Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2018

C. Jennings
Cisco Systems
J. Mattsson
Ericsson AB
D. McGrew
D. Wing
F. Andreason
Cisco Systems
March 5, 2018

Encrypted Key Transport for DTLS and Secure RTP
draft-ietf-perc-srtp-ekt-diet-07

Abstract

Encrypted Key Transport (EKT) is an extension to DTLS and Secure Real-time Transport Protocol (SRTP) that provides for the secure transport of SRTP master keys, rollover counters, and other information within SRTP. This facility enables SRTP for decentralized conferences by distributing a common key to all of the conference endpoints.

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 https://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 September 6, 2018.

Table of Contents

1. Introduction

Real-time Transport Protocol (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 synchronization source (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 along with the SSRC, rollover counter (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 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 and reduces the amount of external signaling control that is needed in a SRTP session with multiple receivers. EKT securely distributes the SRTP master key and other information for each SRTP source. With this method, SRTP entities are free to choose SSRC values as they see fit, and to start up new SRTP sources with new SRTP master keys (see Section 2.2) within a session without coordinating with other entities via external signaling or other external means.

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 furnishes the information needed by the receiver to instantiate an SRTP/SRTCP receiver context.

EKT can be used in conferences where the central media distributor or conference bridge can not decrypt the media, such as the type defined for [I-D.ietf-perc-private-media-framework]. It can also be used for large scale conferences where the conference bridge or media distributor can decrypt all the media but wishes to encrypt the media it is sending just once then send the same encrypted media to a large number of participants. This reduces the amount of CPU time needed for encryption and can be used for some optimization to media sending that use source specific multicast.

EKT does not control the manner in which the SSRC is generated; it is only concerned with their secure transport.

EKT is not intended to replace external key establishment mechanisms. Instead, it is used in conjunction with those methods, and it

relieves those methods of the burden to deliver the context for each SRTP source to every SRTP participant.

2. Overview

This specification defines a way for the server in a DTLS-SRTP negotiation to provide an ekt_key to the client during the DTLS handshake. This ekt_key can be used to encrypt the SRTP master key used to encrypt the media the endpoint sends. This specification also defines a way to send the encrypted SRTP master key along with the SRTP packet. Endpoints that receive this and know the ekt_key can use the ekt_key to decrypt the SRTP master key then use the SRTP master key to decrypt the SRTP packet.

One way to use this is used is described in the architecture defined by [I-D.ietf-perc-private-media-framework]. Each participants in the conference call forms a DTLS-SRTP connection to a common key distributor that gives all the endpoints the same ekt_key. Then each endpoint picks there own SRTP master key for the media they send. When sending media, the endpoint also includes the SRTP master key encrypted with the ekt_key. This allows all the endpoints to decrypt the media.

3. 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].

4. Encrypted Key Transport

EKT defines a new method of providing SRTP master keys to an endpoint. In order to convey the ciphertext corresponding to the SRTP master key, and other additional information, an additional EKT field is added to SRTP packets. The EKT field appears at the end of the SRTP packet. The EKT field appears after the optional authentication tag if one is present, otherwise the EKT field appears after the ciphertext portion of the packet.

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. Senders MUST NOT include MKI when using EKT. Receivers SHOULD simply ignore any MKI field received if EKT is in use.

4.1. EKT Field Formats

The EKT Field uses the format defined below for the FullEKTField and ShortEKTField.

