVITE. An INVITE may be "forked" to multiple recipients which potentially can lead to multiple answers being received. SDESC, however, does not properly support this scenario, mainly because SDP and RTP/RTCP does not contain sufficient information to allow for correlation of an incoming RTP/RTCP packet with a particular answer SDP. Note that extensions providing this correlation do exist (e.g., Interactive Connectivity Establishment (ICE)). SDESC addresses this point-to-multipoint problem by moving each answer to a separate RTP transport address thereby turning a point-to-multipoint scenario into multiple point-to-point scenarios. There are however significant disadvantages to doing so. As long as the crypto attribute in the answer does not contain any declarative parameters that differ from those in the offer, EKT solves this problem by use of the SPI correlator and communication of the answerer's SRTP master key in EKT.

As can be seen from the above, the combination of EKT and SDESC provides a better solution to SRTP negotiation for offer/answer than either of them alone. SDESC negotiates the various SRTP crypto

parameters (which EKT does not), whereas EKT addresses some of the shortcomings of SDESC.

3.3. Overview of Combined EKT and SDESC Operation

We define a new session parameter to SDESC to communicate the EKT cipher, EKT key, and Security Parameter Index to the peer. The original SDESC parameters are used as defined in [RFC4568], however the procedures associated with the SRTP master key differ slightly, since both SDESC and EKT communicate an SRTP master key. In particular, the SRTP master key communicated via SDESC is used only if there is currently no crypto context established for the SSRC in question. This will be the case when an entity has received only the offer or answer, but has yet to receive a valid EKT packet from the peer. Once a valid EKT packet is received for the SSRC, the crypto context is initialized accordingly, and the SRTP master key will then be derived from the EKT packet. Subsequent offer/answer exchanges do not change this: The most recent SRTP master key negotiated via EKT will be used, or, if none is available for the SSRC in question, the most recent SRTP master key negotiated via offer/answer will be used. This is done to avoid race conditions between the offer/answer exchange and EKT, even though this breaks some offer/answer rules. Note that with the rules described in this paragraph, once a valid EKT packet has been received for a given SSRC, rekeying for that SSRC can only be done via EKT. The associated SRTP crypto parameters however can be changed via SDESC.

3.4. EKT Extensions to SDP Security Descriptions

In order to use EKT and SDESC in conjunction with each other, the new SDESC session parameter "EKT" is defined. It MUST NOT appear more than once in a given crypto attribute. In the Offer/Answer model, the EKT parameter is a negotiated parameter.The "EKT" session parameter consists of three parts (the formal grammar is provided in Section 3.9):

"EKT=" <EKT_Cipher> "|" <EKT_Key> "|" <EKT_SPI>

Below are details on each of these attributes.

EKT_Cipher: The (optional) EKT_Cipher field defines the EKT cipher used to encrypt the EKT key within SRTP and SRTCP packets. The default value is "AESKW_128" in accordance with Section 2.3.1. For the AES Key Wrap cipher, the values "AESKW_128", "AESKW_192", and "AESKW_256" are defined for values of L=16, 24, and 32 respectively.

EKT_Key: The (mandatory) EKT_Key field is the EKT key used to encrypt the SRTP Master Key within SRTP and SRTCP packets. The value is base64 encoded with "=" padding if padding is necessary (see Section 3.2 and 4 of [RFC4648]).

EKT_SPI: The (mandatory) EKT_SPI field is the Security Parameter Index. It is encoded as an ASCII string representing the hexadecimal value of the Security Parameter Index. The SPI identifies the *offer* crypto attribute (including the EKT Key and Cipher) being used for the associated SRTP session. A crypto attribute corresponds to an EKT Parameter Set and hence the SPI effectively identifies a particular EKT parameter set. Note that the scope of the SPI is the SRTP session, which may or may not be limited to the scope of the associated SIP dialog. In particular, if one of the participants in an SRTP session is an SRTP translator, the scope of the SRTP session is not limited to the scope of a single SIP dialog. However, if all of the participants in the session are endpoints or mixers, the scope of the SRTP session will correspond to a single SIP dialog.

3.5. Offer/Answer Considerations

In this section, we provide the offer/answer procedures associated with use of the new SDESC session parameter defined in Section 3.4. Since SDESC is defined only for unicast streams, we provide only offer/answer procedures for unicast streams here as well.

3.5.1. Generating the Initial Offer - Unicast Streams

When the initial offer is generated, the offerer MUST follow the steps defined in [RFC4568] Section 7.1.1 as well as the following steps.

[[Editor's Note: following paragraph would benefit from rewording.]]

For each unicast media line using Security Descriptions and where use of EKT is desired, the offerer MUST include the EKT parameter in at least one "crypto" attribute (see [RFC4568]). The EKT paramater MUST contain the EKT_Key and EKT_SPI fields. The EKT_SPI field serves to identify the EKT parameter set used for a particular SRTP or SRTCP packet. Consequently, within a single media line, a given EKT_SPI value MUST NOT be used with multiple crypto attributes. Note that the EKT parameter set to use for the session is not yet established at this point; each offered crypto attribute contains a candidate EKT parameter set. Furthermore, if the media line refers to an existing SRTP session, then any SPI values used for EKT parameter sets in that session MUST NOT be remapped to any different EKT parameter sets. When an offer describes an SRTP session that is already in progress,

the offer SHOULD use an EKT parameter set (including EKT_SPI and EKT_KEY) that is already in use.

As EKT is not defined for use with MKI, a "crypto" attribute containing the EKT parameter MUST NOT contain MKI.

Important Note: The scope of the offer/answer exchange is the SIP dialog(s) established as a result of the INVITE, however the scope of EKT is the direct SRTP session, i.e., all the participants that are able to receive SRTP and SRTCP packets directly. If an SRTP session spans multiple SIP dialogs, the EKT parameter sets MUST be synchronized between all the SIP dialogs where SRTP and SRTCP packets can be exchanged. In the case where the SIP entity operates as an RTP mixer (and hence re-originates SRTP and SRTCP packets with its own SSRC), this is not an issue, unless the mixer receives traffic from the various participants on the same destination IP address and port, in which case further coordination of SPI values and crypto parameters may be needed between the SIP dialogs (note that SIP forking with multiple early media senders is an example of this). However, if it operates as a transport translator (relay) then synchronized negotiation of the EKT parameter sets on *all* the involved SIP dialogs will be needed. This is non-trivial in a variety of use cases, and hence use of the combined SDES/EKT mechanism with RTP translators should be considered very carefully. It should be noted, that use of SRTP with RTP translators in general should be considered very carefully as well.

