TSVWG
Internet-Draft
Updates: RFC5061 (if approved)
Intended status: Standards Track
Expires: 3 September 2026

M. Westerlund
J. Preu Mattsson
C. Porfiri
Ericsson
M. Txen
Mnster Univ. of Appl. Sciences

2 March 2026

Stream Control Transmission Protocol (SCTP) DTLS Chunk
draft-ietf-tsvwg-sctp-dtls-chunk-02

Abstract

This document describes a method for adding Datagram Transport Layer Security (DTLS) based authentication and cryptographic protection to the Stream Control Transmission Protocol (SCTP).

This SCTP extension is intended to enable communication privacy for applications that use SCTP as their transport protocol and allows applications to communicate in a way that is designed to prevent eavesdropping and detect tampering or message forgery. Once enabled, this also applies to the SCTP payload as well as the SCTP control information.

Applications using this SCTP extension can use most of the transport features provided by SCTP and its other extensions. The use of the SCTP Authentication extension defined in RFC 4895 is incompatible with the extension defined in this document but would not provide any additional service. This implies that the Dynamic Address Reconfiguration as specified in RFC 5061 can only be used as described in this document.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-tsvwg-sctp-dtls-chunk/.

Discussion of this document takes place on the Transport Area Working Group (tsvwg) Working Group mailing list (mailto:tsvwg@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/tsvwg/. Subscribe at https://www.ietf.org/mailman/listinfo/tsvwg/.

Source for this draft and an issue tracker can be found at https://github.com/gloinul/draft-westerlund-tsvwg-sctp-DTLS-chunk.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction and Protocol Overview

This document extends the Stream Control Transmission Protocol (SCTP), as specified in [RFC9260], by defining the DTLS chunk format and the procedures for negotiating and managing its usage. DTLS chunk support is integrated into the SCTP implementation, enabling the secure transfer of SCTP packets (including both control and DATA chunks).

The DTLS chunk protects a sequence of SCTP chunks by encapsulating their DTLS-based ciphertext. This processing is based on DTLS 1.3, as specified in [RFC9147].

Key mangement is performed outside of the SCTP implementation and is out of scope of this document. This process is referred to as the DTLS Key Managenement Method. While these methods can be also be based on DTLS 1.3 it is not a requirement.

The DTLS chunk in combination with the DTLS key management method provide mutual authentication, confidentiality, DTLS based data origin authentication, integrity, and replay protection for SCTP packets. The DTLS Key Management Method utilizes an API to provision the SCTP association's DTLS chunk protection with key-material to enable and rekey the protection operations.

Figure 1 is an example illustrating the DTLS chunk processing in regard to SCTP and the Upper Layer Protocol (ULP) using DTLS 1.3 as the DTLS Key Management Method. Here the DTLS Key Management Method provides validation, i.e. using certificates, handshaking, updating policies etc.

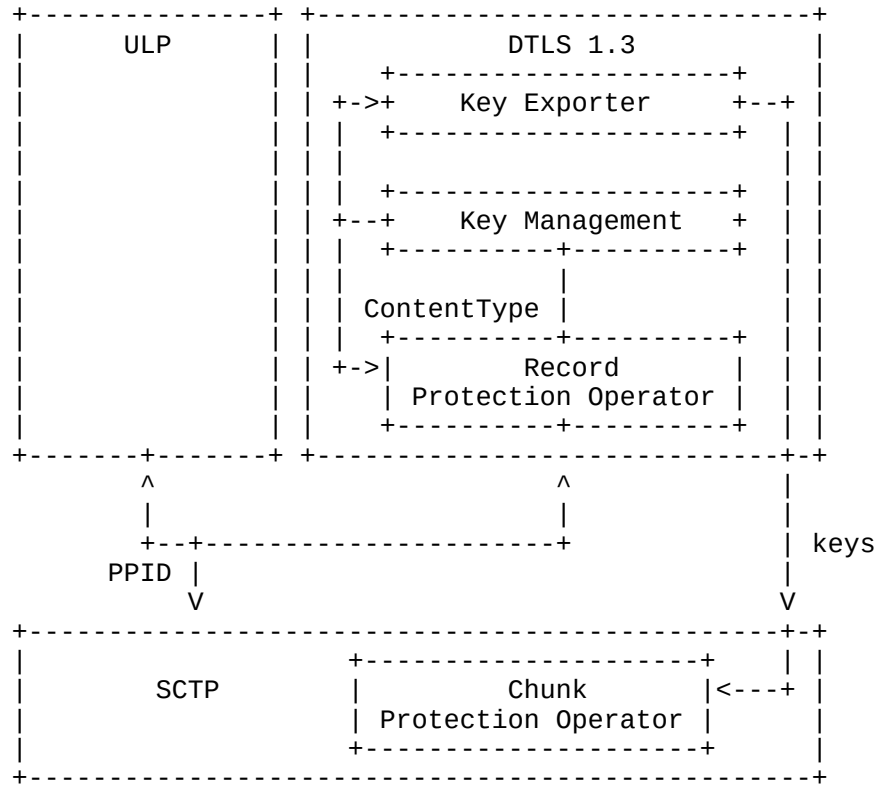

Figure 1: DTLS Chunk Layering in Regard to SCTP and ULP

After receiving an SCTP packet and identifying the association using the SCTP common header, the DTLS chunk is processed by the Chunk Protection Operator. The Chunk Protection Operator performs replay protection, decryption, and authentication. If this processing is successul, the encapsulated SCTP chunks are further processed.

For outgoing traffic, after the SCTP stack has created the unprotected SCTP packet containing control and/or DATA chunks, these SCTP chunks will be processed by the Chunk Protection Operator for protection. This results in a DTLS 1.3 record encapsulated in a DTLS chunk.

The use of the DTLS 1.3 handshake for initial mutual authentication, key establishment, and periodic re-authentication and rekeying with Diffie-Hellman of the DTLS chunk protection is defined in separate documents, (see Section 6). To prevent downgrade attacks affecting the DTLS Key Management negotiation the DTLS Key Management Method should implement specific procedures when deriving keys.

The Chunk Protection Operator performs protection operations on all chunks of an SCTP packet. No information is sent in plain text except for the following:

* The initial SCTP handshake.

* The initial DTLS Key Management traffic.

* the SCTP common header, the SCTP DTLS chunk header.

* The INIT and INIT ACK chunks during an SCTP Restart procedure.

Support of the DTLS chunk and the selection of a DTLS Key Management Method are negotiated during the SCTP handshake using a new parameter. DTLS Key Management and application traffic are then multiplexed using the Payload Protocol Identifier (PPID). This document defines the dedicated PPID 4242 for use by all DTLS Key Management Methods.

Applications using the DTLS chunk can leverage most transport features provided by SCTP and its extensions. However, the following limitations apply:

* The handling of INIT collisions is not supported.

* Performing an SCTP restart without knowing the restart key material is not supported.

* The use of the lookup address in the Dynamic Address Reconfiguration extension as specified in [RFC5061] is not supported.

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Protocol Considerations

3.1. DTLS Considerations

Once the DTLS Key Management Method has established its context, it derives primary and restart keys and configures the Chunk Protection Operator via an API. This establishes the necessary DTLS key contexts for SCTP chunk encryption and decryption. A DTLS key context for normal operations use MUST be created, while a DTLS key context for SCTP association restart SHOULD be created.

In this document we use the terms DTLS Key context for indicating a Key and IV, produced by the DTLS Key Management, and all relevant data that needs to be provided to the Chunk Protection Operator for DTLS encryption and decryption. DTLS Key context includes Keys and IV for sending and receiving, replay window, last used sequence number. Each DTLS key context is associated with a three-value tuple identifying the context, consisting of SCTP Association, the restart indicator, and the DTLS epoch.

The DTLS Connection ID in the DTLS Record layer used in the DTLS Chunk MUST NOT be used.

The first DTLS key context established for any SCTP association MUST use epoch 3.

The replay window for the DTLS Sequence Number need to account for the concurrent transmission of packets on multiple paths in multihomed associations. In particular, this applies to packets containing HEARTBEAT chunks. The window size must be sufficiently large to accommodate these latency differences.

Endpoints implementing DTLS Chunk MUST support DTLS records containing up to 2^14 (16384) bytes of plain text.

