TSVWG
Internet-Draft
Intended status: Standards Track
Expires: 23 April 2026

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

20 October 2025

Stream Control Transmission Protocol (SCTP) DTLS Chunk
draft-ietf-tsvwg-sctp-dtls-chunk-01

Abstract

This document describes a method for adding Cryptographic protection to the Stream Control Transmission Protocol (SCTP). The SCTP DTLS chunk defined in this document is intended to enable communications 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.

Applications using SCTP DTLS chunk can use all transport features provided by SCTP and its extensions but with some limitations.

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

This document defines a DTLS chunk for the Stream Control Transmission Protocol (SCTP), as defined in [RFC9260].

This specification defines the actual DTLS chunk, how to enable its usage, how it interacts with the SCTP association establishment to enable endpoint authentication, key-establishment, and key updates.

The DTLS chunk is designed to enable mutual authentication of endpoints, data confidentiality, data origin authentication, data integrity protection, and data replay protection for SCTP packets after the SCTP association has been established. It is dependent on a key management function that is defined separately to achieve all these capabilities. The key management function uses an API to provision the SCTP association's DTLS chunk protection with key-material to enable and rekey the protection operations.

Applications using SCTP DTLS chunk can use most transport features provided by SCTP and its extensions. However, there can be some limitations or additional requirements for them to function such as those noted for SCTP restart and use of Dynamic Address Reconfiguration, see Section 3.7 and Section 3.8. Due to its level of integration as discussed in next section it will provide its security functions on all content of the SCTP packet, and will thus not impact the potential to utilize any SCTP functionalities or extensions that are possible to use between two SCTP peers with full security and SCTP association state.

DTLS is considered version 1.3 as specified in [RFC9147] whereas other versions are explicitly not part of this document.

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. Overview

3.1. Protocol Overview

The DTLS chunk can be used for secure and confidential transfer of SCTP packets. This is implemented inside the SCTP protocol. Once an SCTP packet has been received and the SCTP common header has been used to identify the SCTP association, the DTLS chunk is processed by the Chunk Protection Operator that will perform replay protection, decrypt, verify authenticity, and if the DTLS chunk is successfully processed provides the protected SCTP chunks for further processing. Figure 1 is an example illustrating the DTLS chunk processing in regard to SCTP and the Upper Layer Protocol (ULP) using DTLS 1.3 for key management. Here the Key Management function contains validation, i.e. using certificates, handshaking, updating policies etc.

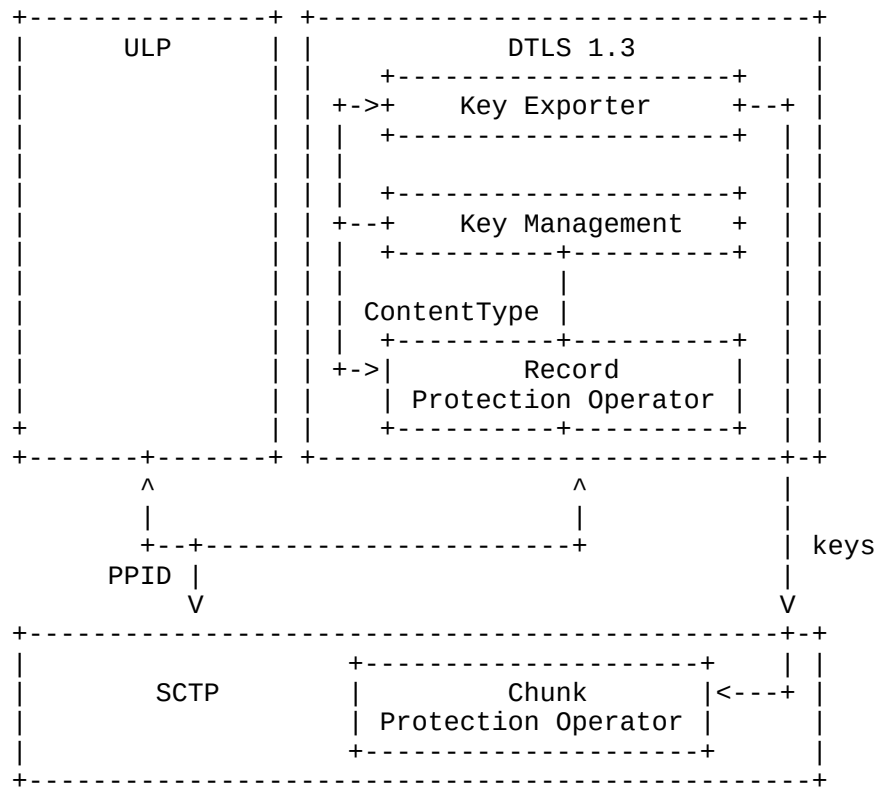

Figure 1: DTLS Chunk Layering in Regard to SCTP and ULP

In the outgoing direction, once the SCTP stack has created the unprotected SCTP packet containing control and/or DATA chunks, SCTP chunks will be processed by the Chunk Protection Operator to be protected. The result of this computation is a DTLS 1.3 record encapsulated in a SCTP chunk which is named the DTLS chunk.

The Chunk Protection Operator performs protection operations on all chunks of an SCTP packet. Information protection is kept during the lifetime of the association and no information is sent unprotected except the initial SCTP handshake, any initial key-management traffic, the SCTP common header, the SCTP DTLS chunk header, and the INIT and INIT-ACK chunks during an SCTP Restart procedure.

The support of the DTLS chunk and the key-management method to use is negotiated by the peers at the setup of the SCTP association using a new parameter. Key management and application traffic is multiplexed using the PPID. The dedicated PPID 4242 is defined for use by key management for DTLS chunk. The key management function uses an API to key the Chunk protection operation function. Usage of the DTLS 1.3 handshake for initial mutual authentication and key establishment as well as periodic re-authentication and rekeying with Diffe-Hellman of the DTLS chunk protection is defined in separate documents, (see Section 3.3). To prevent downgrade attacks of the key-management negotiation the key-management should implement specific procedures when deriving keys.