The session parameter "EKT" can either be included as an optional or mandatory parameter.

3.5.2. Generating the Initial Answer - Unicast Streams

When the initial answer is generated, the answerer MUST follow the steps defined in [RFC4568] Section 7.1.2 as well as the following steps.

For each unicast media line using SDESC, the answerer examines the associated crypto attribute(s) for the presence of the session parameter "EKT". If a mandatory EKT parameter is included with a "crypto" attribute, the answerer MUST support those parameters in order to accept that offered crypto attribute. If an optional EKT parameter is included instead, the answerer MAY accept the offered crypto attribute without using EKT. However, doing so will prevent the offerer from processing any packets received before the answer. If no EKT parameter are included with a crypto attribute, and that crypto attribute is accepted in the answer, EKT MUST NOT be used. If

a given a crypto attribute includes a malformed EKT parameter, that crypto attribute MUST be considered invalid.

When EKT is used with SDESC, the offerer and answerer MUST use the same SRTP master salt. Thus, the SRTP key parameter(s) in the answer crypto attribute MUST use the same master salt as the one accepted from the offer.

When the answerer accepts the offered media line and EKT is being used, the crypto attribute included in the answer MUST include the same EKT parameter values as found in the accepted crypto attribute from the offerer (however, if the default EKT cipher is being used, it may be omitted). Furthermore, the EKT parameter included MUST be mandatory (i.e., no "-" prefix).

Acceptance of a crypto attribute with an EKT parameter leads to establishment of the EKT parameter set for the corresponding SRTP session. Consequently, the answerer MUST send packets in accordance with that particular EKT parameter set only. If the answerer wants to enable the offerer to process SRTP packets received by the offerer before it receives the answer, the answerer MUST NOT include any declarative session parameters that either were not present in the offered crypto attribute, or were present but with a different value. Otherwise, the offerer's view of the EKT parameter set would differ from the answerer's until the answer is received. Similarly, unless the offerer and answerer has other means for correlating an answer with a particular SRTP session, the answer SHOULD NOT include any declarative session parameters that either were not present in the offered crypto attribute, or were present but with a different value. If this recommendation is not followed and the offerer receives multiple answers (e.g., due to SIP forking), the offerer may not be able to process incoming media stream packets correctly.

3.5.3. Processing of the Initial Answer - Unicast Streams

When the offerer receives the answer, it MUST perform the steps in [RFC4568] Section 7.1.3 as well as the following steps for each SRTP media stream it offered with one or more crypto lines containing EKT parameters in it.

[[Editor's Note: following paragraph would benefit from rewording.]]

If the answer crypto line contains an EKT parameter, and the corresponding crypto line from the offer contained the same EKT values, use of EKT has been negotiated successfully and MUST be used for the media stream. When determining whether the values match, an optional and mandatory parameter MUST be considered equal.

Furthermore, if the default EKT cipher is being used, it MAY be either present or absent in the offer and/or answer.

If the answer crypto line does not contain an EKT parameter, then EKT MUST NOT be used for the corresponding SRTP session. Note that if the accepted crypto attribute contained a mandatory EKT parameter in the offer, and the crypto attribute in the answer does not contain an EKT parameter, then negotiation has failed (Section 5.1.3 of [RFC4568]).

If the answer crypto line contains an EKT parameter but the corresponding offered crypto line did not, or if the values don't match or are invalid, then the offerer MUST consider the crypto line invalid (see Section 7.1.3 of [RFC4568] for further operation).

The EKT parameter set is established when the answer is received, however there are a couple of special cases to consider here. First of all, if an SRTP packet containing a Full EKT Field is received prior to the answer, then the EKT parameter set is established provisionally based on the SPI included. Once the answer (which may include declarative session parameters) is received, the EKT parameter set is fully established. The second case involves receipt of multiple answers due to SIP forking. In this case, there will be multiple EKT parameter sets; one for each SRTP session. As mentioned earlier, reliable correlation of SIP dialogs to SRTP sessions requires extensions, and hence if one or more of the answers include declarative session parameters, it may be difficult to fully establish the EKT parameter set for each SRTP session. In the absence of a specific correlation mechanism, it is RECOMMENDED, that such correlation be done based on the signaled receive IP-address in the SDP and the observed source IP-address in incoming SRTP/SRTCP packets, and, if necessary, the signaled receive UDP port and the observed source UDP port.

3.6. SRTP-Specific Use Outside Offer/Answer

Security Descriptions use for SRTP is not defined outside offer/answer and hence neither does Security Descriptions with EKT.

3.7. Modifying the Session

When a media stream using the SRTP security descriptions has been established, and a new offer/answer exchange is performed, the offerer and answerer MUST follow the steps in Section 7.1.4 of [RFC4568] as well as the following steps. SDESC allows for all parameters of the session to be modified, and the EKT session parameter are no exception to that, however, there are a few additional rules to be adhered to when using EKT.

It is permissible to start a session without the use of EKT, and then subsequently start using EKT, however the converse is not. Thus, once use of EKT has been negotiated on a particular media stream, EKT MUST continue to be used on that media stream in all subsequent offer/answer exchanges.

The reason for this is that both SDESC and EKT communicate the SRTP master key with EKT communicated master keys taking precedence. Reverting back to an SDESC-controlled master key in a synchronized manner is difficult.

Once EKT is being used, the salt for the direct SRTP session MUST NOT be changed. Thus, a new offer/answer which does not create a new SRTP session (e.g., because it reuses the same IP address and port) MUST use the same salt for all crypto attributes as is currently used for the direct SRTP session.

[[Editor's Note: following paragraph would benefit from re-arranging into earlier-described steps.]]

Finally, subsequent offer/answer exchanges MUST NOT remap a given SPI value to a different EKT parameter set until 2^15 other mappings have been used within the SRTP session. In practice, this requirements is most easily met by using a monotonically increasing SPI value (modulo 2^15 and starting with zero) per direct SRTP session. Note that a direct SRTP session may span multiple SIP dialogs, and in such cases coordination of SPI values across those SIP dialogs will be required. In the simple point-to-point unicast case without translators, the requirement simply applies within each media line in the SDP. In the point-to-multipoint case, the requirement applies across all the associated SIP dialogs.