3.2. SCTP Considerations

The SCTP authentication extension (SCTP-AUTH) defined in [RFC4895] is incompatible with the extension defined in this document. Therefore, its support MUST NOT be negotiated in combination with the support of the DTLS chunk.

In particular, the dynamic address reconfiguration defined in [RFC5061] cannot use SCTP-AUTH. Instead, the DTLS chunk is used for authentication. This introduces the following limitations:

* The lookup address MUST NOT be used.

* The dynamic address reconfiguration extension MUST NOT be used unless DTLS chunk handling is enabled in both directions.

To mitigate potential information leakage from packet size variations, implementations MAY pad SCTP packets to uniform sizes. However, the padding MUST be applied within the encryption envelope to ensure the padding itself is protected.

Both SCTP and DTLS provide mechanisms for padding packets. If padding of SCTP packets are desired to hide actual message sizes it is RECOMMENDED to use the SCTP Padding Chunk [RFC4820] to generate a consistent SCTP payload size. Support of this chunk is only required on the sender side, any SCTP receiver will safely ignore the PAD Chunk. However, if the PAD chunk is not supported DTLS padding MAY be used.

It should be noted that regardless of whether SCTP padding or DTLS padding is used, the additional bytes are not account for by the SCTP congestion control. Extensive use of padding has potential to worsen congestion situations, as the SCTP association will consume more bandwidth than its derived share by the congestion control.

Using the SCTP PAD chunk is preferred because it is visible in the SCTP layer. Therefore, future extension or SCTP implementation that account for the padding in the congestion control. In contrast, DTLS padding hides this packet expansion from the SCTP layer.

4. New Chunk, Parameter and Error Causes

4.1. DTLS Key Management Parameter

The DTLS Key Management Parameter is used to negotiate the support of the DTLS chunk and the key management method used for the DTLS chunks. The format of this chunk parameter is depicted in Figure 2.

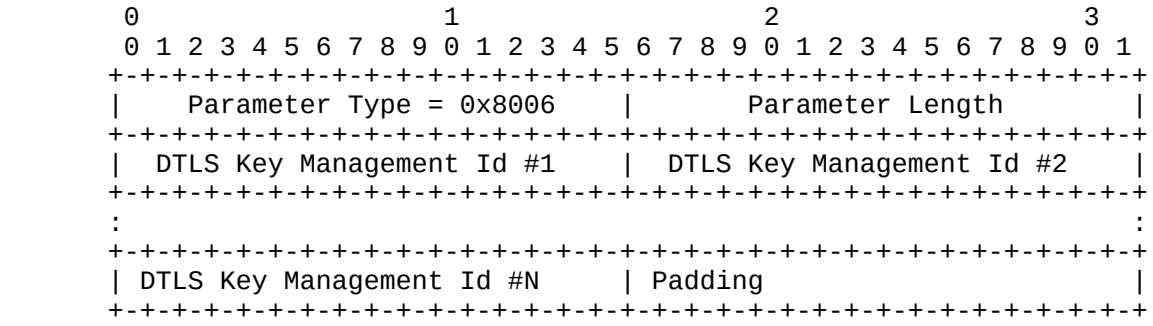

Figure 2: DTLS Key Management Parameter

Parameter Type: 16 bits (unsigned integer)

This value MUST be set to 0x8006. Note that this parameter type requires the receiver to ignore the parameter and continue processing if the parameter type is not supported. This is accomplished (as described Section 3.2.1 of [RFC9260]) by the use of the upper bits of the parameter type.

Parameter Length: 16 bits (unsigned integer)
This value holds the length of the parameter, which will be 2 times the number of DTLS Key Management identifiers (N) plus 4.

DTLS Key Management Identifier: 16 bits (unsigned integer)
Each DTLS Key Management Identifier (Section 9.1) is a 16-bit unsigned integer value indicating a DTLS Key Management Method. In the INIT chunk the DTLS Key Management Methods are listed in descending order of preference, i.e. the first listed in the parameter is the most preferred one and the last listed one is the least preferred by the sender in the INIT chunk. In the INIT ACK chunk the endpoint chooses one of the DTLS Key Management Methods supported by the peer.

Padding: 0 or 16 bits (unsigned integer)
If the number of included DTLS Key Management Methods is odd the parameter MUST be padded with two bytes. The padding MUST be set to 0 by the sender and MUST be ignored by the receiver.

The DTLS Key Management Parameter MAY be included in the INIT and INIT ACK chunk and MUST NOT be included in any other chunk. When included in an INIT chunk, the DTLS Key Management Parameter MUST include at least one DTLS Key Management Identifier. When included in an INIT ACK chunk, it MUST include exactly one DTLS Key Management Identifier.

4.2. DTLS Chunk (DTLS)

The DTLS chunk is used to hold the DTLS 1.3 record with the protected payload of a plain text SCTP packet without the SCTP common header.

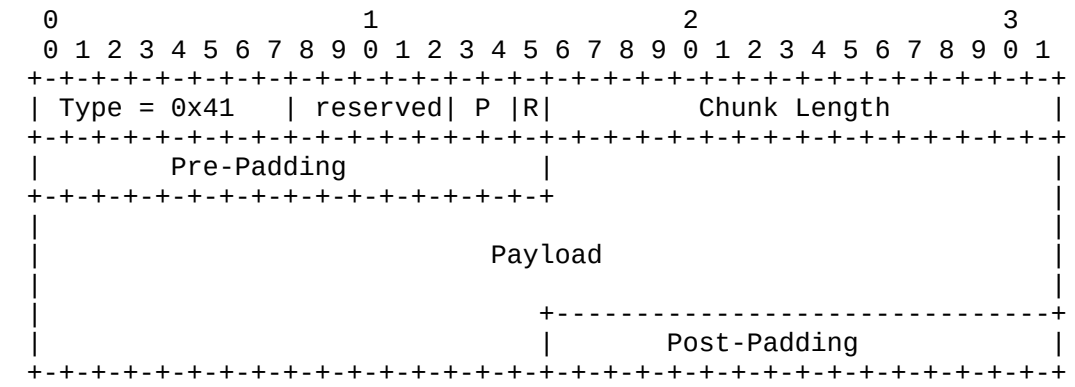

Figure 3: DTLS Chunk

Type: 8 bits (unsigned integer)
This value MUST be set to 0x41. It should be noted that the chunk type requires the receiver stop processing this SCTP packet, discard the unrecognized chunk and all further chunks, and report the unrecognized chunk in an ERROR chunk using the 'Unrecognized Chunk Type' error cause, if the receiver does not support this chunk type. This is accomplished (as described in Section 3.2 of [RFC9260]) by the use of the upper bits of the chunk type.

reserved: 5 bits
Reserved bits for future use. These bits MUST be set to 0 by the sender and MUST be ignored by the receiver.

R: 1 bit (boolean)
Restart indicator. If this bit is set this DTLS chunk is protected with a Restart DTLS Key context.

P: 2 bits (unsigned integer)
Payload Pre-Padding length. It indicates how many bytes are inserted for padding before the DTLSCiphertext. See the text below for computing P.

Chunk Length: 16 bits (unsigned integer)
This value holds the length of the Payload in bytes plus 4 plus the Payload Pre-Padding length.

Pre-Padding: 0, 8, 16, or 24 bits
The length of the padding is given by the Payload Pre-Padding length P. The sender MUST pad with zero bytes and the receiver MUST ignore the padding bytes.

Payload: variable length

This MUST contain exactly one DTLSCiphertext as specified in DTLS 1.3 [RFC9147].

Post-Padding: 0, 8, 16, or 24 bits
If the length of the Payload is not a multiple of 4 bytes, the sender MUST pad the chunk with all zero bytes to make the chunk 32-bit aligned. The Padding MUST NOT be longer than 3 bytes and it MUST be ignored by the receiver.

From Section 4 of [RFC9147], the DTLS record header has variable length and is depicted in Figure 4.

```
struct {
    opaque unified_hdr[variable];
    opaque encrypted_record[length];
} DTLSCiphertext;
```

Figure 4: DTLS DTLSCiphertext

The DTLSCiphertext contains the unified_hdr followed by the encrypted_record, where unified_hdr has variable format. The encrypted_record MUST be 32-bit aligned in relation to the start of the DTLS Chunk. The Pre-Padding MUST be used to achieve this. The format of unified_hdr is depicted in Figure 5.

