PERC Working Group C. Jennings
Internet-Draft Cisco
Intended status: Standards Track J. Mattsson, Ed.
Expires: May 4, 2017 Ericsson
D. McGrew
D. Wing
F. Andreasen
Cisco
October 31, 2016

Encrypted Key Transport for Secure RTP
draft-ietf-perc-srtp-ekt-diet-02

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 within SRTP. This facility enables SRTP for decentralized conferences by distributing a common key to all of the conference endpoints.

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, roll over 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 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.

1.1. 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

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. 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 the format defined below for the FullEKTField and ShortEKTField.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                                                               :
:                        EKT Ciphertext                         :
:                                                               :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Security Parameter Index    | Length                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0 0 0 0 0 0 1 0|
+-+-+-+-+-+-+-+-+
```

Figure 1: Full EKT Field format

```
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|0 0 0 0 0 0 0 0|
+-+-+-+-+-+-+-+-+
```

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 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 2.3. This field is included in SRTP packets when EKT is in use.

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

SRTPMasterKeyLength: The length of the SRTPMasterKey. This depends on the cipher suite negotiated for SRTP using [RFC3264] SDP Offer/Answer 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 must 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 and the whole EKTField SHOULD be discarded if it contains message type value that is less than 64 and is not implemented.

2.2. 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.

2.2.1. Outbound Processing

See Section 2.6 which describes when to send an EKT packet with a FullEKTField. If a FullEKTField is not being sent, then a ShortEKTField needs to be sent so the receiver can correctly determine how to process the packet.

When an SRTP packet is to be 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 2.1. 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 2.3.

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

   > Note: the value of the EKT Ciphertext 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.

2.2.2. Inbound Processing

When receiving a packet on a RTP stream where EKT was negotiated, 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.

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 2.3, 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 2.1, to recover the SRTP Master Key, SSRC, and ROC fields. The 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. 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. 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.

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

2.2.2.1. Implementation Notes for Inbound Processing

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.

2.3. Ciphers

EKT uses an authenticated cipher to encrypt and authenticate the EKTPlaintext. 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).

An EKTCipher 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 may be larger than M. The decryption function D(K, C) takes the following inputs:

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

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

The decryption function returns a plaintext value P that is at least 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 concatenated with optional padding) 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 more that T times.

Security requirements for EKT ciphers are discussed in Section 4.

2.3.1. 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 [RFC3711], AESKW128 MUST be implemented for EKT and AESKW256 MAY be implemented.

2.3.2. Defining New EKT Ciphers

Other specifications may extend this document by defining other EKTCiphers as described in Section 5. 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.

2.4. 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.

2.5. 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.

2.6. 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.

3. 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 one DTLS-SRTP peer to another. 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.

3.1. 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.

3.2. SRTP EKT Key Transport Extensions to DTLS-SRTP

This document defines a new TLS negotiated extension called "srtp_ekt_key_transport"and a new TLS content type called EKTMessage.

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

```
enum {
  reserved(0),
  aeskw_128(1),
  aeskw_256(3),
} EKTCipherType;

struct {
  EKTCipherType ekt_ciphers<0..254>;
} SupportedEKTCiphers;

struct {
  EKTCipherType ekt_cipher;
  uint ekt_key_value<1..256>;
  uint srtp_master_salt<1..256>;
  uint16 ekt_spi;
  uint24 ekt_ttl;
} EKTkey;