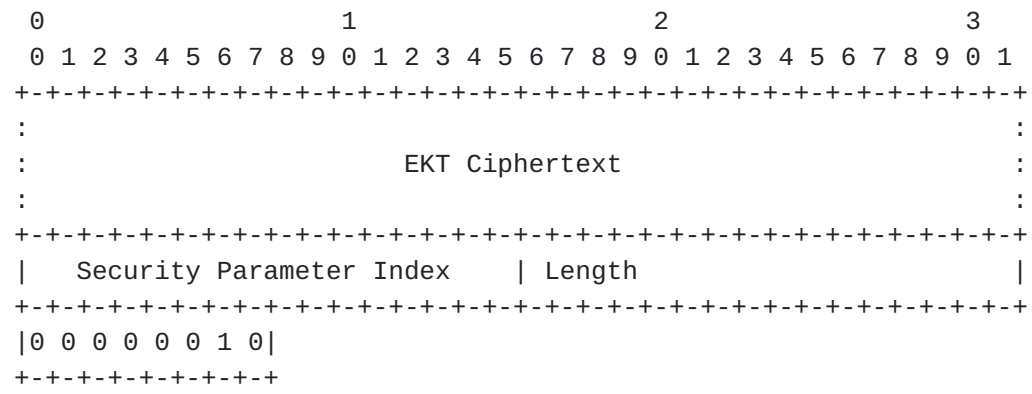


Figure 1: Full EKT Field format

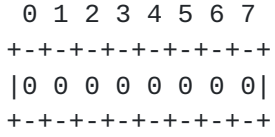


Figure 2: Short EKT Field format

The following shows the syntax of the EKTField expressed in ABNF [RFC5234]. The EKTField is added to the end of an SRTP or SRTCP packet. The EKTPlaintext is the concatenation of SRTPMasterKeyLength, SRTPMasterKey, SSRC, and ROC in that order. The EKTCiphertext is computed by encrypting the EKTPlaintext using the EKTKey. Future extensions to the EKTField MUST conform to the syntax of ExtensionEKTField.

```
BYTE = %x00-FF

EKTMsgTypeFull = %x02
EKTMsgTypeShort = %x00
EKTMsgTypeExtension = %x03-FF

EKTMsgLength = 2BYTE;

SRTPMasterKeyLength = BYTE
SRTPMasterKey = 1*256BYTE
SSRC = 4BYTE; SSRC from RTP
ROC = 4BYTE ; ROC from SRTP FOR THE GIVEN SSRC

EKTPlaintext = SRTPMasterKeyLength SRTPMasterKey SSRC ROC

EKTCiphertext = 1*256BYTE ; EKTEncrypt(EKTKey, EKTPlaintext)
SPI = 2BYTE

FullEKTField = EKTCiphertext SPI EKTMsgLength EKTMsgTypeFull

ShortEKTField = EKTMsgTypeShort

ExtensionData = 1*1024BYTE
ExtensionEKTField = ExtensionData EKTMsgLength EKTMsgTypeExtension

EKTField = FullEKTField / ShortEKTField / ExtensionEKTField
```

Figure 3: EKTField Syntax

These fields and data elements are defined as follows:

EKTPlaintext: 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. This is the concatenation of the SRTP Master Key, the SSRC, and the ROC.

EKTCiphertext: The data that is output from the EKT encryption operation, described in Section 4.4. This field is included in SRTP packets when EKT is in use. The length of EKTCiphertext can be larger than the length of the EKTPlaintext that was encrypted.

SRTPMasterKey: On the sender side, the SRTP Master Key associated with the indicated SSRC.

SRTPMasterKeyLength: The length of the SRTPMasterKey in bytes. This depends on the cipher suite negotiated for SRTP using SDP Offer/ Answer [RFC3264] for the SRTP.

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

Security Parameter Index (SPI): This field indicates the appropriate EKT Key and other parameters for the receiver to use when processing the packet. The length of this field is 16 bits. The parameters identified by this field are:

- The EKT cipher used to process the packet.
- The EKT Key used to process the packet.
- The SRTP Master Salt associated with any Master Key encrypted with this EKT Key. The Master Salt is communicated separately, via signaling, typically along with the EKTKey.

Together, these data elements are called an EKT parameter set. Each distinct EKT parameter set that is used MUST be associated with a distinct SPI value to avoid ambiguity.

EKTMsgLength: All EKT message other that ShortEKTField have a length as second from the last element. This is the length in octets of either the FullEKTField/ExtensionEKTField including this length field and the following message type.

Message Type: The last byte is used to indicate the type of the EKTField. This MUST be 2 in the FullEKTField format and 0 in ShortEKTField format. Values less than 64 are mandatory to understand while other values are optional to understand. A receiver SHOULD discard the whole EKTField if it contains any message type value that is less than 64 and that is not understood. Message type values that are 64 or greater but not implemented or understood can simply be ignored.

4.2. Packet Processing and State Machine

At any given time, each SRTP/SRTCP source 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 for that SSRC. 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 the received EKT parameter sets SHOULD be stored by all of the participants in an SRTP session, for use in processing inbound SRTP and SRTCP traffic.

Either the FullEKTField or ShortEKTField is appended at the tail end of all SRTP packets. The decision on which to send is specified in Section 4.7.

4.2.1. Outbound Processing

See Section 4.7 which describes when to send an SRTP packet with a FullEKTField. If a FullEKTField is not being sent, then a ShortEKTField is sent so the receiver can correctly determine how to process the packet.