When the endpoint authentication and key establishment has been completed, the association is considered to be secured and the ULP is informed about that. From this time on it’s possible for the ULPs to exchange data securely with its peer.

A DTLS chunk will never be retransmitted, retransmission is implemented by SCTP endpoint at chunk level as specified in [RFC9260]. DTLS replay protection will be used to suppress duplicated DTLS chunks.

3.2. DTLS Considerations

The DTLS Chunk architecture splits DTLS 1.3 as shown in Figure 1, where the Key Management functionality is done at DTLS 1.3 block level, acting as a parallel User Level Protocol and a Chunk Protection Operator functionality inside the SCTP Protocol Stack.

Key Management is the set of data and procedures that take care of key distribution, verification, and update, DTLS connection setup, update and maintenance.

Chunk Protection Operator functionality is the set of data and procedures taking care of User Data encryption into DTLS Record and DTLS record decryption into User Data.

DTLS 1.3 operations requires to directly handshake messages with the remote peer for connection setup and other features, this kind of handshake is part of the Key Management functionality. Key Management function achieves these features behaving as a user of the SCTP association. Key Management sends and receives its own data via the SCTP User Level interface. Key Management's own data are distinguished from any other data by means of a dedicated PPID using the value 4242 (see Table 9).

Once Key Management has established a DTLS 1.3 connection, it can derive primary and restart keys and set the Chunk Protection Operator for SCTP Packet Payload encryption/decryption via an API to create the necessary DTLS key contexts. Both a DTLS Key context for normal use (primary) and a DTLS Key context for SCTP association restart needs to be created.

In this document we use the terms DTLS Key context for indicating a Key and IV, produced by the 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 four-value tuple identifying the context, consisting of SCTP Association, the restart indicator, the DTLS Connection ID (if used), and the DTLS epoch.

Support of DTLS Connection ID in the DTLS Record layer used in the DTLS Chunk is OPTIONAL, and negotiated using the key-management function.

The first established DTLS key context for any SCTP association and DTLS connection ID (if used) SHALL use epoch=3. This ensures that the epoch of the DTLS key context will normally match the epoch of a DTLS key-management connection.

The Replay window for the DTLS Sequence Number will need to take into account that heartbeat (HB) chunks are sent concurrently over all paths in multihomed Associations, thus it needs to be large enough to accommodate latency differences.

3.3. Considerations about SCTP Protection Solutions

This document specifies the mechanisms for SCTP to be protected with DTLS, it doesn't specify how the Key Management works, being limited on what the Key Management SHALL provide for achieving the protection. Even though DTLS1.3 is indicated as protocol for providing Key Contexts, different implementations can achieve that and different mechanisms may be used for features such as mutual authentication, rekeying etc. The DTLS Chunk solution may use a number of Key Management mechanisms depending on what is being implemented and available and/or according to the local policies. Key Management methods are called here Protection Solutions, they are defined in their own specific documents, and needs to be registered in the IANA Registry "SCTP Protection Solutions" to get their own unique identifier. This document constitutes a requirement towards any SCTP Protection Solution.

Currently there are two in-band DTLS key management solutions defined, they have different properties. See [I-D.ietf-tsvwg-dtls-chunk-key-management] and [I-D.westerlund-tsvwg-sctp-DTLS-handshake].

3.4. SCTP DTLS Chunk Buffering and Flow Control

DTLS 1.3 operations and SCTP are asynchronous, meaning that the Chunk Protection Operator may deliver the decrypted SCTP Payload to the SCTP endpoint without respecting the reception order. It's up to SCTP endpoint to reorder the chunks in the reception buffer and to take care of the flow control according to what specified in [RFC9260]. From SCTP perspective the DTLS chunk processing is part of the transport network.

Even though the above allows the implementors to adopt a multithreading design of the Chunk Protection Operators, the actual implementation should consider that out-of-order handling of SCTP chunks is not desired and may cause false congestion signals and trigger retransmissions.

3.5. PMTU Considerations

The addition of the DTLS chunk to SCTP reduces the room for payload, due to the size of the DTLS chunk header, padding, and the AEAD authentication tag. Thus, the SCTP layer creating the plain text payload needs to know about the overhead to adjust its target payload size appropriately.

A path MTU discovery function in SCTP will need to know the actual sent and received size of packets for the SCTP packets. This to correctly handle PMTUD probe packets.

From SCTP perspective, if there is a maximum size of plain text data that can be protected by the Chunk Protection Operator that must be communicated to SCTP. As such a limit will limit the PMTU for SCTP to the maximum plain text plus DTLS chunk and algorithm overhead plus the SCTP common header.

3.6. Congestion Control Considerations

The SCTP mechanism for handling congestion control does depend on successful data transfer for enlarging or reducing the congestion window CWND (see [RFC9260] Section 7.2).

It may happen that Chunk Protection Operator discards packets due to replay protection, or integrity errors depending on network induced bit errors or malicious modifications. As those packets do not represent what the peer sent, it is acceptable to ignore them, although in-situ modification on the path of a packet resulting in discarding due to integrity failure will leave a gap, but has to be accepted as part of the path behavior.

The Chunk Protection Operator will not interfere with the SCTP congestion control mechanism, this basically means that from SCTP perspective the congestion control is exactly the same as how specified in [RFC9260].

3.7. Dynamic Address Reconfiguration Considerations

When using Dynamic Address Reconfiguration [RFC5061] in an SCTP association using DTLS Chunk the ASCONF chunk is protected, thus it needs to be unprotected first, furthermore it MAY come from an unknown IP Address. In order to properly address the ASCONF chunk to the relevant Association for being unprotected, Destination Address, Source, Destination ports and VTag shall be used. If the combination of those parameters is not unique the implementor MAY choose to send the DTLS Chunk to all Associations that fit with the parameters in order to find the right one. The association will attempt de-protection operations on the DTLS chunk, and if that is successful the ASCONF chunk can be processed. Note that trial decoding should have a limit in number of tried contexts to prevent denial of service attacks on the endpoint.