3.8. Backwards Compatibility Considerations

Backwards compatibility can be achieved in a couple of ways. First of all, Security Descriptions allows for session parameters to be prefixed with "-" to indicate that they are optional. If the answerer does not support the EKT session parameter, such optional parameters will simply be ignored. When the answer is received, absence of the parameter will indicate that EKT is not being used. Receipt of SRTP or SRTCP packets prior to receipt of such an answer will obviously be problematic (as is normally the case for Security Descriptions without EKT).

Alternatively, Security Descriptions allows for multiple crypto lines to be included for a particular media stream. Thus, two crypto lines that differ in their use of EKT parameters (presence in one, absence in the other) can be used as a way to negotiate use of EKT. When the

answer is received, the accepted crypto attribute will indicate whether EKT is being used or not.

3.9. Grammar

The ABNF [RFC5234] syntax for the one new SDP Security Descriptions session parameter, EKT, comprising three parts is shown in Figure 6.

```
ekt        = "EKT=" cipher "|" key "|" spi
cipher     = cipher-ext / "AESKW_128" / "AESKW_192" / "AESKW_256"
cipher-ext = 1*64(ALPHA / DIGIT / "_")
key        = 1*(base64)    ; See Section 4 of [RFC4648]
base64     = ALPHA / DIGIT / "+" / "/" / "="
spi        = 4HEXDIG   ; See [RFC5234]
```

Figure 6: ABNF for the EKT session parameters

Using the example from Figure 6 with the EKT extensions to SDP Security Descriptions results in the following example SDP:

```
v=0
o=sam 2890844526 2890842807 IN IP4 192.0.2.5
s=SRTP Discussion
i=A discussion of Secure RTP
u=http://www.example.com/seminars/srtp.pdf
e=marge@example.com (Marge Simpson)
c=IN IP4 192.0.2.12
t=2873397496 2873404696
m=audio 49170 RTP/SAVP 0
a=crypto:1 AES_CM_128_HMAC_SHA1_80
  inline:WVNfX19zZW1jdGwgKCkgewkyMjA7fQp9CnVubGVz|2^20
  FEC_ORDER=FEC_SRTP EKT=AESKW_128|WWVzQUxvdmVseUVLVGtleQ|AAE0
a=crypto:2 F8_128_HMAC_SHA1_80
  inline:MTIzNDU2Nzg5QUJDREUwMTIzNDU2Nzg5QUJjZGVm|2^20
  FEC_ORDER=FEC_SRTP EKT=AESKW_128|VHdvTG92Zwx5RUtUa2V5cw|AAE1
```

For legibility the SDP shows line breaks that are not present on the wire.

Figure 7: SDP Security Descriptions example with EKT

4. Use of EKT with DTLS-SRTP

This document defines an extension to DTLS-SRTP called Key Transport. The EKT with the DTLS-SRTP Key Transport enables secure transport of EKT keying material from one DTLS-SRTP peer to another. This enables those peers to process EKT keying material in SRTP (or SRTCP) and retrieve the embedded SRTP keying material. This combination of

protocols is valuable because it combines the advantages of DTLS (strong authentication of the endpoint and flexibility) with the advantages of EKT (allowing secure multiparty RTP with loose coordination and efficient communication of per-source keys).

4.1. DTLS-SRTP Recap

DTLS-SRTP [RFC5764] uses an extended DTLS exchange between two peers to exchange keying material, algorithms, and parapeters for SRTP. The SRTP flow operates over the same transport as the DTLS-SRTP exchange (i.e., the same 5-tuple). DTLS-SRTP combines the performance and encryption flexibility benefits of SRTP with the flexibility and convenience of DTLS-integrated key and association management. DTLS-SRTP can be viewed in two equivalent ways: as a new key management method for SRTP, and a new RTP-specific data format for DTLS.

4.2. EKT Extensions to DTLS-SRTP

This document adds a new TLS negotiated extension called "ekt". This adds a new TLS content type, EKT, and a new negotiated extension EKT. The negotiated extension MUST only be requested in conjunction with the "use_srtp" extension (Section 3.2 of [RFC5764]). The DTLS server MUST include "dtls-srtp-ekt" in its SDP (as a session or media level attribute) and "ekt" in its TLS ServerHello message. If a DTLS client includes "ekt" in its ClientHello, but does not receive "ekt" in the ServerHello, the DTLS client MUST NOT send DTLS packets with the "ekt" content-type.

The formal description of the dtls-srtp-ekt attribute is defined by the following ABNF [RFC5234] syntax:

attribute = "a=dtls-srtp-ekt"

Using the syntax described in DTLS [RFC6347], the following structures are used:

```
enum {
  ekt_key(0),
  ekt_key_ack(1),
  ekt_key_error(254),
  (255)
} SRTPKeyTransportType;

struct {
  SRTPKeyTransportType keytrans_type;
  uint24 length;
  uint16 message_seq;
  uint24 fragment_offset;
  uint24 fragment_length;
  select (SRTPKeyTransportType) {
     case ekt_key:
        EKTkey;
   };
} KeyTransport;

enum {
 RESERVED(0),
 AESKW_128(1),
 AESKW_192(2),
 AESKW_256(3),
} ektcipher;

struct {
  ektcipher EKT_Cipher;
  uint EKT_Key_Value<1..256>;
  uint EKT_Master_Salt<1..256>;
  uint16 EKT_SPI;
} EKTkey;
```

Figure 8: Additional TLS Data Structures

The diagram below shows a message flow of DTLS client and DTLS server using the DTLS-SRTP Key Transport extension. SRTP packets exchanged prior to the ekt_message are encrypted using the SRTP master key derived from the normal DTLS-SRTP key derivation function. After the ekt_key message, they can be encrypted using the SRTP key carried by EKT.

Figure 9: Handshake Message Flow

4.3. Offer/Answer Considerations

This section describes Offer/Answer considerations for the use of EKT together with DTLS-SRTP for unicast and multicast streams. The offerer and answerer MUST follow the procedures specified in [RFC5764] as well as the following ones.

As most DTLS-SRTP processing is performed on the media channel, rather than in SDP, there is little processing performed in SDP other than informational and to redirect DTLS-SRTP to an alternate host. Advertising support for the extension is necessary in SDP because in some cases it is required to establish an SRTP call. For example, a mixer may be able to only support SRTP listeners if those listeners implement DTLS Key Transport (because it lacks the CPU cycles necessary to encrypt SRTP uniquely for each listener).