When an SRTP packet is sent with a FullEKTField, the EKTField for that packet is created as follows, or uses an equivalent set of steps. The creation of the EKTField MUST precede the normal SRTP packet processing.

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

2. The EKTPlaintext field is computed from the SRTP Master Key, SSRC, and ROC fields, as shown in Section 4.1. The ROC, SRTP Master Key, and SSRC used in EKT processing SHOULD be the same as the one used in the SRTP processing.

3. The EKTCiphertext field is set to the ciphertext created by encrypting the EKTPlaintext with the EKT cipher, using the EKTKey as the encryption key. The encryption process is detailed in Section 4.4.

4. Then the FullEKTField is formed using the EKTCiphertext and the SPI associated with the EKTKey used above. Also appended are the Length and Message Type using the FullEKTField format.

   * Note: the value of the EKTCiphertext field is identical in successive packets protected by the same EKTKey and SRTP master key. This value MAY be cached by an SRTP sender to minimize computational effort.

   The computed value of the FullEKTField is written into the packet.

When a packet is sent with the Short EKT Field, the ShortEKFField is simply appended to the packet.

4.2.2. Inbound Processing

When receiving a packet on a RTP stream, the following steps are applied for each received packet.

1. The final byte is checked to determine which EKT format is in use. When an SRTP or SRTCP packet contains a ShortEKTField, the ShortEKTField is removed from the packet then normal SRTP or SRTCP processing occurs. If the packet contains a FullEKTField, then processing continues as described below. The reason for using the last byte of the packet to indicate the type is that the length of the SRTP or SRTCP part is not known until the decryption has occurred. At this point in the processing, there is no easy way to know where the EKT field would start. However, the whole UDP packet has been received so instead of the starting at the front of the packet, the parsing works backwards off the end of the packet and thus the type is put at the very end of the packet.

2. The Security Parameter Index (SPI) field is used to find which EKT parameter set to be used when processing the packet. 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. The EKT parameter set contains the EKTKey, EKTCipher, and SRTP Master Salt.

3. The EKTCiphertext authentication is checked and it is decrypted, as described in Section 4.4, using the EKTKey and EKTCipher found in the previous step. If the EKT decryption operation returns an authentication failure, then the packet processing stops.

4. The resulting EKTPlaintext is parsed as described in Section 4.1, to recover the SRTP Master Key, SSRC, and ROC fields. The SRTP Master Salt that is associated with the EKTKey is also retrieved. If the value of the srtp_master_salt sent as part of the EKTkey is longer than needed by SRTP, then it is truncated by taking the first N bytes from the srtp_master_salt field.

5. If the SSRC in the EKTPlaintext does not match the SSRC of the SRTP packet, then all the information from this EKTPlaintext MUST be discarded and the following steps in this list are not done.

6. The SRTP Master Key, ROC, and SRTP Master Salt from the previous step are saved in a map indexed by the SSRC found in the EKTPlaintext and can be used for any future crypto operations on

the inbound packets with that SSRC. If the SRTP Master Key recovered from the EKTPlaintext is longer than needed by SRTP transform in use, the first bytes are used. If the SRTP Master Key recovered from the EKTPlaintext is shorter than needed by SRTP transform in use, then the bytes received replace the first bytes in the existing key but the other bytes after that remain the same as the old key. This allows for replacing just half the key for transforms such as [I-D.ietf-perc-double]. Outbound packets SHOULD continue to use the old SRTP Master Key for 250 ms after sending any new key. This gives all the receivers in the system time to get the new key before they start receiving media encrypted with the new key.

7. At this point, EKT processing has successfully completed, and the normal SRTP or SRTCP processing takes place including replay protection.

4.3. Implementation Notes

The value of the EKTCiphertext field is identical in successive packets protected by the same EKT parameter set and the same SRTP master key, and ROC. 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 the above steps.

The receiver may want to have a sliding window to retain old SRTP master keys (and related context) for some brief period of time, so that out of order packets can be processed as well as packets sent during the time keys are changing.

When receiving a new EKTKey, implementations need to use the ekt_ttl to create a time after which this key cannot be used and they also need to create a counter that keeps track of how many times the keys has been used to encrypt data to ensure it does not exceed the T value for that cipher. If either of these limits are exceeded, the key can no longer be used for encryption. At this point implementation need to either use the call signaling to renegotiation a new session or need to terminate the existing session. Terminating the session is a reasonable implementation choice because these limits should not be exceeded except under an attack or error condition.