The section 4.1.1 of [RFC5061] specifies that ASCONF message are required to be sent authenticated with SCTP-AUTH [RFC4895]. For SCTP associations using DTLS Chunk this results in the use of redundant

mechanism for Authentication with both SCTP-AUTH and the DTLS Chunk. We recommend to amend [RFC5061] for including DTLS Chunks as Authentication mechanism for ASCONF chunks.

3.8. 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 achieving a Restart of the current Association, thus implementing SCTP Restart.

This specification doesn’t support SCTP Restart as described in [RFC9260] because the COOCKIE-ECHO and COOKIE-ACK chunks are sent encrypted (see Section 3.8.1);

When the upper layer protocols require support of SCTP Restart, as in case of 3GPP NG-C protocol [ETSI-TS-38.413], the endpoint needs to support also initiating protected SCTP Restart procedure described in Section 3.8.1. Implementing initiating protected restart procedure is RECOMMENDED, however not required as persistent secure storage of the restart DTLS Key Context is needed.

The cases where one of the SCTP Endpoints only implements initiating legacy SCTP Restart are described in Section 3.8.2.

3.8.1. Protected SCTP Restart

The protected SCTP Restart procedure keeps the security characteristics of an SCTP Association using DTLS Chunk.

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

In order to support protected SCTP Restart, the SCTP Endpoints shall 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 5.1). Both SCTP Endpoints needs to ensure that Restart DTLS key contexts is 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 SHALL NEVER 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, DTLS connection ID (if used) 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, DTLS connection ID, and epoch needs to be stored secure 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 13.4.

The SCTP Restart handshakes INIT, INIT-ACK, COOCKIE-ECHO, COOKIE-ACK exactly as in legacy SCTP Restart case; INIT, INIT-ACK SHALL be sent plain as in the legacy, whereas COOCKIE-ECHO, COOKIE-ACK Chunks SHALL 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 4). There's exactly one active Restart DTLS Context at a time, the newest. However, a crash at the time having completed the 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 it the key-management confirms the new ones are committed to secure storage. This can for example be 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.

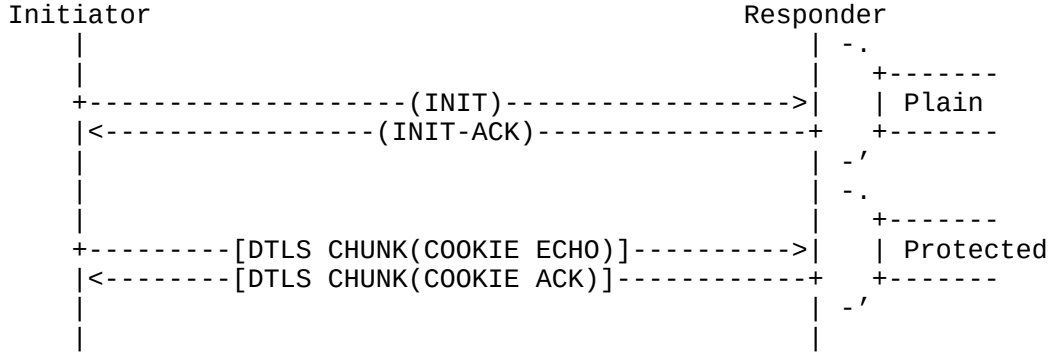

Figure 2: Handshake of SCTP Restart for DTLS in SCTP

The Figure 2 shows how the control chunks being used for SCTP Association Restart are transported within DTLS in SCTP.

The transport of COOCKIE-ECHO, COOCKIE-ACK by means of DTLS chunk ensures that the peer requesting the restart has been previously validated and the SCTP state machine after having reached ESTABLISHED state moves automatically to PROTECTED state.

A restarted SCTP Association SHALL 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 SHALL use epoch=3, i.e. the epoch value is reset. Note that if the restart epoch used also was 3 when not using any DTLS connection ID, then the installation of the new restart DTLS key context needs to be done with some care to avoid dropping valid packets. 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.

3.8.2. Compatibility with Legacy SCTP Restart

An SCTP Endpoint supporting only legacy 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 COOCKE-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 an COOCKIE-ECHO chunk protected by DTLS chunk as described in Section 3.8.1, thus having the R bit (Restart Indicator) set in the DTLS Chunk (see Figure 4), will silently discard it.

Since an SCTP Endpoint supporting only legacy SCTP Restart and involved in an SCTP Association using DTLS Chunks cannot use SCTP Restart legacy procedure, in case of need to restart the Association it SHOULD keep on retrying initiating a new Association until the remote SCTP Endpoint have closed the existing Association (i.e. due to timeout) and will accept a new one. As alternative, depending on the Use Case and the Upper Layer protocol, it MAY use a different SCTP Source port number so that the peer SCTP Endpoint will accept the initiation of the new Association while still supervising the old one.

4. New Parameter Type

This section defines the new parameter type that will be used to negotiate the use of the DTLS 1.3 chunk during association setup, its keying method and indicate preference in relation to different keying and other security solutions. Table 1 illustrates the new parameter type.

Parameter Type	Parameter Name
0x80xx	DTLS 1.3 Chunk Protected Association

Table 1: New INIT/INIT-ACK Parameter

Note that the parameter format requires the receiver to ignore the parameter and continue processing if the parameter is not understood. This is accomplished (as described in [RFC9260], Section 3.2.1.) by the use of the upper bits of the parameter type.

4.1. DTLS 1.3 Chunk Protected Association

This parameter is used to the request and acknowledge of support of DTLS 1.3 Chunk during INIT/INIT-ACK handshake and indicate preference for keying and the preference order between multiple security solutions (if supported).