4.3.1. Generating the Initial Offer

The initial offer contains a new SDP attribute, "dtls-srtp-ekt", which contains no value. This attribute MUST only appear at the media level. This attribute indicates the offerer is capable of supporting DTLS-SRTP with EKT extensions, and indicates the desire to use the "ekt" extension during the DTLS-SRTP handshake.

An example of SDP containing the dtls-srtp-ekt attribute::

```
v=0
o=sam 2890844526 2890842807 IN IP4 192.0.2.5
s=SRTP Discussion
i=A discussion of Secure RTP
u=http://www.example.com/seminars/srtp.pdf
e=marge@example.com (Marge Simpson)
c=IN IP4 192.0.2.12
t=2873397496 2873404696
m=audio 49170 UDP/TLS/RTP/SAVP 0
a=fingerprint:SHA-1
  4A:AD:B9:B1:3F:82:18:3B:54:02:12:DF:3E:5D:49:6B:19:E5:7C:AB
a=dtls-srtp-ekt
```

For legibility the SDP shows line breaks that are not present on the wire.

4.3.2. Generating the Initial Answer

Upon receiving the initial offer, the presence of the dtls-srtp-ekt attribute indicates a desire to receive the EKT extension in the DTLS-SRTP handshake. DTLS messages should be constructed according to those two attributes.

If the answerer does not wish to perform EKT, it MUST NOT include a=dtls-srtp-ekt in the SDP answer, and it MUST NOT negotiate EKT during its DTLS-SRTP exchange.

Otherwise, the dtls-srtp-ekt attribute SHOULD be included in the answer, and EKT SHOULD be negotiated in the DTLS-SRTP handshake.

4.3.3. Processing the Initial Answer

The presence of the dtls-srtp-ekt attribute indicates a desire by the answerer to perform DTLS-SRTP with EKT extensions. There are two indications the remote peer does not want to do EKT: the dtls-srtp-ekt attribute is not present in the answer, or the DTLS-SRTP exchange fails to negotiate the EKT extension. If the dtls-srtp-ekt attribute is not present in the answer, the DTLS-SRTP exchange MUST NOT attempt to negotiate the EKT extension. If the dtls-srtp-ekt attribute is present in the answer but the DTLS-SRTP exchange fails to negotiate the EKT extension, EKT MUST NOT be used with that media stream.

After successful DTLS negotiation of the EKT extension, the DTLS client and server MAY exchange SRTP packets, encrypted using the KDF described in [RFC5764]. This is normal and expected, even if Key Transport was negotiated by both sides, as neither side may (yet)

have a need to alter the SRTP key. However, it is also possible that one (or both) peers will immediately send an EKT packet before sending any SRTP, and also possible that SRTP, encrypted with an unknown key, may be received before the EKT packet is received.

4.3.4. Sending DTLS EKT Key Reliably

In the absence of a round trip time estimate, the DTLS ekt_key message is sent using an exponential backoff initialized to 250ms, so that if the first message is sent at time 0, the next transmissions are at 250ms, 500ms, 1000ms, and so on. If a recent round trip time estimate is available, exponential backoff is used with the first transmission at 1.5 times the round trip time estimate. In either case, re-transmission stops when ekt_key_ack or ekt_key_error message is received for the matching message_seq.

4.3.5. Modifying the Session

As DTLS-SRTP-EKT processing is done on the DTLS-SRTP channel (media channel) rather than signaling, no special processing for modifying the session is necessary.

If the initial offer and initial answer both contained EKT attributes (indicating the answerer desired to perform EKT), a subsequent offer/answer exchange MUST also contain those same EKT attributes. If not, operation is undefined and the sesion MAY be terminated. If the initial offer and answer failed to negotiate EKT (that is, the answer did not contain EKT attributes), EKT negotiation failed and a subsequent offer SHOULD NOT include EKT attributes.

5. Use of EKT with MIKEY

The advantages outlined in Section 1 are useful in some scenarios in which MIKEY is used to establish SRTP sessions. In this section, we briefly review MIKEY and related work, and discuss these scenarios.

An SRTP sender or a group controller can use MIKEY to establish a SRTP cryptographic context. This capability includes the distribution of a TEK generation key (TGK) or the TEK itself, security policy payload, crypto session bundle ID (CSB_ID) and a crypto session ID (CS_ID). The TEK directly maps to an SRTP master key, whereas the TGK is used along with the CSB_ID and a CS_ID to generate a TEK. The CS_ID is used to generate multiple TEKs (SRTP master keys) from a single TGK. For a media stream in SDP, MIKEY allocates two consecutive numbers for the crypto session IDs, so that each direction uses a different SRTP master key (see [RFC4567]).

The MIKEY specification [RFC3830] defines three modes to exchange keys, associated parameters and to protect the MIKEY message: pre-shared key, public-key encryption and Diffie-Hellman key exchange. In the first two modes the MIKEY initiator only chooses and distributes the TGK or TEK, whereas in the third mode both MIKEY entities (the initiator and responder) contribute to the keys. All three MIKEY modes have in common that for establishing a SRTP session the exchanged key is valid for the send and receive direction. Especially for group communications it is desirable to update the SRTP master key individually per direction. EKT provides this property by distributing the SRTP master key within the SRTP/SRTCP packet.

MIKEY already supports synchronization of ROC values between the MIKEY initiator and responder. The SSRC / ROC value pair is part of the MIKEY Common Header payload. This allows providing the current ROC value to late joiners of a session. However, in some scenarios a key management based ROC synchronization is not sufficient. For example, in mobile and wireless environments, members may go in and out of coverage and may miss a sequence number overrun. In point-to-multipoint translator scenarios it is desirable to not require the group controller to track the ROC values of each member, but to provide the ROC value by the originator of the SRTP packet. A better alternative to synchronize the ROC values is to send them directly via SRTP/SRTCP as EKT does. A separate SRTP extension [RFC4771] includes the ROC in a modified authentication tag but that extension does not support updating the SRTP master key.

Besides the ROC, MIKEY synchronizes also the SSRC values of the SRTP streams. Each sender of a stream sends the associated SSRC within the MIKEY message to the other party. If an SRTP session participant starts a new SRTP source (SSRC) or a new participant is added to a group, subsequent SDP offer/answer and MIKEY exchanges are necessary to update the SSRC values. EKT improves these scenarios by updating the keys and SSRC values without coordination on the signaling channel. With EKT, SRTP can handle early media, since the EKT SPI allows the receiver to identify the cryptographic keys and parameters used by the source.