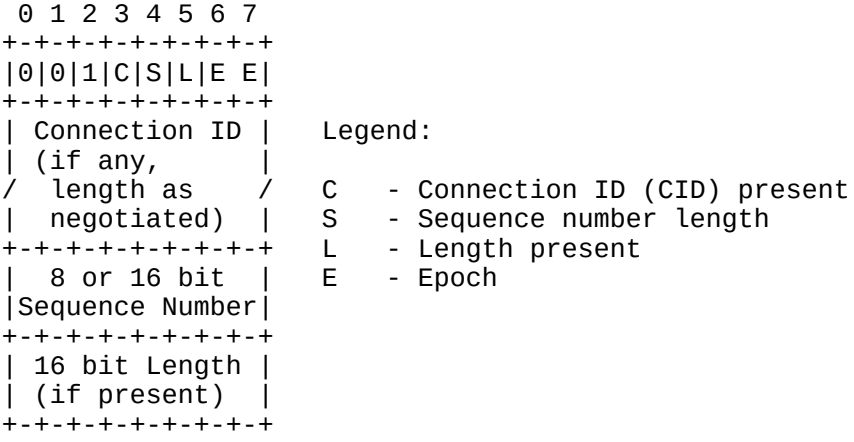

Figure 5: DTLS unified_hdr

Examples of preferred DTLSCiphertext are shown in Figure 6.

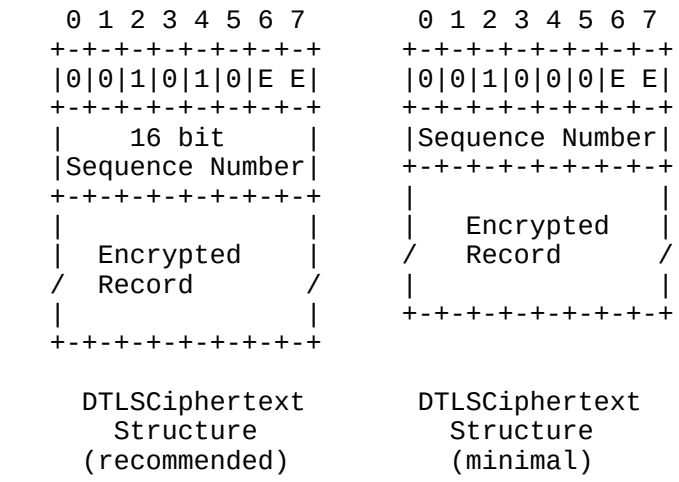

Figure 6: DTLSCiphertext recommended structure

Thus the size of the DTLSCiphertext header is computed from the first_byte as follows:

```
#define S_BIT 0x08
#define L_BIT 0x04

size = 1;
/* Add in the size of the sequence number. */
if ((first_byte & S_BIT) != 0)
    size += 2;
else
    size += 1;
/* Add in the size of the length field, if present. */
if ((first_byte & L_BIT) != 0)
    size += 2;
```

Then the Payload Pre-Padding length P can be computed by

```
P = (4 - (size & 0x3)) & 0x03;
```

The use of the Pre-Padding allows a receiver to perform an in-place decryption and then process the sequence of chunks. SCTP as specified in [RFC9260] guarantees that chunks start on a 32-bit boundary.

4.3. New Error Causes

This specification defines two new error causes.

4.3.1. Missing DTLS Chunk Support

The DTLS Chunk Support Required error cause can be sent by the receiver of the packet containing the INIT chunk to indicate that the receiver requires the support of the DTLS chunk, but no DTLS Key Management parameter was included in the INIT chunk.

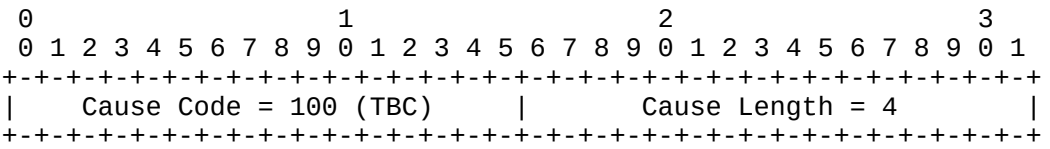

Figure 7: Error Missing DTLS Chunk Support

Cause Code: 16 bits (unsigned integer)
This value MUST be set to 100 (TBC).

Cause Length: 16 bits (unsigned integer)
This value MUST be set to 4.

This error cause MAY be included in an ABORT chunk. It MUST NOT be included in any other chunk.

4.3.2. No Common DTLS Key Management Method

The No Common DTLS Key Management Method error cause can be used by the receiver of the packet containing the INIT chunk to indicate that receiver does not support any of DTLS key management methods offered by the sender of packet containing the INIT chunk.

The format of this error cause is depicted in Figure 8.

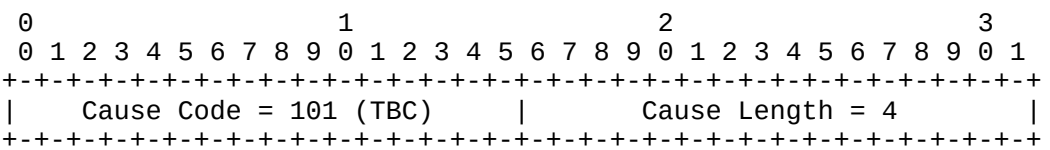

Figure 8: Error Cause No Common DTLS Key Management Method

Cause Code: 16 bits (unsigned integer)
This value MUST be set to 101 (TBC).

Cause Length: 16 bits (unsigned integer)
This value MUST be set to 4.

This error cause MAY be included in an ABORT chunk. It MUST NOT be included in any other chunk.

5. Procedures

5.1. Establishment of a Protected Association

To initiate an SCTP association, an SCTP endpoint wanting to use the DTLS chunk MUST send an SCTP packet with an INIT chunk containing the DTLS Key Management Parameter (see Section 4.1). This parameter lists the supported DTLS Key Management Identifiers (see Figure 2) in descending order or preference.

If an SCTP endpoint receives an SCTP packet containing an INIT chunk with a DTLS Key Management Parameter, it MUST reply with a packet containing an INIT ACK containing its own DTLS Key Management Parameter. This parameter MUST contains exactly one DTLS Key Management Method Identifier out of the list of received ones. If there is no DTLS Key Management Method supported by both endpoints, and the endpoints' policy requires requires the use of the DTLS chunk, the receiver MUST reply with an SCTP packet containing an ABORT chunk and MAY include the error cause "No Common DTLS Key Management Method" (see Section 4.3.2). If the endpoints' policy does not require the use of the DTLS chunk, the receiver MAY reply with an SCTP packet containing an INIT ACK chunk without any DTLS Key Management Parameter to indicate that it is willing to setup an association without DTLS chunk support. Additionally, when an SCTP endpoint requiring DTLS chunk support receives an SCTP packet containing an INIT chunk without a DTLS Key Management Parameter, it MUST reply with an packet containing an ABORT chunk an MAY include the error cause indicating that the DTLS chunk support is missing (see Section 4.3.1).

If an SCTP endpoint, which has sent an SCTP packet with an INIT chunk containing DTLS Key Management Parameter receives a packet containing an INIT ACK chunk with a DTLS Key Management Parameter not containing exactly one of the DTLS Key Mangement Identifiers it sent, the endpoint MUST reply with an packet containing an ABORT chunk. The ABORT chunk MAY include the Protocol Violation error cause. If the INIT ACK does not contain any DTLS Key Management Parameter and the endpoint's policy requires the use of the DTLS chunk, the endpoint MUST send an SCTP packet with an ABORT chunk. It MAY include the error cause indicating that DTLS chunk support is missing (see Section 4.3.1). If the endpoint's policy does not require the use of the DTLS chunk, it MAY continue with the handshake.

When the SCTP association has been established the process defined by the selected DTLS Key Management Method MUST be followed for establishing DTLS Key Contexts and installing them.

5.2. DTLS Chunk Handling

The DTLS chunk MUST NOT be bundled with any other chunk. In particular, it MUST be the first and only chunk.