4.4. Ciphers

EKT uses an authenticated cipher to encrypt and authenticate the EKTPlaintext. This specification defines the interface to the cipher, in order to abstract the interface away from the details of that function. This specification also defines the default cipher

that is used in EKT. The default cipher described in Section 4.4.1 MUST be implemented, but another cipher that conforms to this interface MAY be used.

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

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

o 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 may be larger than 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. The EKTKey MUST NOT be used for encryption more that T times. Note that if the same FullEKTField is retransmitted 3 times, that only counts as 1 encryption.

Security requirements for EKT ciphers are discussed in Section 6.

4.4.1. Ciphers

The default EKT Cipher is the Advanced Encryption Standard (AES) 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 + (M mod 8) + 8 octets. It can be used with key sizes of L = 16, and L = 32 octets, and its use with those key sizes is indicated as AESKW128, or AESKW256, respectively. The key size determines the length of the AES key used by the Key Wrap algorithm. With this cipher, T=2^48.

Cipher	L	T
AESKW128	16	2^48
AESKW256	32	2^48

Table 1: EKT Ciphers

As AES-128 is the mandatory to implement transform in SRTP, AESKW128 MUST be implemented for EKT and AESKW256 MAY be implemented.

4.4.2. Defining New EKT Ciphers

Other specifications may extend this document by defining other EKTCiphers as described in Section 7. This section defines how those ciphers interact with this specification.

An EKTCipher determines how the EKTCiphertext 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, and T MUST be defined by each EKTCipher. The cipher MUST provide integrity protection.

4.5. Synchronizing Operation

If a source has its EKTKey changed by the key management, it MUST also change its SRTP master key, which will cause it to send out a new FullEKTField. This ensures that if key management thought the EKTKey needs changing (due to a participant leaving or joining) and communicated that 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 EKTKey.

4.6. Transport

EKT SHOULD be used over SRTP, and other specification MAY define how to use it 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.

4.7. Timing and Reliability Consideration

A system using EKT learns the SRTP master keys distributed with FullEKTFields sent with the SRTP, rather than with call signaling. A

receiver can immediately decrypt an SRTP packet, provided the SRTP packet 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.

The three cases are:

New sender:
A new sender SHOULD send a packet containing the FullEKTField as soon as possible, always before or coincident with sending its initial SRTP packet. To accommodate packet loss, it is RECOMMENDED that three consecutive packets contain the Full EKT Field be transmitted.

Rekey:
By sending EKT over SRTP, the rekeying event shares fate with the SRTP packets protected with that new SRTP master key. To accommodate packet loss, it is RECOMMENDED that three consecutive packets contain the FullEKTField be transmitted.

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 FullEKTField. 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 FullEKT Field. If sending audio and video, the RECOMMENDED frequency is the same as the rate of intra coded video frames. If only sending audio, the RECOMMENDED frequency is every 100ms.

5. Use of EKT with DTLS-SRTP

This document defines an extension to DTLS-SRTP called SRTP EKT Key Transport which enables secure transport of EKT keying material from the DTLS-SRTP peer in the server role to the client. This allows 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, which has strong authentication of the endpoint and flexibility, along with allowing secure multiparty RTP with loose coordination and efficient communication of per-source keys.

5.1. DTLS-SRTP Recap

DTLS-SRTP [RFC5764] uses an extended DTLS exchange between two peers to exchange keying material, algorithms, and parameters 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.

5.2. SRTP EKT Key Transport Extensions to DTLS-SRTP

This document defines a new TLS negotiated extension supported_ekt_ciphers and a new TLS handshake message type ekt_key. The extension negotiates the cipher to be used in encrypting and decrypting EKTCiphertext values, and the handshake message carries the corresponding key.

The diagram below shows a message flow of DTLS 1.3 client and server using EKT configured using the DTLS extensions described in this section. (The initial cookie exchange and other normal DTLS messages are omitted.)