The MIKEY specification [RFC3830] suggests the use of unicast for rekeying. This method does not scale well to large groups or interactive groups. The EKT extension of SRTP/SRTCP provides a solution for rekeying the SRTP master key and for ROC/SSRC synchronization. EKT is not a substitution for MIKEY, but rather a complementary addition to address the above described limitations of MIKEY.

In the next section we provide an extension to MIKEY for support of EKT. EKT can be used only with the pre-shared key or public-key encryption MIKEY mode of [RFC3830]. The Diffie-Hellman exchange mode is not suitable in conjunction with EKT, because it is not possible to establish one common EKT key over multiple EKT entities. Additional MIKEY modes specified in separate documents are not considered for EKT.

5.1. EKT Extensions to MIKEY

In order to use EKT with MIKEY, the EKT cipher, EKT key and EKT SPI is negotiated in the MIKEY message exchange.

The following parameters are added to the MIKEY Security Protocol Parameters namespace ([RFC3830], Section 6.10.1). (TBD will be requested from IANA [NOTE TO RFC EDITOR])

Type	Meaning	Possible values
TBD	EKT cipher	see below
TBD	EKT SPI	a 15-bit value

Figure 10: MIKEY Security Protocol Parameters

EKT cipher	Value
(reserved)	0
AESKW_128	1
AESKW_192	2
AESKW_256	3

Figure 11: EKT Cipher Parameters

EKT_Key is transported in the MIKEY KEMAC payload within one separate Key Data sub-payload. As specified in Section 6.2 of [RFC3830], the KEMAC payload carries the TEK Generation Key (TGK) or the Traffic Encryption Key (TEK). One or more TGKs or TEKs are carried in individual Key Data sub-payloads within the KEMAC payload. The KEMAC payload is encrypted as part of MIKEY. The Key Data sub- payload, specified in Section 6.13 of [RFC3830], has the following format:

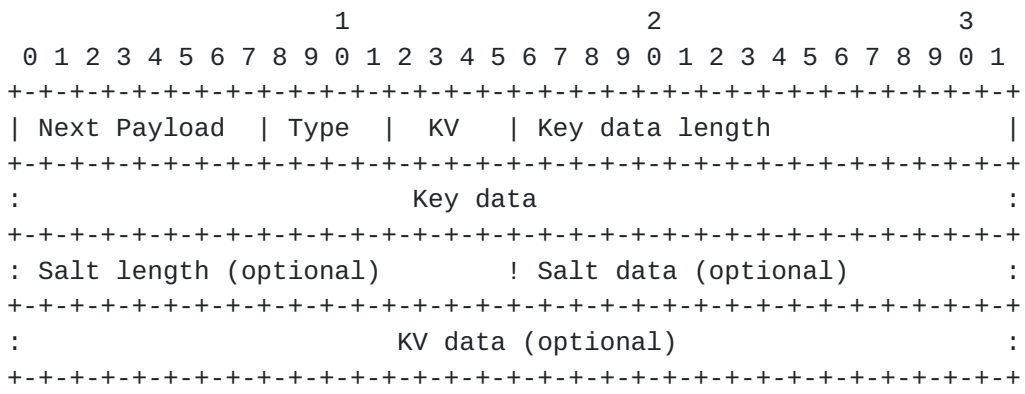

Figure 12: Key Data Sub-Payload of MIKEY

These fields are described below:

Type: 4 bits in length, indicates the type of key included in the payload. We define Type = TBD (will be requested from IANA [NOTE TO RFC EDITOR]) to indicate transport of the EKT key.

KV: (4 bits): indicates the type of key validity period specified. KV=1 is currently specified as an SPI. We use that value to indicate the KV data contains the EKT_SPI for the key type EKT_Key. KV data would be 16 bits in length, but it is also possible to interpret the length from the 'Key data len' field. KV data MUST be present for the key type EKT_Key when KV=1.

Salt length, Salt Data: These optional fields SHOULD be omitted for the key type EKT_Key, if the SRTP master salt is already present in the TGK or TEK Key Data sub-payload. The EKT_Key sub-payload MUST contain a SRTP master salt, if the SRTP master salt is not already present in the TGK or TEK Key Data sub-payload.

KV Data: length determined by Key Data Length field.

5.2. Offer/Answer Considerations

This section describes Offer/Answer considerations for the use of EKT together with MIKEY for unicast streams. The offerer and answerer MUST follow the procedures specified in [RFC3830] and [RFC4567] as well as the following ones.

5.2.1. Generating the Initial Offer

If it is intended to use MIKEY together with EKT, the offerer MUST include at least one MIKEY key-mgmt attribute with one EKT_Key Key Data sub-payload and the SRTP Security Policy payload (SP) with the policy parameter EKT SPI. The policy parameter EKT Cipher is

OPTIONAL, The default value is "AESKW_128" in accordance with Section 2.3.1. MIKEY can be used on session or media level. On session level, MIKEY provides the keys for multiple SRTP sessions in the SDP offer. The EKT SPI references a EKT parameter set including the Secure RTP parameters as specified in Section 8.2 in [RFC3711]. If MIKEY is used on session level, it is only possible to use one EKT SPI value. Therefore, the session-level MIKEY message MUST contain one SRTP Security Policy payload only, which is valid for all related SRTP media lines. If MIKEY is used on media level, different SRTP Security Policy parameters (and consequently different EKT SPI values) can be used for each media line. If MIKEY is used on session and media level, the media level content overrides the session level content.

EKT requires a single shared SRTP master salt between all participants in the direct SRTP session. If a MIKEY key-mgmt attribute contains more than one TGK or TEK Key Data sub-payload, all the sub-payloads MUST contain the same master salt value. Consequently, the EKT_Key Key Data sub-payload MAY also contain the same salt or MAY omit the salt value. If the SRTP master salt is not present in the TGK and TEK Key Data sub-payloads, the EKT_Key sub-payload MUST contain a master salt.

5.2.2. Generating the Initial Answer

For each media line in the offer using MIKEY, provided on session and/or on media level, the answerer examines the related MIKEY key-mgmt attributes for the presence of EKT parameters. In order to accept the offered key-mgmt attribute, the MIKEY message MUST contain one EKT_Key Key Data sub-payload and the SRTP Security Policy payload with policy parameter EKT SPI. The answerer examines also the existence of a SRTP master salt in the TGK/TEK and/or the EKT_Key sub-payloads. If multiple salts are available, all values MUST be equal. If the salt values differ or no salt is present, the key-mgmt attribute MUST be considered as invalid.