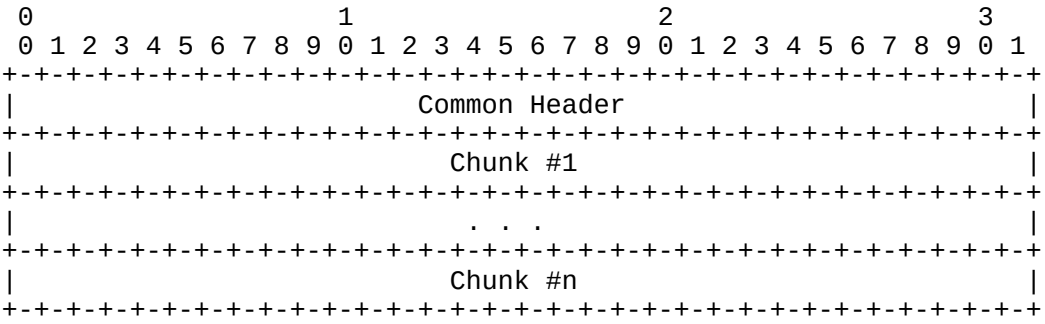

Figure 9: Unprotected SCTP Packet

The diagram shown in Figure 9 describes the structure of an unprotected SCTP packet as described in [RFC9260].

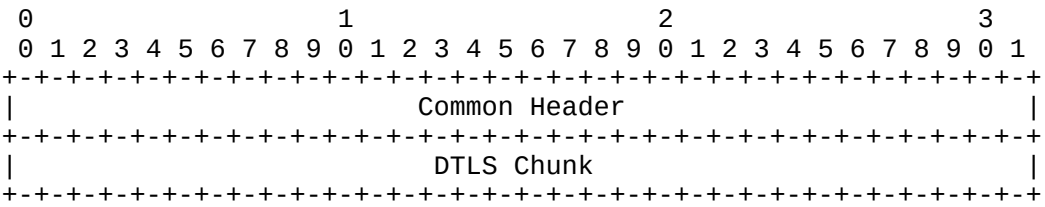

Figure 10: Protected SCTP Packets

The diagram shown in Figure 10 describes the structure of a protected SCTP packet being sent. Such packets contain only the SCTP common header and one DTLS chunk.

Once the credentials for sending DTLS chunks have been configured by the application, all SCTP packets are sent with a DTLS chunk.

When an SCTP packet needs to be sent, the sequence of chunks is used as DTLSInnerPlaintext.content and DTLSInnerPlaintext.type is set to application_data. Then the DTLSCiphertext is computed and used as the payload of the DTLS chunk. Finally the SCTP common header is prepended.

When the DTLS chunk is used, the endpoint MUST consider the DTLS chunk header and the overhead of DTLS to ensure that the final SCTP packet does not exceed the PMTU.

When an SCTP packet containing a DTLS chunk bundled with any other chunk is received, the packet MUST be silently discarded.

After the application has restricted the SCTP packet handling to protected SCTP packets only, a SCTP packet not containing a DTLS chunk MUST be silently discarded.

When processing the payload of the DTLS chunk (i.e. the DTLSCiphertext), the Restart flag in addition to the unified_hdr is used to find the keys for processing the encrypted_record.

After the encrypted_record has been verified and decrypted, the corresponding chunks (the DTLSInnerPlaintext.content) are processed as defined in the corresponding specifications.

When the Chunk Protection Operator will experience a non-critical error, it MUST NOT abort the association.

5.3. Termination of a Protected Association

Note that the closure of any DTLS Key Management Method doesn’t compromise the capability of terminating the SCTP association gracefully as that capability only relies on the Key Context and not on the DTLS Key Management Method from where it has been derived.

5.4. SCTP Restart Considerations

This section deals with the handling of an unexpected INIT chunk during an Association lifetime as described in Section 5.2 of [RFC9260] with the purpose of defining a protected restart procedure.

When the upper layer protocols require support of SCTP Restart for associations using the DTLS chunk, as in case of 3GPP NG-C protocol [ETSI-TS-38.413], the endpoint needs to support also the protected SCTP Restart procedure described below. Implementing the protected restart procedure is RECOMMENDED, however not required as persistent secure storage of the restart DTLS Key Context is needed.

The protected SCTP restart procedure keeps the security characteristics of an SCTP Association using DTLS chunks.

In protected SCTP Restart, INIT and INIT ACK chunks are sent strictly according to [RFC9260], but COOKIE ECHO and COOKIE ACK chunks are encrypted using DTLS chunks and the restart DTLS Key contexts.

In order to support protected SCTP Restart, the SCTP Endpoints need to allocate and maintain dedicated restart DTLS Key contexts, SCTP packets protected by these contexts will be identified in the DTLS chunk with the R (Restart) bit set (see Section 4.2). Both SCTP endpoints need to ensure that Restart DTLS key contexts are preserved for supporting the protected SCTP Restart use case.

In order for the protected SCTP endpoint to be available for protected SCTP Restart purposes, the DTLS chunk needs access to a DTLS Key context for this SCTP association that needs to be kept in a well-known state so that both SCTP endpoints are aware of the DTLS sequence numbers and replay window, i.e. initialized but never used. An SCTP endpoint MUST NOT use the SCTP Restart DTLS Key for any other use case than SCTP association restart.

An SCTP endpoint wanting to be able to initiate a protected SCTP restart needs to store securely and persistently the restart Keys, and related DTLS epoch, indexed so that when performing a restart with the peer node it had a protected SCTP association which can identify the right restart Key and DTLS epoch and initialize the restart DTLS Key Context for when restarting the SCTP association. The keys and epoch need to be stored securely and persistently so that they survive the events that are causing protected SCTP Restart procedure to be used, for instance a crash of the SCTP stack. The security considerations for persistent secure storage of keying materials is further discussed in Section 10.4.

The SCTP Restart handshakes INIT, INIT ACK, COOKIE ECHO, COOKIE ACK exactly as in legacy SCTP Restart case; INIT, INIT ACK MUST be sent plain as in the legacy, whereas COOKIE ECHO, COOKIE ACK Chunks MUST be sent as DTLS chunk protected using the restart DTLS key context.

A DTLS Chunk using the restart DTLS key context is identified by having the R bit (Restart Indicator) set in the DTLS Chunk (see Figure 3). There’s exactly one active Restart DTLS Context at a time, the newest. However, a crash at the time having completed the DTLS Key Management exchange but failing to commit the DTLS Key Context to persistent secure storage could result in loss of the latest DTLS Key Context. Therefore, the endpoints SHOULD retain the old restart DTLS key context until the DTLS Key Management confirms the new ones are committed to secure storage. This can for example ensure that at key-changes signals to terminate the old DTLS Key Contexts (including the restart) is never sent until the new restart DTLS Key Context has been committed to storage.

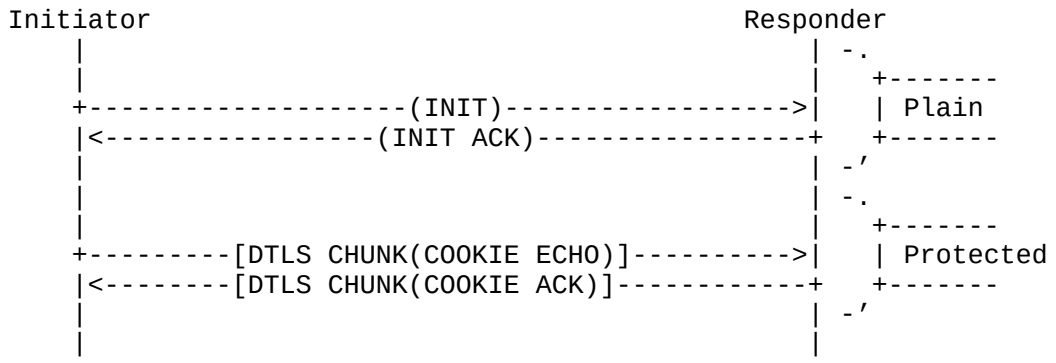

Figure 11: Handshake of SCTP Restart for DTLS in SCTP

The Figure 11 shows how the control chunks being used for SCTP Association Restart are transported within DTLS in SCTP.

A restarted SCTP Association MUST continue to use the Restart DTLS Key Context, for user traffic until a new primary DTLS Key Context will be available. The implementors SHOULD initiate a rekeying as soon as possible, and derive the primary and restart keys so that the time when no Restart DTLS Key Context is available is kept to a minimum. Note that another restart attempt prior to having created new restart DTLS Key context for the new SCTP association will result in the endpoints being unable to restart the SCTP association.