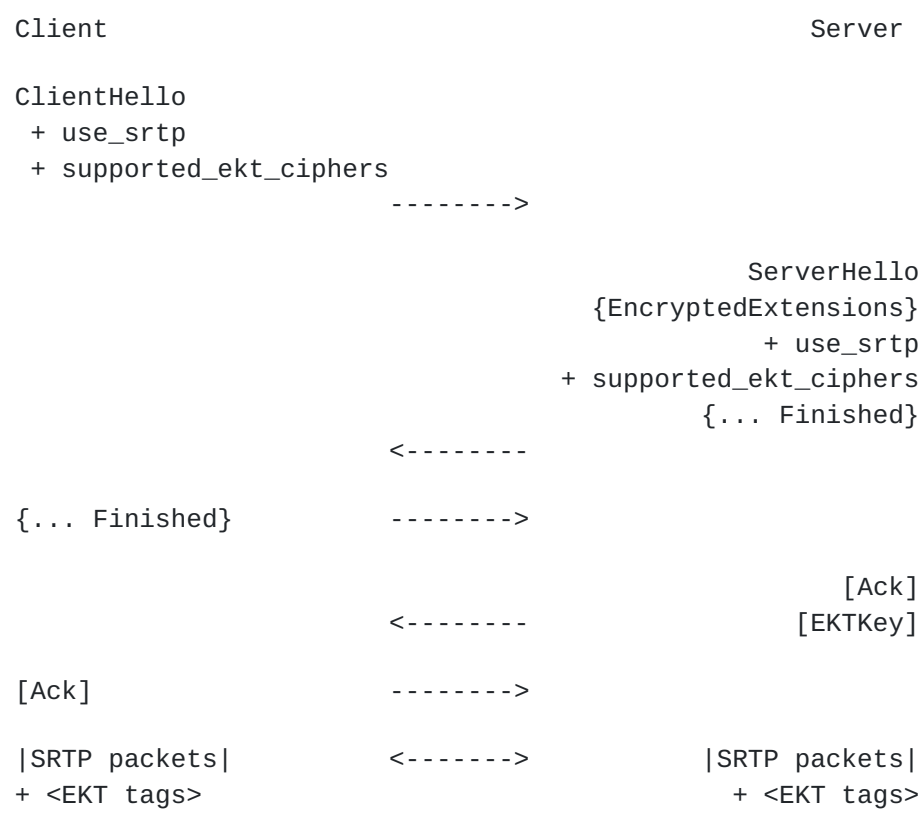
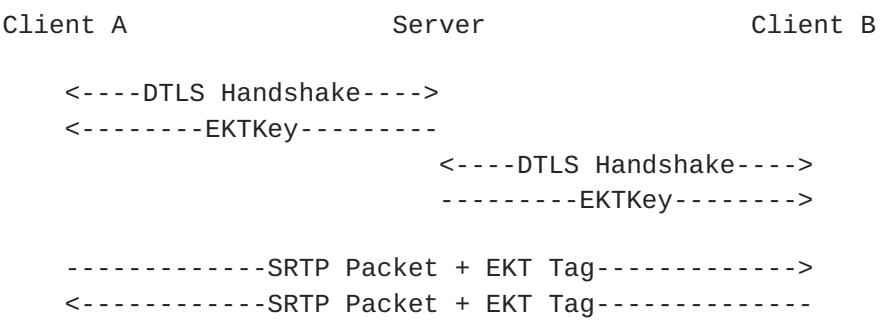


Figure 4

In the context of a multi-party SRTP session in which each endpoint performs a DTLS handshake as a client with a central DTLS server, the extensions defined in this session allow the DTLS server to set a common EKT key among all participants. Each endpoint can then use EKT tags encrypted with that common key to inform other endpoint of the keys it is using to protect SRTP packet. This avoids the need for many individual DTLS handshakes among the endpoints, at the cost of preventing endpoints from directly authenticating one another.



5.2.1. Negotiating an EKT Cipher

To indicate its support for EKT, a DTLS-SRTP client includes in its ClientHello an extension of type supported_ekt_ciphers listing the EKT ciphers the client supports in preference order, with the most preferred version first. If the server agrees to use EKT, then it includes a supported_ekt_ciphers extension in its ServerHello containing a cipher selected from among those advertised by the client.