The MIKEY responder message in the SDP answer does not contain a MIKEY KEMAC or Security Policy payload and consequently does not contain any EKT parameters. If a key-mgmt attribute for a media line was accepted by the answerer, the EKT parameter set of the offerer is valid for both directions of the SRTP session.

5.2.3. Processing the Initial Answer

On reception of the answer, the offerer examines if EKT has been accepted for the offered media lines. If a MIKEY key-mgmt attribute is received containing a valid MIKEY responder message, EKT has been successfully negotiated. On receipt of a MIKEY error message, EKT

negotiation has failed. For example, this may happen if an EKT extended MIKEY initiator message is sent to a MIKEY entity not supporting EKT. A MIKEY error code 'Invalid SPpar' or 'Invalid DT' is returned to indicate that the EKT parameters (EKT Cipher and EKT SPI) in the SRTP Security Policy payload or the EKT_Key sub-payload is not supported. In this case, the offerer may send a second SDP offer with a MIKEY key-mgmt attribute without the additional EKT extensions.

This behavior can be improved by offering two key-mgmt SDP attributes. One attribute offers MIKEY with SRTP and EKT and the other attribute offers MIKEY with SRTP without EKT.

5.2.4. Modifying the Session

Once an SRTP stream has been established, a new offer/answer exchange can modify the session including the EKT parameters. If the EKT key or EKT cipher is modified (i.e., a new EKT parameter set is created) the offerer MUST also provide a new EKT SPI value. The offerer MUST NOT remap an existing EKT SPI value to a new EKT parameter set. Similar, a modification of the SRTP Security Policy leads to a new EKT parameter set and requires a fresh EKT SPI, even if the EKT key or cipher did not change.

Once EKT is being used, the SRTP master salt for the SRTP session MUST NOT be changed. The salt in the Key Data sub-payloads within the subsequent offers MUST be the same as the one already used.

After EKT has been successfully negotiated for a session and an SRTP master key has been transported by EKT, it is difficult to switch back to a pure MIKEY based key exchange in a synchronized way. Therefore, once EKT is being used for a session, EKT MUST be used also in all subsequent offer/answer exchanges for that session.

6. Using EKT for Interoperability between Key Management Systems

A media gateway (MGW) can provide interoperability between an SRTP-EKT endpoint and a non-EKT SRTP endpoint. When doing this function, the MGW can perform non-cryptographic transformations on SRTP packets outlined above. However, there are some uses of cryptography that will be required for that gateway. If a new SRTP master key is communicated to the MGW (via EKT from the EKT leg, or via Security Descriptions without EKT from the Security Descriptions leg), the MGW needs to convert that information for the other leg, and that process will incur some cryptographic operations. Specifically, if the new key arrived via EKT, the key must be decrypted and then sent in Security Descriptions (e.g., as a SIP re-INVITE); likewise, if a new

key arrives via Security Descriptions that must be encrypted via EKT and sent in SRTP/SRTCP.

Additional non-normative information can be found in Appendix A.

7. Design Rationale

From [RFC3550], a primary function of RTCP is to carry the CNAME, a "persistent transport-level identifier for an RTP source" since "receivers require the CNAME to keep track of each participant." EKT works in much the same way but uses SRTP to carry information needed for the proper processing of the SRTP traffic.

With EKT, SRTP gains the ability to synchronize the creation of cryptographic contexts across all of the participants in a single session. This feature provides some, but not all, of the functionality that is present in IKE phase two (but not phase one). Importantly, EKT does not provide a way to indicate SRTP options.

With EKT, external signaling mechanisms provide the SRTP options and the EKT Key, but need not provide the key(s) for each individual SRTP source. EKT provides a separation between the signaling mechanisms and the details of SRTP. The signaling system need not coordinate all SRTP streams, nor predict in advance how many sources will be present, nor communicate SRTP-level information (e.g., rollover counters) of current sessions.

EKT is especially useful for multi-party sessions, and for the case where multiple RTP sessions are sent to the same destination transport address (see the example in the definition of "RTP session" in [RFC3550]). A SIP offer that is forked in parallel (sent to multiple endpoints at the same time) can cause multiple RTP sessions to be sent to the same transport address, making EKT useful for use with SIP.

EKT can also be used in conjunction with a scalable group-key management system like GDOI [RFC6407]. In such a combination GDOI would provide a secure entity authentication method for group members, and a scalable way to revoke group membership; by itself, EKT does not attempt to provide either capability.

EKT carries the encrypted key in a new SRTP field (at the end of the SRTP packet). This maintains compatibility with the existing SRTP specification by defining a new crypto function that incorporates the encrypted key, and a new authentication transform to provide implicit authentication of the encrypted key.

The main motivation for the use of the variable-length EKT format is bandwidth conservation. When EKT is sent over SRTP, there will be a loss of (usable) bandwidth due to the additional EKT bytes in each RTP packet. For some applications, this bandwidth loss is significant.

7.1. Alternatives

In its current design, EKT requires that the Master Salt be established out of band. That requirement is undesirable. In an offer/answer environment, it forces the answerer to re-use the same Master Salt value used by the offerer. The Master Salt value could be carried in EKT packets though that would consume yet more bandwidth.

In some scenarios, two SRTP sessions may be combined into a single session. When using EKT in such sessions, it is desirable to have an SPI value that is larger than 15 bits, so that collisions between SPI values in use in the two different sessions are unlikely (since each collision would confuse the members of one of the sessions).

An alternative that addresses both of these needs is as follows: the SPI value can be lengthed from 15 bits to 63 bits, and the Master Salt can be identical to, or constructed from, the SPI value. SRTP conventionally uses a 14-byte Master Salt, but shorter values are acceptable. This alternative would add six bytes to each EKT packet; that overhead may be a reasonable tradeoff for addressing the problems outlined above. This is considered too high a bandwidth penalty.

8. Security Considerations

EKT inherits the security properties of the SRTP keying it uses: Security Descriptions, DTLS-SRTP, or MIKEY.

With EKT, each SRTP sender and receiver MUST generate distinct SRTP master keys. This property avoids any security concern over the re-use of keys, by empowering the SRTP layer to create keys on demand. Note that the inputs of EKT are the same as for SRTP with key-sharing: a single key is provided to protect an entire SRTP session. However, EKT remains secure even in the absence of out-of-band coordination of SSRCs, and even when SSRC values collide.