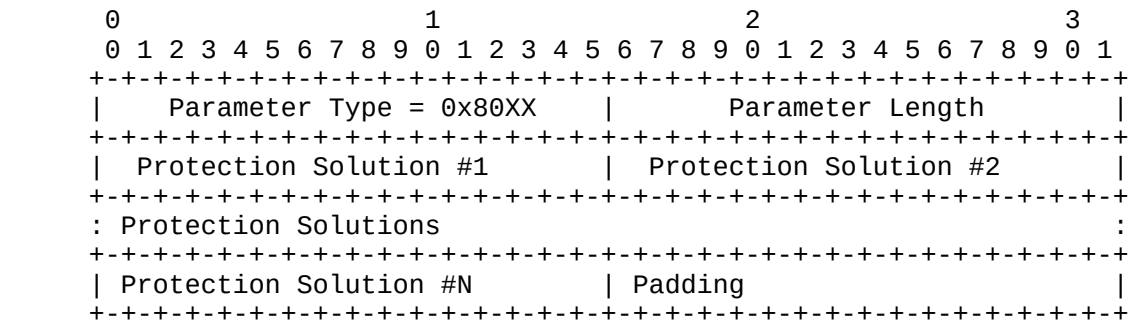

Figure 3: Protected Association Parameter

Parameter Type: 16 bits (unsigned integer)
This value MUST be set to 0x80XX.

Parameter Length: 16 bits (unsigned integer)
This value holds the length of the parameter, which will be the number of Protection Solution fields (N) times two plus 4.

Protection Solution fields: one or more 16-bit SCTP Protection Solution Identifiers:
Each Protection Solution Identifier (Section 12.1) is a 16-bit unsigned integer value indicting a Protection Solution. Protection solutions include both DTLS Chunk based, where a solution combines the DTLS chunk with a key-management solution, or non DTLS Chunk based security solution. The Protection Solutions are listed in descending order of preference, i.e. the first listed in the parameter is the most preferred and the last the least preferred by the sender in the INIT chunk. In the INIT-ACK chunk the endpoint includes all of the offered solutions which it supports and lists the selected one first. Including its decreasing preference on the additional Protection Solutions.

Padding: If the number of included Protection solutions is odd the parameter MUST be padded with two zero (0) bytes of padding to make the parameter 32-bit aligned.

RFC-Editor Note: Please replace 0x08XX with the actual parameter type value assigned by IANA and then remove this note.

5. New Chunk Type

5.1. DTLS Chunk (DTLS)

This section defines the new chunk type that will be used to transport the DTLS 1.3 record containing protected SCTP payload. Table 2 illustrates the new chunk type.

Chunk Type	Chunk Name
0x4X	DTLS Chunk (DTLS)

Table 2: DTLS Chunk Type

RFC-Editor Note: Please replace 0x4x with the actual chunk type value assigned by IANA and then remove this note.

It should be noted that the DTLS chunk format 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. This is accomplished (as described in [RFC9260] Section 3.2.) by the use of the upper bits of the chunk type.

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.

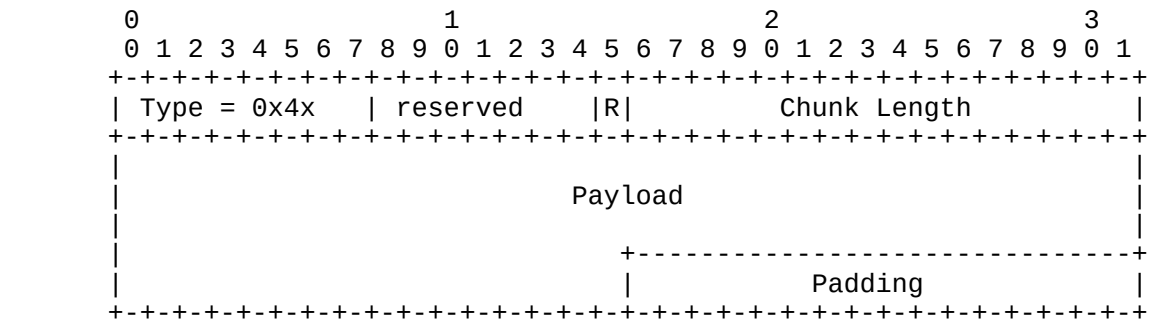

Figure 4: DTLS Chunk Structure

reserved: 7 bits Reserved bits for future use. Sender MUST set these bits to 0 and MUST be ignored on reception.

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

Chunk Length: 16 bits (unsigned integer) This value holds the length

of the Payload in bytes plus 4.

Payload: variable length This holds the DTLSCiphertext as specified in DTLS 1.3 [RFC9147].

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.

6. Error Handling

This specification introduces a new set of error causes that are to be used when SCTP endpoint detects a faulty condition. The special case is when the error is detected by the DTLS 1.3 Protection that may provide additional information.

6.1. DTLS 1.3 Chunk Protected Association Parameter Missing

When an initiator SCTP endpoint sends an INIT chunk that doesn't contain the DTLS 1.3 Chunk Protected Association or other protection solutions towards an SCTP endpoint that only accepts protected associations, SCTP will send an ABORT chunk in response to the INIT chunk (Section 5.1 of [RFC9260] including the error cause 'Policy Not Met' (TBA10) (see Section 12.4 and the DTLS 1.3 chunk protected association parameter identifier Section 4.1 in the missing param Information field. It may also include additional parameters representing other supported protection mechanisms that are acceptable per endpoint security policy.