After restart the next primary DTLS key context MUST use epoch 3, i.e. the epoch value is reset. After having derived new primary DTLS Key Context the endpoint installs the primary DTLS Key Context first, and start using it. The new restart DTLS Key Context is only installed after any old in-flight restart packets will have been received.

An SCTP Endpoint supporting only normal SCTP Restart and involved in an SCTP Association using DTLS Chunks SHOULD NOT attempt to restart the Association. The effect will be that the restart initiator will receive INIT ACK but then all sent packets with COOKIE ECHO will be dropped until the peer nodes times out the SCTP Association from lack of any response from the restarting node.

An SCTP Endpoint supporting only legacy SCTP Restart and involved in an SCTP Association using DTLS Chunks, when receiving a COOKIE ECHO chunk protected by DTLS chunk as described above, thus having the R bit (Restart Indicator) set in the DTLS Chunk (see Figure 3), MUST silently discard it.

6. DTLS Key Management Method Considerations

This document specifies the mechanisms for protecting SCTP associations using DTLS chunks, including an API for managing the corresponding key material. While this document defines the interface, the upper layer is responsible for the actual key management.

DTLS Key Management Methods define the procedures for initial key generation, key updates and peer authentication. These procedures are out of scope for this document. Currently two DTLS Key Management Methods with different properties (such as mutual authentication and rekeying) are defined:

* [I-D.ietf-tsvwg-dtls-chunk-key-management]

* [I-D.westerlund-tsvwg-sctp-DTLS-handshake]

An SCTP endpoint MAY support multiple DTLS Key Management Methods subject to implementation requirements and local security policies.

Every DTLS Key Management Method

* MUST be registered in the IANA Registry Section 9.1 to receive a unique identifier, enabling negotiation during the SCTP handshake.

* SHOULD use the PPID from Section 9.5 to ensure that the DTLS Key Management Method related user messages are processed by the relevant entity.

* SHOULD ensure that the local receive keys are installed before the peer installs the corresponding send keys.

* MUST include the DTLS Key Management Method Identifiers sent and received during the SCTP handshake in the key derivation to mitigate downgrade attacks. Both sides MUST use the same byte ordering for the DTLS Key Management Method Identifiers.

7. Abstract API

This section describes an abstract API that is needed between a DTLS Key Management Method and the DTLS Chunk. This is an example API and there are alternative implementations.

This API enables the cryptographical protection operations by setting client/server write keys, sequence number keys, and IVs for primary and restart DTLS contexts. The write key is used by the cipher suite for DTLS record protection (Section 5.2 of [RFC8446]). The

initialization vector (IV) is random material used to XOR with the sequence number to create the nonce per Section 5.3 of [RFC8446]. The sequence number key is used to encrypt the sequence number (Section 4.2.3 of [RFC9147]).

7.1. Set Supported DTLS Key Management Methods

Prior to attempting to establish an SCTP assocation an SCTP endpoint needs to configure which DTLS Key Management Methods it supports if establishing a SCTP association. This will be included in INIT if the endpoint initiated the SCTP association. Else it will be used to determine the selected DTLS Key Management method that is returned in the INIT ACK.

Request: Set Supported DTLS Key Management Methods

Parameters:

* SCTP Association Handle: The handle to what may become an SCTP Association or a server port accepting association establishment.

* List of Identifiers: A prioritized list of DTLS Key Management identifiers that are supported, from the most preferred to the least preferred.

Reply: Success or Error

Parameters: None

7.2. Get Offered DTLS Key Management Methods

After an SCTP association has been established the key management function will need to get the list of DTLS key management IDs that was present in DTLS Key Management parameter in the INIT and INIT ACK chunks. This list will be used by the selected DTLS key management method to derive security keys and prevent downgrade attacks.

Request: Get DTLS Key Management Methods

Parameters:

* SCTP Association: Reference to the relevant SCTP association to request the exchanged Identifiers.

Reply: Offered and Selected DTLS Key Management Methods

* List of Identifiers: This API call returns a list of DTLS key-

management Identifiers. The list first contains all the Identifiers present in DTLS Key Management Parameter in the INIT Chunk, followed by the single identifier for the selected methods that was exchanged in the DTLS Key Management Parameter in the INIT ACK chunk.

7.3. Cipher Suite Capabilities

The DTLS Key Management Method needs to know which cipher suites defined for usage with DTLS 1.3 that are supported by the DTLS chunk and its protection operations block. All TLS cipher suites that are defined are listed in the TLS cipher suite registry [TLS-CIPHER-SUITES] at IANA and are identified by a 2-byte value. Thus this needs to return a list of all supported cipher suites to the higher layer.

Request : Get Cipher Suites

Parameters : none

Reply : Cipher Suites

Parameters : list of cipher suites

7.4. Establish Client Write Keying Material

The DTLS Chunk can use one out of multiple sets of cipher suite and corresponding key materials.

The following information needs to be provided when setting Client Write (transmit) Keying material:

Request : Establish Client Write Key and IV

Parameters :

* SCTP Association: Reference to the relevant SCTP association to set the keying material for.

* Restart indication: A bit indicating whether the Key is for restart purposes

* DTLS Epoch: The DTLS epoch these keys are valid for. Note that Epoch lower than 3 are not expected as they are used during DTLS handshake.

* Cipher Suite: 2 bytes cipher suite identification for the DTLS

1.3 Cipher suite used to identify the DTLS AEAD algorithm to perform the DTLS record protection. The cipher suite is fixed for a (SCTP Association, Key) pair.

* Write Key, Sequence Number Key and IV: The cipher suite specific binary object containing all necessary information for protection operations. The secret will be used by the DTLS 1.3 client to encrypt the record. Binary arbitrary long object depending on the cipher suite used.

Reply : Established or Failed

7.5. Establish Server Write Keying Material

The DTLS Chunk can use one out of multiple sets of cipher suite and corresponding key materials.

The following information needs to be provided when setting Server Write (transmit) Keying material:

Request : Establish Server Write Key and IV

Parameters :

* SCTP Association: Reference to the relevant SCTP association to set the keying material for.

* Restart indication: A bit indicating whether the Key is for restart purposes

* DTLS Epoch: The DTLS epoch these keys are valid for. Note that Epoch lower than 3 are not expected as they are used during DTLS handshake.

* Cipher Suite: 2 bytes cipher suite identification for the DTLS 1.3 Cipher suite used to identify the DTLS AEAD algorithm to perform the DTLS record protection. The cipher suite is fixed for a (SCTP Association, Key) pair.

* Write Key, Sequence Number Key and IV: The cipher suite specific binary object containing all necessary information for protection operations. The secret will be used by the DTLS 1.3 client to encrypt the record. Binary arbitrary long object depending on the cipher suite used.

Reply : Established or Failed

7.6. Destroy Client Write Keying Material

A function to destroy the Client Write (transmit) keying material for a given epoch for a given Key for a given SCTP Association.

Request : Destroy client write key and IV

Parameters :

* SCTP Association
* Restart indication
* DTLS Epoch

Reply: Destroyed

Parameters : true or false

7.7. Destroy Server Write Keying Material

A function to destroy the Server Write (transmit) keying material for a given epoch for a given Key for a given SCTP Association.

Request : Destroy server write key and IV

Parameters :

* SCTP Association
* Restart indication
* DTLS Epoch

Reply: Destroyed

Parameters : true or false

7.8. Set Key to Use

Set which key to use to protect the future SCTP packets sent by the SCTP Association.

Request : Set Key used

Parameters :

* SCTP Association

* Restart indication

* DTLS Epoch

Reply: Key set

Parameters : true or false

7.9. Require Protected SCTP Packets

A function to configure an SCTP association to require that normal SCTP packets being received must be protected in a DTLS Chunk going forward.

Parameters:

* SCTP Association

Reply: Acknowledgement

7.10. Get AEAD Encryption Invocations

Get the number of AEAD encryption invocations (protected messages) for a given epoch.

Request : Get AEAD Encryption Invocations

Parameters :

* SCTP Association

* Restart indication

* DTLS Epoch

Reply: AEAD Encryption Invocations

Parameters : non-negative integer

7.11. Get AEAD Decryption Invocations

Get the number of AEAD decryption invocations for a given epoch.