The EKT Cipher includes its own authentication/integrity check. For an attacker to successfully forge a full EKT packet, it would need to defeat the authentication mechanisms of both the EKT Cipher and the SRTP authentication mechanism.

The presence of the SSRC in the EKT_Plaintext ensures that an attacker cannot substitute an EKT_Ciphertext from one SRTP stream into another SRTP stream.

An attacker who strips a Full_EKT_Field from an SRTP packet may prevent the intended receiver of that packet from being able to decrypt it. This is a minor denial of service vulnerability. Similarly, an attacker who adds a Full_EKT_Field can disrupt service.

An attacker could send packets containing either Short EKT Field or Full EKT Field, in an attempt to consume additional CPU resources of the receiving system. In the case of the Short EKT Field, this field is stripped and normal SRTP or SRTCP processing is performed. In the case of the Full EKT Field, the attacker would have to have guessed or otherwise determined the SPI being used by the receiving system. If an invalid SPI is provided by the attacker, processing stops. If a valid SPI is provided by the attacker, the receiving system will decrypt the EKT ciphertext and return an authentication failure (Step 3 of Section 2.2.2).

EKT can rekey an SRTP stream until the SRTP rollover counter (ROC) needs to roll over. EKT does not extend SRTP's rollover counter (ROC), and like SRTP itself EKT cannot properly handle a ROC rollover. Thus even if using EKT, new (master or session) keys need to be established after 2^48 packets are transmitted in a single SRTP stream as described in Section 3.3.1 of [RFC3711]. Due to the relatively low packet rates of typical RTP sessions, this is not expected to be a burden.

The confidentiality, integrity, and authentication of the EKT cipher MUST be at least as strong as the SRTP cipher.

Part of the EKT_Plaintext is known, or easily guessable to an attacker. Thus, the EKT Cipher MUST resist known plaintext attacks. In practice, this requirement does not impose any restrictions on our choices, since the ciphers in use provide high security even when much plaintext is known.

An EKT cipher MUST resist attacks in which both ciphertexts and plaintexts can be adaptively chosen. An EKT cipher MUST resist attacks in which both ciphertexts and plaintexts can be adaptively chosen and adversaries that can query both the encryption and decryption functions adaptively.

9. IANA Considerations

IANA is requested to register EKT from Section 3.9 into the Session Description Protocol (SDP) Security Descriptions [iana-sdp-sdesc] registry for "SRTP Session Parameters".

IANA is requested to register the following new attributes into the SDP Attributes registry [iana-sdp-attr].

Attribute name: dtls-srtp-ekt

Long form name: DTLS-SRTP with EKT

Type of attribute: Media-level

Subject to charset: No

Purpose: Indicates support for DTLS-SRTP with EKT

Appropriate values: No values

Contact name: Dan Wing, dwing@cisco.com

We request the following IANA assignments from the existing [iana-mikey] name spaces in the IETF consensus range (0-240) [RFC3830]:

- From the Key Data payload name spaces, a value to indicate the type as the 'EKT_Key'.

Furthermore, we need the following two new IANA registries created, populated with the initial values in this document. New values for both of these registries can be defined via Specification Required [RFC5226].

- EKT parameter type, initially populated with the list from Figure 10

- EKT cipher, initially populated with the list from Figure 11

10. Acknowledgements

Thanks to Lakshminath Dondeti for assistance with earlier versions of this document. Thanks to Kai Fischer for writing the MIKEY section.

Thanks to Nermeen Ismail, Eddy Lem,Rob Raymond, and Yi Cheng for fruitful discussions and comments. Thanks to Felix Wyss for his review and comments regarding ciphers. Thanks to Michael Peck for

his review. Thanks to Magnus Westerlund for his review. Thanks to Michael Peck and Jonathan Lennox for their review comments.

11. References

11.1. Normative References

[FIPS197] National Institute of Standards and Technology (NIST), "The Advanced Encryption Standard (AES)", FIPS-197 Federal Information Processing Standard, November 2001.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC4563] Carrara, E., Lehtovirta, V., and K. Norrman, "The Key ID Information Type for the General Extension Payload in Multimedia Internet KEYing (MIKEY)", RFC 4563, June 2006.

[RFC4567] Arkko, J., Lindholm, F., Naslund, M., Norrman, K., and E. Carrara, "Key Management Extensions for Session Description Protocol (SDP) and Real Time Streaming Protocol (RTSP)", RFC 4567, July 2006.

[RFC4568] Andreasen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", RFC 4568, July 2006.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, May 2010.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

11.2. Informative References

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3830] Arkko, J., Carrara, E., Lindholm, F., Naslund, M., and K. Norrman, "MIKEY: Multimedia Internet KEYing", RFC 3830, August 2004.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

[RFC4771] Lehtovirta, V., Naslund, M., and K. Norrman, "Integrity Transform Carrying Roll-Over Counter for the Secure Real-time Transport Protocol (SRTP)", RFC 4771, January 2007.

[RFC5649] Housley, R. and M. Dworkin, "Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm", RFC 5649, September 2009.

[RFC6407] Weis, B., Rowles, S., and T. Hardjono, "The Group Domain of Interpretation", RFC 6407, October 2011.

[iana-mikey] IANA, , "Multimedia Internet KEYing (Mikey) Payload Name Spaces", 2011, <http://www.iana.org/assignments/mikey-payloads/mikey-payloads.xhtml>.

[iana-sdp-attr] IANA, , "SDP Parameters", 2011, <http://www.iana.org/assignments/sdp-parameters/sdp-parameters.xml>.

[iana-sdp-sdesc]
IANA, , "Session Description Protocol (SDP) Security Descriptions: SRTP Session Parameters", 2011, <http://www.iana.org/assignments/sdp-security-descriptions/sdp-security-descriptions.xml#sdp-security-descriptions-4>.

Appendix A. Using EKT to Optimize Interworking DTLS-SRTP with Security Descriptions

Today, SDP Security Descriptions [RFC4568] is used for distributing SRTP keys in several different IP PBX systems. The IP PBX systems are typically used within a single enterprise. A Session Border Controller is a reasonable solution to interwork between Security Descriptions in one network and DTLS-SRTP in another network. For example, a mobile operator (or an Enterprise) could operate Security Descriptions within their network and DTLS-SRTP towards the Internet.