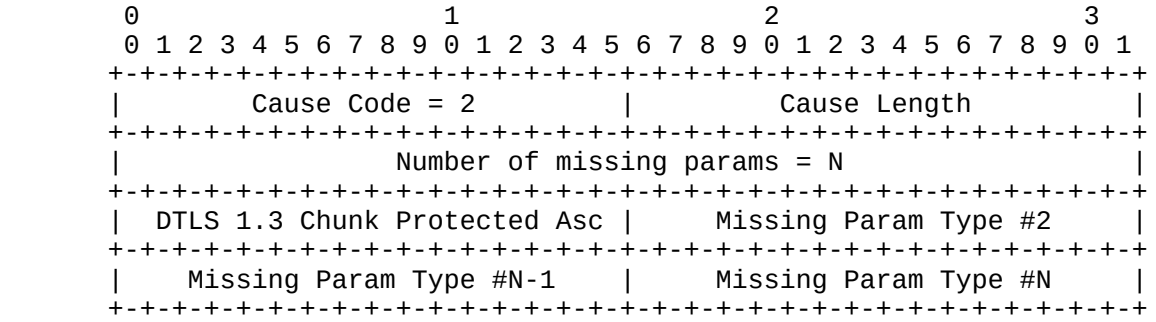

Figure 5: ERROR Missing Protected Association Parameter

Note: Cause Length in bytes is equal to following with the number of missing parameters as N: 8 + N * 2 according to [RFC9260], section 3.3.10.2. Also the Protection Association ID may be present in any of the N missing params, no order implied by the example in Figure 5.

6.2. Error in DTLS Chunk

A new Error Type is defined for the DTLS Chunk, it's used for any error related to the DTLS chunk's protection mechanism described in this document and has a structure that allows detailed information to be added as extra causes.

This specification describes some of the causes whilst the key establishment specification MAY add further causes.

When detecting an error, SCTP will send an ABORT chunk containing the relevant Error Type and Causes.

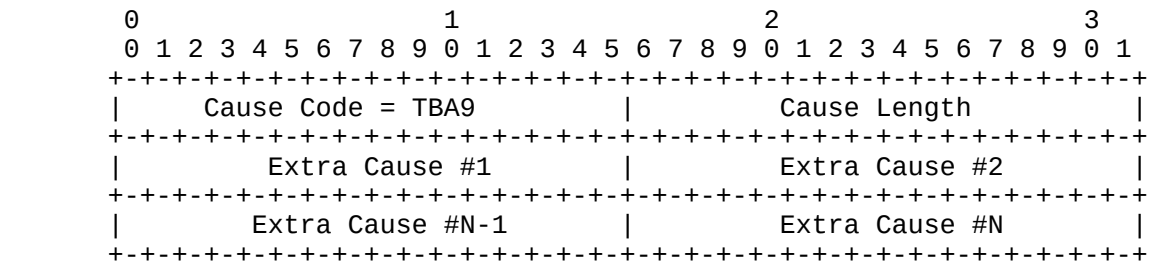

Figure 6: Error in DTLS Chunk Cause Format

Cause Code: 16 bits (unsigned integer)
The SCTP Error Chunk Cause Code indicating "Error in Protection" is TBA9.

Cause Length: 16 bits (unsigned integer)
Is for N extra Causes equal to 4 + N * 2

Extra Cause: 16 bits (unsigned integer)
Each Extra Cause indicate an additional piece of information as part of the error. There MAY be zero to as many as can fit in the extra cause field in the ERROR Chunk (A maximum of 32764).

Editor's Note: Please replace TBA9 above with what is assigned by IANA.

Below a number of defined Error Causes (Extra Cause above) are defined, additional causes can be registered with IANA following the rules in Section 12.4.

6.2.1. No Common Protection Solution

If the responder to do not support any of the protection solutions offered by the association initiator in the Protection Soluiton Parameters Figure 3 SCTP will send an ABORT chunk in response to the INIT chunk (Section 5.1 of [RFC9260], including the error cause "No Common Protection" (TBA11) (see Section 12.4).

6.3. Critical Error from DTLS

The Chunk Protection Operator MAY inform local SCTP endpoint about errors. When an Error in the DTLS 1.3 compromises the protection mechanism, the Chunk Protection Operator may stop processing data altogether, thus the local SCTP endpoint will not be able to send or receive any chunk for the specified Association. This will cause the SCTP Association to be closed by legacy timer-based mechanism. Since the Association protection is compromised no further data will be sent and the remote peer will also experience timeout on the Association.

6.4. Non-critical Error in the Protection

A non-critical error in Chunk Protection Operator means that the Chunk Protection Operator is capable of recovering without the need of the whole SCTP Association to be re-established.

From SCTP perspective, a non-critical error will be perceived as a temporary problem in the transport and will be handled with retransmissions and SACKS according to [RFC9260].

When the Chunk Protection Operator will experience a non-critical error, an ABORT chunk SHALL NOT be sent.

7. Procedures

7.1. Establishment of a Protected Association

An SCTP Endpoint acting as initiator willing to create a DTLS 1.3 chunk protected association shall send to the remote peer an INIT chunk containing the DTLS 1.3 Chunk Protected Association parameter (see Section 4.1) indicating supported and preferred key-management solutions (see Figure 3).

An SCTP Endpoint acting as responder, when receiving an INIT chunk with DTLS 1.3 Chunk Protected Association parameter, will reply with INIT-ACK with its own DTLS 1.3 Chunk Protected Association parameter containing the selected protection solution out of the set of supported ones. In case there are no common set of supported

solutions that are accepted by the responder, and the endpoints policy require secured association it SHALL reply with an ABORT chunk, include the error cause "No Common Protection" (TBA11) (see Section 12.4). Otherwise, the responder MAY send an INIT-ACK without the DTLS 1.3 Chunk Protected Association parameter to indicate it is willing to create a session without security.

Additionally, an SCTP Endpoint acting as responder willing to support only protected associations shall consider an INIT chunk not containing the DTLS 1.3 Chunk Protected Association parameter or another Protection Solution accepted by own security policy solution as an error, thus it will reply with an ABORT chunk according to what specified in Section 6.1 indicating that for this endpoint mandatory DTLS 1.3 Chunk Protected Association parameter is missing.

When initiator and responder have agreed on a DTLS Chunk protected association and the key-management method by means of handshaking INIT/INIT-ACK the SCTP association establishment continues until it has reached the ESTABLISHED state.

When the SCTP session has been established follow the process defined by the selected key-management solution for establishing DTLS Key Contexts and installing them.

7.1.1. Offering Multiple Security Solutions

An initiator of an SCTP association may want to offer multiple different key-management solutions for DTLS Chunk or in combination with other security solutions in addition to DTLS 1.3 chunks for the SCTP association. This can be done but need to consider the downgrade attack risks (see Section 13.3).