The extension_data field of this extension contains an "EKTCipher" value, encoded using the syntax defined in [RFC5246]:

```
enum {
  reserved(0),
  aeskw_128(1),
  aeskw_256(2),
} EKTCipherType;

struct {
    select (Handshake.msg_type) {
        case client_hello:
            EKTCipherType supported_ciphers<1..255>;

        case server_hello:
            EKTCipherType selected_cipher;
    };
} EKTCipher;
```

5.2.2. Establishing an EKT Key

Once a client and server have concluded a handshake that negotiated an EKT cipher, the server MUST provide to the client a key to be used when encrypting and decrypting EKTCiphertext values. EKT keys are sent in encrypted handshake records, using handshake type ekt_key(TBD). The body of the handshake message contains an EKTKey structure:

[[NOTE: RFC Editor, please replace "TBD" above with the code point assigned by IANA]]

```
struct {
  opaque ekt_key_value<1..256>;
  opaque srtp_master_salt<1..256>;
  uint16 ekt_spi;
  uint24 ekt_ttl;
} EKTKey;
```

The contents of the fields in this message are as follows:

ekt_key_value
: The EKT Key that the recipient should use when generating EKTCiphertext values

srtp_master_salt
: The SRTP Master Salt to be used with any Master Key encrypted with this EKT Key

ekt_spi
: The SPI value to be used to reference this EKT Key and SRTP Master Salt in EKT tags (along with the EKT cipher negotiated in the handshake)

ekt_ttl
: The maximum amount of time, in seconds, that this EKT Key can be used. The ekt_key_value in this message MUST NOT be used for encrypting or decrypting information after the TTL expires.

If the server did not provide a supported_ekt_ciphers extension in its ServerHello, then EKTKey messages MUST NOT be sent by either the client or the server.

When an EKTKey is received and processed successfully, the recipient MUST respond with an Ack handshake message as described in Section 7 of [I-D.ietf-tls-dtls13]. The EKTKey message and Ack must be retransmitted following the rules in Section 4.2.4 of [RFC6347].

Note: To be clear, EKT can be used with versions of DTLS prior to 1.3. The only difference is that in a pre-1.3 TLS stacks will not have built-in support for generating and processing Ack messages.

If an EKTKey message is received that cannot be processed, then the recipient MUST respond with an appropriate DTLS alert.

5.3. Offer/Answer Considerations

When using EKT with DTLS-SRTP, the negotiation to use EKT is done at the DTLS handshake level and does not change the [RFC3264] Offer / Answer messaging.

5.4. Sending the DTLS EKTKey Reliably

The DTLS EKTKey message is sent using the retransmissions specified in Section 4.2.4. of DTLS [RFC6347]. Retransmission is finished with an Ack message or an alert is received.

6. Security Considerations

EKT inherits the security properties of the DTLS-SRTP (or other) keying it uses.

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 when SSRC values collide.

SRTP master keys MUST be randomly generated, and [RFC4086] offers some guidance about random number generation. SRTP master keys MUST NOT be re-used for any other purpose, and SRTP master keys MUST NOT be derived from other SRTP master keys.

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

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

An attacker who tampers with the bits in FullEKTField can prevent the intended receiver of that packet from being able to decrypt it. This is a minor denial of service vulnerability. Similarly the attacker could take an old FullEKTField from the same session and attach it to the packet. The FullEKTField would correctly decode and pass integrity but the key extracted from the FullEKTField , when used to decrypt the SRTP payload, would be wrong and the SRTP integrity check would fail. Note that the FullEKTField only changes the decryption key and does not change the encryption key. None of these are

considered significant attacks as any attacker that can modify the packets in transit and cause the integrity check to fail.

An attacker could send packets containing a Full EKT Field, in an attempt to consume additional CPU resources of the receiving system by causing the receiving system will decrypt the EKT ciphertext and detect an authentication failure. In some cases, caching the previous values of the Ciphertext as described in Section 4.3 helps mitigate this issue.

Each EKT cipher specifies a value T that is the maximum number of times a given key can be used. An endpoint MUST NOT encrypt more than T different Full EKT Field using the same EKTKey. In addition, the EKTKey MUST NOT be used beyond the lifetime provided by the TTL described in Figure 4.

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

Part of the EKTPlaintext 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 and adversaries that can query both the encryption and decryption functions adaptively.

In some systems, when a member of a conference leaves the conferences, the conferences is rekeyed so that member no longer has the key. When changing to a new EKTKey, it is possible that the attacker could block the EKTKey message getting to a particular endpoint and that endpoint would keep sending media encrypted using the old key. To mitigate that risk, the lifetime of the EKTKey SHOULD be limited using the ekt_ttl.