Request : Get AEAD Decryption Invocations

Parameters :

* SCTP Association

* Restart indication

* DTLS Epoch

Reply: AEAD Decryption Invocations

Parameters : non-negative integer

7.12. Get Failed AEAD Decryption Invocations

Get the number of failed AEAD decryption invocations for a given epoch.

Request : Get Failed AEAD Decryption Invocations

Parameters :

* SCTP Association

* Restart indication

* DTLS Epoch

Reply: Failed AEAD Decryption Invocations

Parameters : non-negative integer

7.13. Configure Replay Protection

The DTLS replay protection in this usage is expected to be fairly robust. Its depth of handling is related to maximum network path reordering that the receiver expects to see during the SCTP association. However as the actual reordering in number of packets is a combination of how delayed one packet may be compared to another times the actual packet rate this can grow for some applications and may need to be tuned. Thus, having the potential for setting this a more suitable value depending on the use case should be considered.

Note this sets this configuration to be used across any DTLS key context for a given SCTP Association and primary/restart usages.

Request : Configure Replay Protection

Parameters :

* SCTP Association

* Restart indication

* Configuration parameters

Reply: Replay Protection Configured

Parameters : true or false

8. Socket API Considerations

This section describes how the socket API defined in [RFC6458] needs to be extended to provide a way for the application to control the usage of the DTLS chunk.

A 'Socket API Considerations' section is contained in all SCTP-related specifications published after [RFC6458] describing an extension for which implementations using the socket API as specified in [RFC6458] would require some extension of the socket API. Please note that this section is informational only.

Please also note, that the API described in this section can change in a non-backwards compatible way during the evolution of this document due to changed functionality or gained experience during the implementation.

A socket API implementation based on [RFC6458] is extended by supporting several new IPPROTO_SCTP-level socket options and a new flag for recvmsg().

8.1. SCTP_ASSOC_CHANGE Notification

When an SCTP_ASSOC_CHANGE notification (specified in Section 6.1.1 of [RFC6458]) is delivered indicating a sac_state of SCTP_COMM_UP or SCTP_RESTART for an SCTP association where both peers support the DTLS chunk, SCTP_ASSOC_SUPPORTS_DTLS should be listed in the sac_info field.

8.2. A New Flag for recvmsg() (MSG_PROTECTED)

This flag is returned by recvmsg() in msg_flags for all user messages for which all DATA chunks were received in protected SCTP packets. This also means that if sctp_recvv() is used, MSG_PROTECTED is returned in the *flags argument.

8.3. Functions

8.3.1. sctp_dtls_nr_cipher_suites()

sctp_dtls_nr_cipher_suites() returns the number of cipher suites supported by the SCTP implementation.

The function prototype is:

```
unsigned int
sctp_dtls_nr_cipher_suites(void);
```

This function can be used in combination with sctp_dtls_cipher_suites().

8.3.2. sctp_dtls_cipher_suites()

sctp_dtls_cipher_suites() returns the cipher suites supported by the SCTP implementation.

The function prototype is:

```
int
sctp_dtls_cipher_suites(uint8_t cipher_suites[][2], unsigned int n);
```

and the arguments are

cipher_suites: An array where the supported cipher suites are stored. A cipher suite is represented by two uint8_t using the IANA assigned values in the TLS cipher suite registry [TLS-CIPHER-SUITES].

n: The number of cipher suites which can be stored in cipher_suites.

sctp_dtls_cipher_suites returns -1, if n is smaller than the number of cipher suites supported by the stack. If n is equal to or larger than the number of cipher suites supported by the SCTP implementation, the cipher suites are stored in cipher_suites and the number of supported cipher suites is returned.

8.4. Socket Options

The following table provides an overview of the IPPROTO_SCTP-level socket options defined by this section.

Option Name	Data Type	Set	Get
SCTP_DTLS_LOCAL_KMIDS	struct sctp_dtls_kmids	X	X
SCTP_DTLS_REMOTE_KMIDS	struct sctp_dtls_kmids		X
SCTP_DTLS_SET_SEND_KEYS	struct sctp_dtls_keys	X	
SCTP_DTLS_ADD_RECV_KEYS	struct sctp_dtls_keys	X	
SCTP_DTLS_DEL_RECV_KEYS	struct sctp_dtls_keys_id	X	
SCTP_DTLS_ENFORCE_PROTECTION	struct sctp_assoc_value	X	X
SCTP_DTLS_REPLAY_WINDOW	struct sctp_assoc_value	X	X
SCTP_DTLS_STATS	struct sctp_dtls_stats		X

Table 1: Socket Options

sctp_opt_info() needs to be extended to support:

* SCTP_DTLS_LOCAL_KMIDS,

* SCTP_DTLS_REMOTE_KMIDS,

* SCTP_DTLS_ENFORCE_PROTECTION,

* SCTP_DTLS_REPLAY_WINDOW, and

* SCTP_DTLS_STATS.

8.4.1. Get or Set the Local DTLS Key Management Identifiers (SCTP_DTLS_LOCAL_KMIDS)

This socket option sets the DTLS Key Management identifiers which will be sent to the peer during the handshake. It can also be used to retrieve the DTLS Key Management identifiers which were sent during the handshake.

The following structure is used as the option_value:

```
struct sctp_dtls_kmids {
        sctp_assoc_t sdk_assoc_id;
        uint16_t sdk_nr_kmids;
        uint16_t sdk_kmids[];
};
```

sdk_assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets, this parameter indicates upon which SCTP association the caller is performing the action. For setsockopt(), only SCTP_FUTURE_ASSOC can be used. For getsockopt(), it is an error to use SCTP_{CURRENT|ALL}_ASSOC.

sdk_nr_kmids: The number of entries in sdk_kmids.

sdk_kmids: The DTLS Key Management identifiers which will be or have been sent to the peer in the sequence they were contained in the DTLS Key Management Parameter and in network byte order.

This socket option can be used with setsockopt() for SCTP endpoints in the SCTP_CLOSED or SCTP_LISTEN state to configure the protection method identifiers to be sent. When used with getsockopt() on an SCTP endpoint in the SCTP_LISTEN state, the protection method identifiers which will be sent can be retrieved. If the SCTP endpoint is in a state other than SCTP_CLOSED or SCTP_LISTEN, the protection method identifiers which have been sent can be retrieved.

8.4.2. Get the Remote DTLS Key Management Identifiers (SCTP_DTLS_REMOTE_KMIDS)

This socket option reports the DTLS Key Management identifiers reported by the peer during the handshake.

The following structure is used as the option_value:

```
struct sctp_dtls_kmids {
        sctp_assoc_t sdk_assoc_id;
        uint16_t sdk_nr_kmids;
        uint16_t sdk_kmids[];
};
```

sdk_assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets, this parameter indicates upon which SCTP association the caller is performing the action. It is an error to use SCTP_{FUTURE|CURRENT|ALL}_ASSOC.

sdk_nr_kmids: The number of entries in sdk_kmids.

sdk_kmids: The DTLS Key Management identifiers reported by the peer in the sequence they were contained in the DTLS Key Management Parameter and in network byte order.

This socket option will fail on any SCTP endpoint in state SCTP_CLOSED, SCTP_COOKIE_WAIT and SCTP_COOKIE_ECHOED.

8.4.3. Set Send Keys (SCTP_DTLS_SET_SEND_KEYS)

Using this socket option allows to add a particular set of keys used for sending DTLS chunks.

The following structure is used as the option_value:

```
struct sctp_dtls_keys {
        sctp_assoc_t sdk_assoc_id;
        uint8_t sdk_cipher_suite[2];
        uint8_t sdk_restart;
        uint8_t sdk_unused; /* if sizeof(sctp_assoc_t) == 4 */
        uint64_t sdk_epoch;
        uint16_t sdk_key_len;
        uint16_t sdk_iv_len;
        uint16_t sdk_sn_key_len;
        uint8_t sdk_keys[];
};
```

sdk_assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets, this parameter indicates upon which SCTP association the caller is performing the action. It is an error to use SCTP_{FUTURE|CURRENT|ALL}_ASSOC.

sdk_cipher_suite: The cipher suite for which the keys are used.

sdk_restart: If the value is 0, the regular keys are added, if a value different from 0 is used, the restart keys are added.

sdk_unused: This field is ignored.

sdk_epoch: The epoch for which the keys are added.

sdk_key_len: The length of the key specified in sdk_keys.

sdk_iv_len: The length of the initialization vector specified in sdk_keys.

sdk_sn_key_len: The length of the sequence number key specified in sdk_keys.

sdk_keys: The key of length sdk_key_len directly followed by the initialization vector of length sdk_iv_len directly followed by the sequence number key of length sdk_sn_key_len.