The initiator MAY include in its INIT additional security solutions that are compatible to offer in parallel with DTLS 1.3 Chunks. This may include SCTP-AUTH [I-D.ietf-tsvwg-rfc4895-bis]. This will result in that a number of different SCTP parameters may be included that are not possible to use simultaneously. Instead the responder needs to parse these parameters to figure out which sets of solutions that are offered that the implementation support, and apply its security policies to select the most appropriate. For example an offer of DTLS 1.3 Chunks and SCTP-AUTH, could be interpreted as three different solutions with different properties, namely DTLS 1.3 Chunks, DTLS/SCTP [RFC6083], and SCTP-AUTH [I-D.ietf-tsvwg-rfc4895-bis] only. However, here the DTLS 1.3 Chunk Protected Association Parameter can indicate both preference and which of the solutions that are preferred.

The responder selects one or possibly more of compatible security solutions that can be used simultaneously and include them in the response (INIT-ACK). If DTLS 1.3 chunks was selected and the Key-Management method follows the recommendation for down-grade prevention the endpoints can know that down-grade did not happen.

7.2. Termination of a Protected Association

Besides the procedures for terminating an association explained in [RFC9260], DTLS 1.3 chunk SHALL ask the SCTP endpoint for terminating an association when having an internal error or by detecting a security violation. Note that the closure of any key-management connection doesn’t compromise the capability of sending and receiving protected SHUTDOWN-COMPLETE chunks as that capability only relies on the Key Context and not on the key-management connection from where it has been derived.

7.3. Considerations on Key Management

It is up to the upper layer to manage the keys for the DTLS chunk. The meaning of key-management is described in Section 3.3.

The key-management SHOULD use a dedicated PPID to ensure that the key-management related user messages are handled by the appropriate layer.

When performing key-management, the keys for receiving SHOULD be installed before the corresponding send keys at the peer. For mitigating downgrade attacks the key derivation MUST include the protection solution Identifiers that were sent and received.

The communication is only protected after both sides have configured the keys for sending and both sides have enforced the protection.

8. DTLS Chunk Handling

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

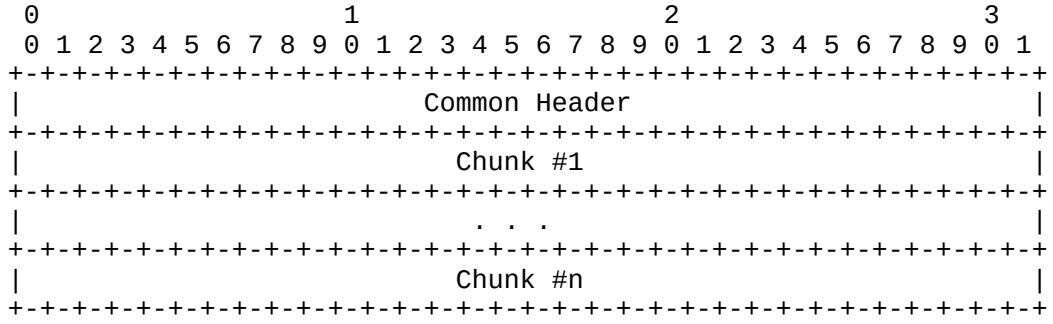

Figure 7: Unprotected SCTP Packet

The diagram shown in Figure 7 describes the structure of an unprotected SCTP packet as described in [RFC9260].

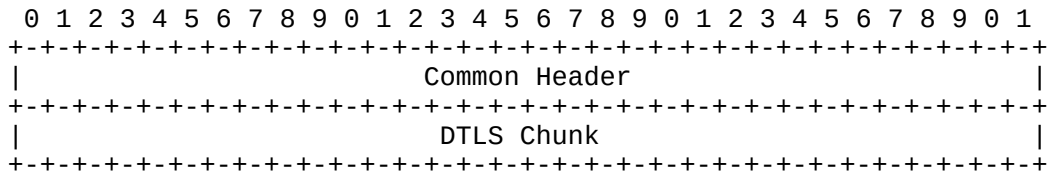

Figure 8: Protected SCTP Packets

The diagram shown in Figure 8 describes the structure of a protected SCTP packet being sent. Such packets are built with the SCTP common header. Only one DTLS chunk can be sent in a SCTP packet.

8.1. Sending of DTLS Chunks

When 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 DTLS chunk header and the overhead of DTLS has to be considered to ensure that the final SCTP packet does not exceed the PMTU.

8.2. Receiving of DTLS Chunks

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.

9. Abstract API

This section describes an abstract API that is needed between a key-managment function and the DTLS 1.3 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 the used by the cipher suite for DTLS record protection (Section 5.2 of [RFC8446]. The initilization 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]).

9.1. Cipher Suit Capabilities

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

Request : Get Cipher Suits

Parameters : none

Reply : Cipher Suits

Parameters : list of cipher suits

9.2. Establish Client Write Keying Material

The DTLS Chunk can use one of out of multiple sets of cipher suit 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 Connection ID: : If DTLS connection ID (CID) has been negotiated by the key-management its field length and value are include. The field length can be set to zero (0) to indicate that CID is not used.

* 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 Suit: 2 bytes cipher suit identification for the DTLS 1.3 Cipher suit 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 suit specific binary object containing all necessary information for protection operations. The secret will used by the DTLS 1.3 client to encrypt the record. Binary arbitrary long object depending on the cipher suit used.

Reply : Established or Failed

9.3. Establish Server Write Keying Material

The DTLS Chunk can use one of out of multiple sets of cipher suit 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 Connection ID: : If DTLS connection ID (CID) has been negotiated by the key-management its field length and value are include. The field length can be set to zero (0) to indicate that CID is not used.

* DTLS Epoch: The DTLS epoch these keys are valid for. Note that Epoch lower than 3 are note expected as they are used during DTLS handshake.