7. IANA Considerations

7.1. EKT Message Types

IANA is requested to create a new table for "EKT Messages Types" in the "Real-Time Transport Protocol (RTP) Parameters" registry. The initial values in this registry are:

Message Type	Value	Specification
Short	0	RFCAAAA
Full	2	RFCAAAA
Reserved	63	RFCAAAA
Reserved	255	RFCAAAA

Table 2: EKT Messages Types

Note to RFC Editor: Please replace RFCAAAA with the RFC number for this specification.

New entries to this table can be added via "Specification Required" as defined in [RFC8126]. When requesting a new value, the requestor needs to indicate if it is mandatory to understand or not. If it is mandatory to understand, IANA needs to allocate a value less than 64, if it is not mandatory to understand, a value greater than or equal to 64 needs to be allocated. IANA SHOULD prefer allocation of even values over odd ones until the even code points are consumed to avoid conflicts with pre standard versions of EKT that have been deployed.

All new EKT messages MUST be defined to have a length as second from the last element.

7.2. EKT Ciphers

IANA is requested to create a new table for "EKT Ciphers" in the "Real-Time Transport Protocol (RTP) Parameters" registry. The initial values in this registry are:

Name	Value	Specification
AESKW128	1	RFCAAAA
AESKW256	2	RFCAAAA
Reserved	255	RFCAAAA

Table 3: EKT Cipher Types

Note to RFC Editor: Please replace RFCAAAA with the RFC number for this specification.

New entries to this table can be added via "Specification Required" as defined in [RFC8126]. The expert SHOULD ensure the specification

defines the values for L and T as required in Section 4.4 of RFCAAAA. Allocated values MUST be in the range of 1 to 254.

7.3. TLS Extensions

IANA is requested to add supported_ekt_ciphers as a new extension name to the "ExtensionType Values" table of the "Transport Layer Security (TLS) Extensions" registry with a reference to this specification and allocate a value of TBD to for this.

[[Note to RFC Editor: TBD will be allocated by IANA.]]

Considerations for this type of extension are described in Section 5 of [RFC4366] and requires "IETF Consensus".

7.4. TLS Handshake Type

IANA is requested to add ekt_key as a new entry in the "TLS HandshakeType Registry" table of the "Transport Layer Security (TLS) Parameters" registry with a reference to this specification, a DTLS-OK value of "Y", and allocate a value of TBD to for this content type.

[[Note to RFC Editor: TBD will be allocated by IANA.]]

This registry was defined in Section 12 of [RFC5246] and requires "Standards Action".

8. Acknowledgements

Thank you to Russ Housley provided detailed review and significant help with crafting text for this document. Thanks to David Benham, Yi Cheng, Lakshminath Dondeti, Kai Fischer, Nermeen Ismail, Paul Jones, Eddy Lem, Jonathan Lennox, Michael Peck, Rob Raymond, Sean Turner, Magnus Westerlund, and Felix Wyss for fruitful discussions, comments, and contributions to this document.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, DOI 10.17487/RFC3264, June 2002, <https://www.rfc-editor.org/info/rfc3264>.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, DOI 10.17487/RFC3711, March 2004, <https://www.rfc-editor.org/info/rfc3711>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5649] Housley, R. and M. Dworkin, "Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm", RFC 5649, DOI 10.17487/RFC5649, September 2009, <https://www.rfc-editor.org/info/rfc5649>.

[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, DOI 10.17487/RFC5764, May 2010, <https://www.rfc-editor.org/info/rfc5764>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

9.2. Informative References

[I-D.ietf-perc-double] Jennings, C., Jones, P., Barnes, R., and A. Roach, "SRTP Double Encryption Procedures", draft-ietf-perc-double-07 (work in progress), September 2017.

[I-D.ietf-perc-private-media-framework]
Jones, P., Benham, D., and C. Groves, "A Solution Framework for Private Media in Privacy Enhanced RTP Conferencing", draft-ietf-perc-private-media-framework-05 (work in progress), October 2017.

[I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-22 (work in progress), November 2017.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC4366] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4366, DOI 10.17487/RFC4366, April 2006, <https://www.rfc-editor.org/info/rfc4366>.

Authors' Addresses

Cullen Jennings
Cisco Systems

Email: fluffy@iii.ca

John Mattsson
Ericsson AB

Email: john.mattsson@ericsson.com

David A. McGrew
Cisco Systems

Email: mcgrew@cisco.com

Dan Wing
Cisco Systems

Email: dwing@cisco.com

Flemming Andreason
Cisco Systems

Email: fandreas@cisco.com