This socket option can only be used on SCTP endpoints in states other than SCTP_LISTEN, SCTP_COOKIE_WAIT and SCTP_COOKIE_ECHOED. If the socket options is successful, all affected DTLS chunks sent will use the specified keys until the keys are changed again by another call of this socket option.

8.4.4. Add Receive Keys (SCTP_DTLS_ADD_RECV_KEYS)

Using this socket option allows to add a particular set of keys used for receiving DTLS chunks.

The following structure is used as the option_value:

```
struct sctp_dtls_keys {
        sctp_assoc_t sdk_assoc_id;
        uint8_t sdk_cipher_suite[2];
        uint8_t sdk_restart;
        uint8_t sdk_unused; /* if sizeof(sctp_assoc_t) == 4 */
        uint64_t sdk_epoch;
        uint16_t sdk_key_len;
        uint16_t sdk_iv_len;
        uint16_t sdk_sn_key_len;
        uint8_t sdk_keys[];
};
```

sdk_assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets, this parameter indicates upon which SCTP association the caller is performing the action. It is an error to use SCTP_{FUTURE|CURRENT|ALL}_ASSOC.

sdk_cipher_suite: The cipher suite for which the keys are used.

sdk_restart: If the value is 0, the regular keys are added, if a value different from 0 is used, the restart keys are added.

sdk_unused: This field is ignored.

sdk_epoch: The epoch for which the keys are added.

sdk_key_len: The length of the key specified in sdk_keys.

sdk_iv_len: The length of the initialization vector specified in sdk_keys.

sdk_sn_key_len: The length of the sequence number key specified in sdk_keys.

sdk_keys: The key of length sdk_key_len directly followed by the initialization vector of length sdk_iv_len directly followed by the sequence number key of length sdk_sn_key_len.

This socket option can only be used on SCTP endpoints in states other than SCTP_LISTEN, SCTP_COOKIE_WAIT and SCTP_COOKIE_ECHOED.

8.4.5. Delete Receive Keys (SCTP_DTLS_DEL_RECV_KEYS)

Using this socket option allows to remove a particular set of keys used for receiving DTLS chunks.

The following structure is used as the option_value:

```
struct sctp_dtls_keys_id {
   sctp_assoc_t sdki_assoc_id;
   uint32_t sdki_restart;
   uint64_t sdki_epoch;
}
```

sdki_assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets, this parameter indicates upon which SCTP association the caller is performing the action. It is an error to use SCTP_{FUTURE|CURRENT|ALL}_ASSOC.

sdki_restart: If the value is 0, the regular keys are removed, if a value different from 0 is used, the restart keys are removed.

sdki_epoch: The epoch for which the keys are removed.

This socket option can only be used on SCTP endpoints in states other than SCTP_CLOSED, SCTP_LISTEN, SCTP_COOKIE_WAIT and SCTP_COOKIE_ECHOED.

8.4.6. Set or Get Protection Enforcement (SCTP_DTLS_ENFORCE_PROTECTION)

Enabling this socket option on an SCTP endpoint enforces that received SCTP packets are only processed, if they are protected. All received packets with the first chunk not being an INIT chunk, INIT ACK chunk, or DTLS chunk will be silently discarded.

The following structure is used as the option_value:

```
struct sctp_assoc_value {
   sctp_assoc_t assoc_id;
   uint32_t assoc_value;
};
```

assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets, this parameter indicates upon which SCTP association the caller is performing the action. It is an error to use SCTP_{FUTURE|CURRENT|ALL}_ASSOC.

assoc_value: The value 0 represents that the option is off, any other value represents that the option is on.

This socket option is off by default on any SCTP endpoint. Once protection has been enforced by enabling this socket option on an SCTP endpoint, it cannot be disabled again. Whether protection has been enforced on an SCTP endpoint can be queried in any state other than SCTP_CLOSED. Protection can be enforced in any state other than SCTP_CLOSED, SCTP_COOKIE_WAIT and SCTP_COOKIE_ECHOED.

8.4.7. Get Statistic Counters (SCTP_DTLS_STATS)

This socket options allows to get various statistic counters for a specific SCTP endpoint.

The following structure is used as the option_value:

```
struct sctp_dtls_stats {
   sctp_assoc_t sds_assoc_id;
   uint64_t sds_dropped_unprotected;
   uint64_t sds_aead_failures;
   uint64_t sds_recv_protected;
   uint64_t sds_sent_protected;
   /* There will be added more fields before the WGLC. */
   /* There might be additional platform specific counters. */
};
```

sds_assoc_id: This parameter is ignored for one-to-one style

sockets. For one-to-many style sockets, this parameter indicates upon which SCTP association the caller is performing the action. It is an error to use SCTP_{FUTURE|CURRENT|ALL}_ASSOC.

sds_dropped_unprotected: The number of unprotected packets received for this SCTP endpoint after protection was enforced.

sds_aead_failures: The number of AEAD failures when processing received DTLS chunks.

sds_recv_protected: The number of DTLS chunks successfully processed.

sds_sent_protected: The number of DTLS chunks sent.

8.4.8. Get or Set the Replay Protection Window Size (SCTP_DTLS_REPLAY_WINDOW)

This socket option can be used to configure the replay protection for SCTP endpoints.

The following structure is used as the option_value:

```
struct sctp_assoc_value {
   sctp_assoc_t assoc_id;
   uint32_t assoc_value;
};
```

assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets, this parameter indicates upon which SCTP association the caller is performing the action. It is an error to use SCTP_{CURRENT|ALL}_ASSOC.

assoc_value: The maximum number of DTLS chunks in the replay protection window.

9. IANA Considerations

This document adds registry entries or registries in the Stream Control Transmission Protocol (SCTP) Parameters group handled by IANA:

* One new registry for the Key Management IDs

* One new SCTP Chunk Types and its corresponding Chunk Type Flags registry

* One new SCTP Chunk Parameter Type

* Two new SCTP Error Cause Codes

* A new Payload Protocol Identifier

9.1. DTLS Key Management Method Identifiers

Note: The RFC Editor is requested to replace RFC-To-Be with a reference to this document.

IANA is requested to create a new registry called "DTLS Key Management Method". This registry is part of the Stream Control Transmission Protocol (SCTP) Parameters grouping.

The purpose of this registry is to assign DTLS Key Management Method Identifier for any DTLS Key Management Method used for the extension described in this document. Each entry will be assigned a 16-bit unsigned integer value from the suitable range.

Identifier	Key Management Method Name	Reference	Contact
0	DTLS Chunk with Pre-shared cryptographic parameters	RFC-To-Be	Draft Authors
1-4095	Available for Assignment using Specification Required policy		
4096-65535	Available for Assignment using First Come, First Served policy		

Table 2: DTLS Key Management Method Identifiers

New entries in the range 0-4095 are registered following the Specification Required policy as defined by [RFC8126]. New entries in the range 4096-65535 are first come, first served.

9.2. SCTP Chunk Type

In the Stream Control Transmission Protocol (SCTP) Parameters group's "Chunk Types" registry, IANA is requested to update the reference for the DTLS chunk as depicted in Table 3 with a reference to this document.

ID Value	Chunk Type	Reference
0x41	DTLS Chunk (DTLS)	RFC-To-Be

Table 3: DTLS Chunk Type

IANA is requested to add the corresponding registration table for the chunk flags of the DTLS chunk with the initial contents shown in Table 4:

Chunk Flag Value	Chunk Flag Name	Reference
0x01	R bit	RFC-To-Be
0x02	P low order bit	RFC-To-Be
0x04	P high order bit	RFC-To-Be
0x08	Unassigned	
0x10	Unassigned	
0x20	Unassigned	
0x40	Unassigned	
0x80	Unassigned	

Table 4: DTLS Chunk Flags

9.3. SCTP Chunk Parameter Types

In the Stream Control Transmission Protocol (SCTP) Parameters group's "Chunk Parameter Types" registry, IANA is requested to update the reference for the DTLS Key Management as depicted in Table 5 with a reference to this document.