* Cipher Suit: 2 bytes cipher suit identification for the DTLS 1.3 Cipher suit 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 suit specific binary object containing all necessary information for protection operations. The secret will used by the DTLS 1.3 client to encrypt the record. Binary arbitrary long object depending on the cipher suit used.

Reply : Established or Failed

9.4. 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 CID

* DTLS Epoch

Reply: Destroyed

Parameters : true or false

9.5. 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 CID
* DTLS Epoch

Reply: Destroyed

Parameters : true or false

9.6. 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 CID
* DTLS Epoch

Reply: Key set

Parameters : true or false

9.7. 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

9.8. 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 CID

* DTLS Epoch

Reply: AEAD Encryption Invocations

Parameters : non-negative integer

9.9. 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 CID

* DTLS Epoch

Reply: AEAD Decryption Invocations

Parameters : non-negative integer

9.10. 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 CID
* DTLS Epoch

Reply: Failed AEAD Decryption Invocations

Parameters : non-negative integer

9.11. 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 are 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

10. 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().

10.1. 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.

10.2. 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 where received in protected SCTP packets. This also means that if sctp_recvv() is used, MSG_PROTECTED is returned in the *flags argument.

10.3. Functions

10.3.1. sctp_dtls_nr_cipher_suits()

sctp_dtls_nr_cipher_suits() returns the number of cypher suits supported by the SCTP implementation.

The function prototype is:

```
unsigned int
sctp_dtls_nr_cipher_suits(void);
```

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

10.3.2. sctp_dtls_cipher_suits()

sctp_dtls_cipher_suits() returns the cypher suits supported by the SCTP implementation.

The function prototype is:

```
int
sctp_dtls_cipher_suits(uint8_t cipher_suits[][2], unsigned int n);
```

and the arguments are

cipher_suits: An array where the supported cypher suits are stored. A cypher suit is represented by two uint8_t using the IANA assigned values in the TLS cipher suit registry [TLS-CIPHER-SUITS].

n: The number of cipher suits which can be stored in cipher_suits.

sctp_dtls_cipher_suits 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 the the SCTP implementation, the cipher suits are stored in cipher_suits and the number of supported cipher suits is returned.

10.4. 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_PMIDS	struct sctp_dtls_pmids	X	X
SCTP_DTLS_REMOTE_PMIDS	struct sctp_dtls_pmids		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 3: Socket Options

sctp_opt_info() needs to be extended to support:

* SCTP_DTLS_LOCAL_PMIDS,

* SCTP_DTLS_REMOTE_PMIDS,

* SCTP_DTLS_ENFORCE_PROTECTION,

* SCTP_DTLS_REPLAY_WINDOW, and

* SCTP_DTLS_STATS.

10.4.1. Get or Set the Local Protection Method Identifiers (SCTP_DTLS_LOCAL_PMIDS)

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

The following structure is used as the option_value:

```
struct sctp_dtls_pmids {
        sctp_assoc_t sds_assoc_id;
        uint16_t sdp_nr_pmids;
        uint16_t sdp_pmids[];
};
```

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. For setsockopt(), only SCTP_FUTURE_ASSOC can be used. For getsockopt(), it is an error to use SCTP_{CURRENT|ALL}_ASSOC.

sdp_nr_pmids: The number of entries in sdp_pmids.

sdp_pmids: The protection method identifiers which will be or have been sent to the peer in the sequence they were contained in the DTLS 1.3 Chunk Protected Association parameter and in host 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 that SCTP_CLOSED or SCTP_LISTEN, the protection method identifiers which have been sent can be retrieved.

10.4.2. Get the Remote Protection Method Identifiers (SCTP_DTLS_REMOTE_PMIDS)

This socket option reports the protection method identifiers reported by the peer during the handshake.

The following structure is used as the option_value:

```
struct sctp_dtls_pmids {
        sctp_assoc_t sds_assoc_id;
        uint16_t sdp_nr_pmids;
        uint16_t sdp_pmids[];
};
```

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.

sdp_nr_pmids: The number of entries in sdp_pmids.

sdp_pmids: The protection method identifiers reported by the peer in the sequence they were contained in the DTLS 1.3 Chunk Protected Association parameter and in host byte order.

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

10.4.3. 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_suit[2];
        uint8_t sdk_restart;
        uint8_t sdk_conn_id_len;
        uint16_t sdk_key_len;
        uint16_t sdk_iv_len;
        uint16_t sdk_sn_key_len;
        uint16_t sdk_unused;
        uint64_t sdk_epoch;
        uint8_t *sdk_conn_id;
        uint8_t *sdk_key;
        uint8_t *sdk_iv;
        uint8_t *sdk_sn_key;
};
```

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_suit: The cipher suit 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_conn_id_len: The length of the DTLS connection identifier specified in sdk_conn_id.

sdk_key_len: The length of the initialization vector specified in sdk_key.

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

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

sdk_unused: This field is ignored.

sdk_epoch: The epoch for which the keys are added.

sdk_conn_id: A pointer to the connection identifier for which the keys are added. Using NULL means that no connection identifier is used.

sdk_key: A pointer to the key.

sdk_iv: A pointer to the initialization vector.

sdk_sn_key: A pointer to the sequence number key.

This socket option can only be used on SCTP endpoints in states other then 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.

10.4.4. 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_suit[2];
        uint8_t sdk_restart;
        uint8_t sdk_conn_id_len;
        uint16_t sdk_key_len;
        uint16_t sdk_iv_len;
        uint16_t sdk_sn_key_len;
        uint16_t sdk_unused;
        uint64_t sdk_epoch;
        uint8_t *sdk_conn_id;
        uint8_t *sdk_key;
        uint8_t *sdk_iv;
        uint8_t *sdk_sn_key;
};
```

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_suit: The cipher suit 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_conn_id_len: The length of the DTLS connection identifier specified in sdk_conn_id.

sdk_key_len: The length of the initialization vector specified in sdk_key.