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

struct {
  EKTMessageType ekt_message_type;
  select (EKTMessage.ekt_message_type) {
  case ekt_key:
    EKTKey;
  } message;
} EKTMessage;
```

Figure 4: Additional TLS Data Structures

If a DTLS client includes "srtp_ekt_key_transport" in its ClientHello, then a DTLS server that supports this extensions will includes "srtp_ekt_key_transport" in its ServerHello message. If a DTLS client includes "srtp_ekt_key_transport" in its ClientHello, but does not receive "srtp_ekt_key_transport" in the ServerHello, the DTLS client MUST NOT send DTLS EKTMessage messages.

When a DTLS client sends the "srtp_ekt_key_transport" in its ClientHello message, it MUST include the SupportedEKTCiphers as the extension_data for the extension, listing the EKTCipherTypes the client is willing to use in preference order, with the most preferred version first. When the server responds in the "srtp_ekt_key_transport" in its ServerHello message, it must include a SupportedEKTCiphers list that selects a single EKTCipherType to use (selected from the list provided by the client) or it returns an empty list to indicate there is no matching EKTCipherType in the clients list that the server is also willing to use. The value to be used in the EKTCipherType for future extensions that define new ciphers is the value from the "EKT Ciphers Type" IANA registry defined in Section 5.2.

The figure above defines the contents for a new TLS content type called EKTMessage which is registered in Section 5.4. The EKTMessage above is used as the opaque fragment in the TLSPlaintext structure defined in Section 6.2.1 of [RFC5246] and the "srtp_ekt_message" as the content type. The "srtp_ekt_message" content type is defined and registered in Section 5.3.

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

When the Server wishes to provide a new EKT Key, it can send EKTMessage containing an EKTKey with the new key information. The client MUST respond with an EKTMessage of type ekt_key_ack, if the EKTKey was successfully processed and stored or respond with the the ekt_key_error EKTMessage otherwise.

The diagram below shows a message flow of DTLS client and DTLS server using the DTLS-SRTP Key Transport extension.

```
   Client                                               Server

   ClientHello + use_srtp + srtp_ekt_key_trans
                                 -------->
                                  ServerHello+use_srtp+srtp_ekt_key_trans
                                                Certificate*
                                          ServerKeyExchange*
                                         CertificateRequest*
                                 <--------     ServerHelloDone
   Certificate*
   ClientKeyExchange
   CertificateVerify*
   [ChangeCipherSpec]
   Finished                      -------->
                                          [ChangeCipherSpec]
                                 <--------            Finished
   ekt_key                       <--------
   ekt_key_ack                   -------->
   SRTP packets                  <------->      SRTP packets
   SRTP packets                  <------->      SRTP packets
   ekt_key (rekey)               <-------
   ekt_key_ack                   -------->
   SRTP packets                  <------->      SRTP packets
   SRTP packets                  <------->      SRTP packets
```

Figure 5: DTLS/SRTP Message Flow

3.3. 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.

3.4. Sending the DTLS EKT_Key Reliably

The DTLS ekt_key is sent using the retransmissions specified in Section 4.2.4. of DTLS [RFC6347].

4. 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 full EKT packet, 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.

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 2.2.2.1 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 send more than T Full EKT Field using the same EKTKey. In addition, the EKTKey MUST NOT be used beyond the lifetime provided by the TTL described in Section 3.2.

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.

5. IANA Considerations

5.1. EKT Message Types

IANA is requested to create a new registry for "EKT Messages Types". 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 [RFC5226]. 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.

5.2. EKT Ciphers

IANA is requested to create a new registry for "EKT Ciphers". The initial values in this registry are:

Name	Value	Specification
AESKW128	1	RFCAAAA
AESKW256	3	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 [RFC5226]. The expert SHOULD ensure the specification defines the values for L and T as required in Section 2.3 of RFCAAA. Allocated values MUST be in the range of 1 to 254.

5.3. TLS Extensions

IANA is requested to add "srtp_ekt_key_transport" as an 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".

5.4. TLS Content Type

IANA is requested to add "srtp_ekt_message" as an new descriptions name to the "TLS ContentType Registry" table of the "Transport Layer Security (TLS) Extensions" 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".

6. Acknowledgements

Thank you to Russ Housley provided detailed review and significant help with crafting text for this document. Thanks to David Benham, Eddy Lem, Felix Wyss, Jonathan Lennox, Kai Fischer, Lakshminath

Dondeti, Magnus Westerlund, Michael Peck, Nermeen Ismail, Paul Jones, Rob Raymond, and Yi Cheng for fruitful discussions, comments, and contributions to this document.

This draft is a cut down version of draft-ietf-avtcore-srtp-ekt-03 and much of the text here came from that draft.

7. References

7.1. 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, <http://www.rfc-editor.org/info/rfc2119>.

[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, <http://www.rfc-editor.org/info/rfc3711>.

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

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://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, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc6347>.

7.2. Informative References

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

[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, <http://www.rfc-editor.org/info/rfc4366>.

Authors' Addresses

Cullen Jennings
Cisco Systems
Calgary, AB
Canada

Email: fluffy@iii.ca

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
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
510 McCarthy Blvd.
Milpitas, CA 95035
US

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

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

Email: fandreas@cisco.com