ID Value	Chunk Parameter Type	Reference
0x8006	DTLS Key Management	RFC-To-Be

Table 5: DTLS Key Management Chunk Parameter

9.4. SCTP Error Cause Codes

In the Stream Control Transmission Protocol (SCTP) Parameters group's "Error Cause Codes" registry, IANA is requested to add the new entries depicted below in Table 6 with a reference to this document.

ID Value	Error Cause Codes	Reference
100 (TBC)	Missing DTLS Chunk Support	RFC-To-Be
101 (TBC)	No Common DTLS Key Management Method	RFC-To-Be

Table 6: Error Cause Codes

The suggested cause code will need to be confirmed by IANA.

9.5. SCTP Payload Protocol Identifier

In the Stream Control Transmission Protocol (SCTP) Parameters group's "Payload Protocol Identifiers" registry, IANA is requested to update the reference for the PPID 4242 as depicted in Table 7 with a reference to this document.

ID Value	SCTP Payload Protocol Identifier	Reference
4242	DTLS Key Management Messages	RFC-To-Be

Table 7: Protection Operator Protocol Identifier Registered

10. Security Considerations

All the security and privacy considerations of the security protocol used as the Chunk Protection Operator applies.

DTLS replay protection MUST NOT be turned off.

10.1. Privacy Considerations

Use of the SCTP DTLS chunk provides privacy to SCTP by protecting user data and much of the SCTP control signaling. The SCTP association is identifiable based on the 5-tuple where the destination IP and port are fixed for each direction. Advanced privacy features such as sequence number encryption might therefore have limited effect.

10.2. AEAD Limit Considerations

Section 4.5.3 of [RFC9147] defines limits on the number of records q that can be protected using the same key as well as limits on the number of received packets v that fail authentication with each key. To adhere to these limits the DTLS Key Management Method can periodically poll the DTLS protection operation function to see when a limit has been reached or is close to being reached. Instead of periodic polling, a callback can be used.

10.3. Downgrade Attacks

Downgrade attacks may attempt to force the DTLS Key Management Method by altering the content of INIT chunk, for instance by removing all offered DTLS Key Management Methods but the one desired. This is possible if the attacker is an on-path attacker that can modify packet because INIT and INIT ACK chunks are plain text.

Preventing the downgrade attacks is implemented by using at the initiator the list of offered DTLS Key Management Method sent in the INIT chunk plus the selected DTLS Key Management Method received in the INIT ACK chunk from the responder for deriving the keys from the handshaked secrets obtained during DTLS initial handshake. At the responder, the list of offered DTLS Key Management Methods received in the INIT chunk plus the selected DTLS Key Management Method that is sent in the INIT ACK chunk will be used for deriving the keys from the handshaked secrets obtained during DTLS initial handshake.

If the attacker succeeds in changing the DTLS Key Management Methods in either INIT, INIT ACK or both chunks, the peers will not be able to derive the same keys and the Association will not be possible to proceed.

Thus, as long as the DTLS Key Management Method includes the ordered list of protection solutions indicators present in the parameter part of the INIT chunk for the SCTP Association in its key-derivation the association will be protected from down-grade.

In case any DTLS Key Management Method does not include the parameter content in its key-derivation down-grade might be possible if that DTLS Key Management Method method is selected. It is up to endpoint policies to determine which protection it deems necessary against down-grade attacks.

10.4. Persistent Secure Storage of Restart Key Context

The Restart DTLS Key Context needs to be stored securely and persistently. Securely as access to this security context may enable an attacker to perform a restart, resulting in a denial of service on the existing SCTP Association. It can also give the attacker access to the ULP. Thus the storage needs to provide at least as strong resistant against exfiltration as the main DTLS Key Context store.

When it comes to how to realize persistent storage that is highly dependent on the ULP and how it can utilize restarted SCTP Associations. One way can be to have an actual secure persistent storage solution accessible to the endpoint. In other use cases the persistence part might be accomplished by keeping the current restart DTLS Key Context with the ULP State if that is sufficiently secure.

11. Acknowledgments

The authors thank Hannes Tschofenig and Tirumaleswar Reddy for their participation in the design team and their contributions to this document. We also like to thank Amanda Baber with IANA for feedback on our IANA registry.

12. References

12.1. Normative References

[RFC4820] Tuexen, M., Stewart, R., and P. Lei, "Padding Chunk and Parameter for the Stream Control Transmission Protocol (SCTP)", RFC 4820, DOI 10.17487/RFC4820, March 2007, <https://www.rfc-editor.org/info/rfc4820>.

[RFC4895] Tuexen, M., Stewart, R., Lei, P., and E. Rescorla, "Authenticated Chunks for the Stream Control Transmission Protocol (SCTP)", RFC 4895, DOI 10.17487/RFC4895, August 2007, <https://www.rfc-editor.org/info/rfc4895>.

[RFC5061] Stewart, R., Xie, Q., Tuexen, M., Maruyama, S., and M. Kozuka, "Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration", RFC 5061, DOI 10.17487/RFC5061, September 2007, <https://www.rfc-editor.org/info/rfc5061>.

[RFC6083] Tuexen, M., Seggelmann, R., and E. Rescorla, "Datagram Transport Layer Security (DTLS) for Stream Control Transmission Protocol (SCTP)", RFC 6083, DOI 10.17487/RFC6083, January 2011, <https://www.rfc-editor.org/info/rfc6083>.

[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>.

[RFC9147] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/info/rfc9147>.

[RFC9260] Stewart, R., Txen, M., and K. Nielsen, "Stream Control Transmission Protocol", RFC 9260, DOI 10.17487/RFC9260, June 2022, <https://www.rfc-editor.org/info/rfc9260>.

[TLS-CIPHER-SUITES] "TLS Cipher Suites", November 2023, <https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-4>.

[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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

12.2. Informative References

[RFC6458] Stewart, R., Tuexen, M., Poon, K., Lei, P., and V. Yasevich, "Sockets API Extensions for the Stream Control Transmission Protocol (SCTP)", RFC 6458, DOI 10.17487/RFC6458, December 2011, <https://www.rfc-editor.org/info/rfc6458>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[I-D.ietf-tsvwg-rfc4895-bis] Txen, M., Stewart, R. R., Lei, P., and H. Tschofenig, "Authenticated Chunks for the Stream Control Transmission Protocol (SCTP)", Work in Progress, Internet-Draft, draft-ietf-tsvwg-rfc4895-bis-05, 21 April 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-rfc4895-bis-05>.

[I-D.ietf-tsvwg-dtls-chunk-key-management]
Txen, M., Tschofenig, H., and T. Reddy.K, "Using Datagram Transport Layer Security (DTLS) for Key Management for the Stream Control Transmission Protocol (SCTP) DTLS Chunk", Work in Progress, Internet-Draft, draft-ietf-tsvwg-dtls-chunk-key-management-00, 4 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-dtls-chunk-key-management-00>.

[I-D.westerlund-tsvwg-sctp-DTLS-handshake]
Westerlund, M., Preu Mattsson, J., and C. Porfiri, "Datagram Transport Layer Security (DTLS) in the Stream Control Transmission Protocol (SCTP) DTLS Chunk", July 2025, <https://datatracker.ietf.org/doc/draft-westerlund-tsvwg-sctp-dtls-handshake/>.

[ETSI-TS-38.413]
"NG Application Protocol (NGAP) version 18.5.0 Release 18", n.d., <https://www.etsi.org/deliver/etsi_ts/138400_138499/138413/18.05.00_60/ts_138413v180500p.pdf>.

Authors' Addresses

Magnus Westerlund
Ericsson
Email: magnus.westerlund@ericsson.com

John Preu Mattsson
Ericsson
Email: john.mattsson@ericsson.com

Claudio Porfiri
Ericsson
Email: claudio.porfiri@ericsson.com

Michael Txen
Mnster University of Applied Sciences
Stegerwaldstrasse 39
48565 Steinfurt
Germany
Email: tuexen@fh-muenster.de