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

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

sdk_unused: This field is ignored.

sdk_epoch: The epoch for which the keys are added.

sdk_conn_id: A pointer to the connection identifier for which the keys are added. Using NULL means that no connection identifier is used.

sdk_key: A pointer to the key.

sdk_iv: A pointer to the initialization vector.

sdk_sn_key: A pointer to the sequence number key.

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

10.4.5. 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;
        uint8_t sdki_restart;
        uint8_t sdki_conn_id_len;
        uint16_t sdki_unused;
        uint64_t sdki_epoch;
        uint8_t *sdki_conn_id;
}
```

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_conn_id_len: The length of the DTLS connection identifier specified in sdki_conn_id.

sdki_unused: This field is ignored.

sdki_epoch: The epoch for which the keys are removed.

sdki_conn_id: A pointer to the connection identifier for which the keys are removed. Using NULL means that no connection identifier is used.

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

10.4.6. 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.

10.4.7. 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;
        uint32_t sds_dropped_unprotected;
        uint32_t sds_aead_failures;
        uint32_t sds_recv_protected;
        uint32_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.

10.4.8. 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.

11. Implementation Considerations

11.1. Privacy Padding of SCTP Packets

To reduce the potential information leakage from packet size variations one may select to pad the SCTP Packets to uniform packet sizes. This size may be either the maximum used, or in block sized increments. However, the padding needs to be done inside of the encryption envelope.

Both SCTP and DTLS contains mechanisms to pad SCTP payloads, and DTLS records respectively. If padding of SCTP packets are desired to hide actual message sizes it RECOMMEDED 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 needs to be noted that independent if SCTP padding or DTLS padding is used the padding is not taken into account by the SCTP congestion control. Extensive use of padding has potential for worsen congestion situations as the SCTP association will consume more bandwidth than its derived share by the congestion control.

The use of SCTP PAD chunk is recommended as it at least can enable future extension or SCTP implementation that account also for the padding. Use of DTLS padding hides this packet expansion from SCTP.

12. IANA Considerations

This document defines two new registries in the Stream Control Transmission Protocol (SCTP) Parameters group that IANA maintains. Theses registries are for the extra cause codes for protection related errors. It also adds registry entries into several other registries in the Stream Control Transmission Protocol (SCTP) Parameters group:

* One new SCTP Chunk Types

* One new SCTP Chunk Parameter Type

* Three new SCTP Error Cause Code

And finally the update of one registered SCTP Payload Protocol Identifier.

12.1. SCTP Protection Solution Identifiers

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

The purpose of this registry is to assign Protection Solution Identifier for any security solution that is either the DTLS Chunk combined with a key-management method, offered as an alternative to DTLS chunk. Any security solution that is offered through a parameter exchange during the SCTP handshake are potential to be included here.

Each entry will be assigned a 16-bit unsigned integer value from the suitable range.

Identifier	Solution Name	Reference	Contact
0	DTLS 1.3 Chunk with Pre-	RFC-TBD	Draft Authors
1-4095	Available for Assignment using Specification Required policy		
4096-65535	Available for Assignment using First Come, First Served policy		

Table 4: Protection Solution 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.

12.2. SCTP Chunk Type

In the Stream Control Transmission Protocol (SCTP) Parameters group's "Chunk Types" registry, IANA is requested to add the one new entry depicted below in in Table 5 with a reference to this document. The registry at time of writing was available at: https://www.iana.org/assignments/sctp-parameters/sctp-parameters.xhtml#sctp-parameters-1

ID Value	Chunk Type	Reference
TBA6	DTLS Chunk (DTLS)	RFC-To-Be

Table 5: New Chunk Type Registered

The registration table for the chunk flags of this chunk type is initially:

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

Table 6: DTLS Chunk Flags

12.3. SCTP Chunk Parameter Types

In the Stream Control Transmission Protocol (SCTP) Parameters group's "Chunk Parameter Types" registry, IANA is requested to add the new entry depicted below in in Table 7 with a reference to this document. The registry at time of writing was available at: https://www.iana.org/assignments/sctp-parameters/sctp-parameters.xhtml#sctp-parameters-2

ID Value	Chunk Parameter Type	Reference
TBA8	DTLS 1.3 Chunk Protected Association	RFC-To-Be

Table 7: New Chunk Type Parameters Registered

12.4. 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 entry depicted below in in Table 8 with a reference to this document. The registry at time of writing was available at: https://www.iana.org/assignments/sctp-parameters/sctp-parameters.xhtml#sctp-parameters-24

ID Value	Error Cause Codes	Reference
TBA9	DTLS Chunk Error	RFC-To-Be
TBA10	Policy Not Met	RFC-To-Be
TBA11	No Common Protection	RFC-To-Be

Table 8: Error Cause Codes Parameters Registered

12.5. SCTP Payload Protocol Identifier

In the Stream Control Transmission Protocol (SCTP) Parameters group's "Payload Protocol Identifiers" registry, IANA is requested to update the entry depicted below in in Table 9 with a reference to this document. The registry at time of writing was available at: https://www.iana.org/assignments/sctp-parameters/sctp-parameters.xhtml#sctp-parameters-25

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

Table 9: Protection Operator Protocol Identifier Registered

13. 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.

13.1. 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 changing DTLS Connection ID and sequence number encryption might therefore have limited effect.

13.2. 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 key management function can periodically poll the DTLS protection operation function to see when a limit have been reached or is closed to being reached. Instead of periodic polling, a callback can be used.

13.3. Downgrade Attacks

As long as the Key-management include 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 key-management do not include the parameter content in its key-derivation down-grade might be possible if that key-management method is selected. It is up to endpoint policies to determine which protection it deems necessary against down-grade attacks.

13.4. Persistent Secure Storage of Restart Key Context

The Restart DTLS Key Context needs to be stored securely and persistent. Securely as access to this security context may enable an attacker to perform a restart, resulting 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 be keeping the current restart DTLS Key Context with the ULP State if that is sufficiently secure.

14. 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.

15. References

15.1. 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-SUITS] "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>.

15.2. 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