However, due to the way Security Descriptions and DTLS-SRTP manage their SRTP keys, such an SBC has to authenticate, decrypt, re-encrypt, and re-authenticate the SRTP (and SRTCP) packets in one direction, as shown in Figure 13, below. This is computationally expensive.

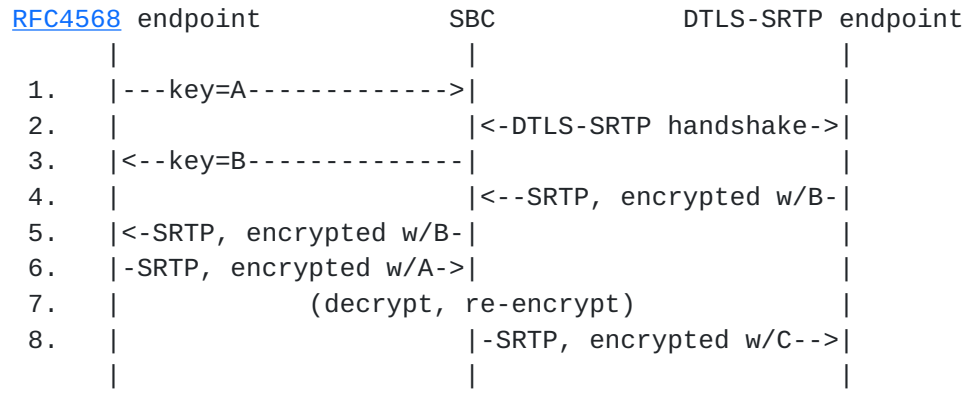

Figure 13: Interworking Security Descriptions and DTLS-SRTP

The message flow is as follows (similar steps occur with SRTCP):

1. The Security Descriptions [RFC4568] endpoint discloses its SRTP key to the SBC, using a=crypto in its SDP.

2. SBC completes DTLS-SRTP handshake. From this handshake, the SBC derives the SRTP key for traffic from the DTLS-SRTP endpoint (key B) and to the DTLS-SRTP endpoint (key C).

3. The SBC communicates the SRTP encryption key (key B) to the Security Descriptions endpoint (using a=crypto). (There is no way, with DTLS-SRTP, to communicate the Security Descriptions key to the DTLS-SRTP key endpoint.)

4. The DTLS-SRTP endpoint sends an SRTP key, encrypted with its key B. This is received by the SBC.

5. The received SRTP packet is simply forwarded; the SBC does not need to do anything with this packet as its key (key B) was already communicated in step 3.

6. The Security Descriptions endpoint sends an SRTP packet, encrypted with its key A.

7. The SBC has to authenticate and decrypt the SRTP packet (using key A), and re-encrypt it and generate an HMAC (using key C).

8. The SBC sends the new SRTP packet.

If EKT is deployed on the DTLS-SRTP endpoints, EKT helps to avoid the computationally expensive operation so the SBC does not need to perform any per-packet operations on the SRTP (or SRTCP) packets in either direction. With EKT the SBC can simply forward the SRTP (and SRTCP) packets in both directions without per-packet HMAC or cryptographic operations.

To accomplish this interworking, DTLS-SRTP EKT must be supported on the DTLS-SRTP endpoint, which allows the SBC to transport the Security Description key to the EKT endpoint and send the DTLS-SRTP key to the Security Descriptions endpoint. This works equally well for both incoming and outgoing calls. An abbreviated message flow is shown in Figure 14, below.

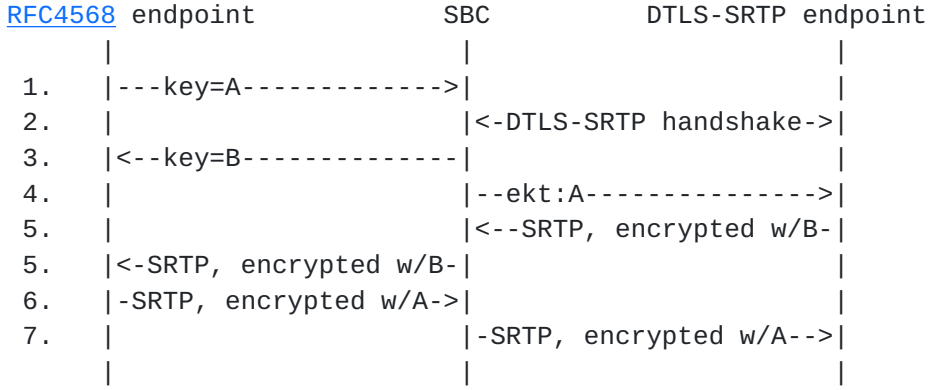

Figure 14: Interworking Security Descriptions and EKT

The message flow is as follows (similar steps occur with SRTCP):

1. Security Descriptions endpoint discloses its SRTP key to the SBC (a=crypto).

2. SBC completes DTLS-SRTP handshake. From this handshake, the SBC derives the SRTP key for traffic from the DTLS-SRTP endpoint (key B) and to the DTLS-SRTP endpoint (key C).

3. The SBC communicates the SRTP encryption key (key B) to the Security Descriptions endpoint.

4. The SBC sends an EKT packet indicating that SRTP will be encrypted with 'key A' towards the DTLS-SRTP endpoint.

5. The DTLS-SRTP endpoint sends an SRTP key, encrypted with its key B. This is received by the SBC.

6. The received SRTP packet is simply forwarded; the SBC does not need to do anything with this packet as its key (key B) was communicated in step 3.

7. The Security Descriptions endpoint sends an SRTP packet, encrypted with its key A.

8. The received SRTP packet is simply forwarded; the SBC does not need to do anything with this packet as its key (key A) was communicated in step 4.

Authors' Addresses

John Mattsson (editor)
Ericsson AB
SE-164 80 Stockholm
Sweden

Phone: +46 10 71 43 501
Email: john.mattsson@ericsson.com

David A. McGrew
Cisco Systems, Inc.
510 McCarthy Blvd.
Milpitas, CA 95035
US

Phone: (408) 525 8651
Email: mcgrew@cisco.com
URI: http://www.mindspring.com/~dmcgrew/dam.htm

Dan Wing
Cisco Systems, Inc.
510 McCarthy Blvd.
Milpitas, CA 95035
US

Phone: (408) 853 4197
Email: dwing@cisco.com

Flemming Andreason
Cisco Systems, Inc.
499 Thornall Street
Edison, NJ 08837
US

Email: fandreas@cisco.com