Network Working Group
Internet-Draft
Obsoletes: 4895 (if approved)
Intended status: Standards Track
Expires: 23 October 2025

M. T端xen
M端nster Univ. of Applied Sciences
R. Stewart

P. Lei
Netflix, Inc.
H. Tschofenig
21 April 2025

Authenticated Chunks for the Stream Control Transmission Protocol (SCTP)
draft-ietf-tsvwg-rfc4895-bis-05

Abstract

This document describes a new chunk type, several parameters, and procedures for the Stream Control Transmission Protocol (SCTP). This new chunk type can be used to authenticate SCTP chunks by using shared keys between the sender and receiver. The new parameters are used to establish the shared keys. This document obsoletes RFC 4895.

Status of This Memo

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

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

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

This Internet-Draft will expire on 23 October 2025.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components

extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction

SCTP uses 32-bit verification tags to protect itself against blind attackers. These values are not changed during the lifetime of an SCTP association.

Looking at new SCTP extensions, there is the need to have a method of proving that an SCTP chunk(s) was really sent by the original peer that started the association and not by a malicious attacker.

Since it is required to protect SCTP control data, any solution only protecting SCTP user data is not sufficient.

Therefore, an SCTP extension that provides a mechanism for deriving shared keys for each association is presented. These association shared keys are derived from endpoint pair shared keys, which are configured and might be empty, and data that is exchanged during the SCTP association setup.

The extension presented in this document allows an SCTP sender to authenticate chunks using shared keys between the sender and receiver. The receiver can then verify that the chunks are sent from the sender and not from a malicious attacker (as long as the attacker does not know an association shared key).

The extension described in this document places the result of a Hashed Message Authentication Code (HMAC) computation before the data covered by that computation. Placing it at the end of the packet would have required placing a control chunk after DATA chunks in case of authenticating DATA chunks. This would break the rule that control chunks occur before DATA chunks in SCTP packets. It should also be noted that putting the result of the HMAC computation after the data being covered would not allow sending the packet during the computation of the HMAC because the result of the HMAC computation is needed to compute the CRC32C checksum of the SCTP packet, which is placed in the common header of the SCTP packet.

The protocol extension defined in this document can be used in combination with any other currently defined SCTP extension. Its usage is required by the SCTP extension for Dynamic Address Reconfiguration as specified in [RFC5061].

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. New Parameter Types

This section defines the new parameter types that will be used to negotiate the authentication during association setup. Table 1 illustrates the new parameter types.

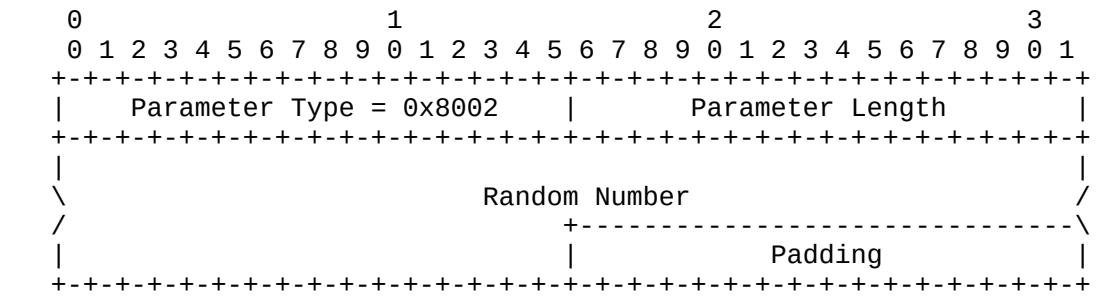
Parameter Type	Parameter Name
0x8002	Random Parameter (RANDOM)
0x8003	Chunk List Parameter (CHUNKS)
0x8004	Requested HMAC Algorithm Parameter (HMAC ALGO)
0x8006 (suggested)	All Chunks Parameter (ALL CHUNKS)

Table 1

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.

3.1. Random Parameter (RANDOM)

This parameter is used to carry a random number of an arbitrary length.



Parameter Type: 2 bytes (unsigned integer)
This value MUST be set to 0x8002.

Parameter Length: 2 bytes (unsigned integer)
This value is the length of the Random Number in bytes plus 4.

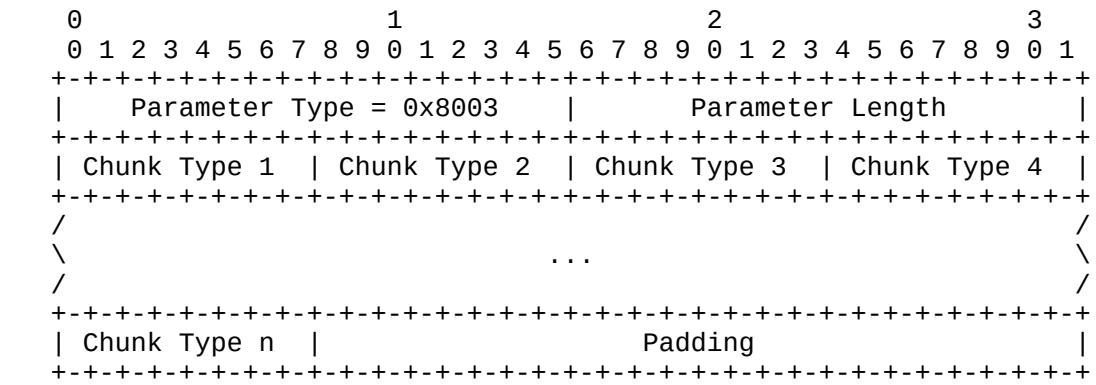
Random Number: n bytes (unsigned integer)
This value represents an arbitrary Random Number in network byte order.

Padding: 0, 1, 2, or 3 bytes (unsigned integer)
If the length of the Random Number is not a multiple of 4 bytes, the sender MUST pad the parameter with all zero bytes to make the parameter 32-bit aligned. The Padding MUST NOT be longer than 3 bytes and it MUST be ignored by the receiver.

The RANDOM parameter MUST be included once in the INIT or INIT ACK chunk, if the sender wants to send or receive authenticated chunks, to provide a 32-byte Random Number. For 32-byte Random Numbers, the Padding is empty.

3.2. Chunk List Parameter (CHUNKS)

This parameter is used to specify which chunk types are required to be authenticated before being sent by the peer.



Parameter Type: 2 bytes (unsigned integer)
This value MUST be set to 0x8003.

Parameter Length: 2 bytes (unsigned integer)
This value is the number of listed Chunk Types plus 4.

Chunk Type n: 1 byte (unsigned integer)
Each Chunk Type listed is required to be authenticated when sent by the peer.

Padding: 0, 1, 2, or 3 bytes (unsigned integer)
If the number of Chunk Types is not a multiple of 4, the sender MUST pad the parameter with all zero bytes to make the parameter 32-bit aligned. The Padding MUST NOT be longer than 3 bytes and it MUST be ignored by the receiver.

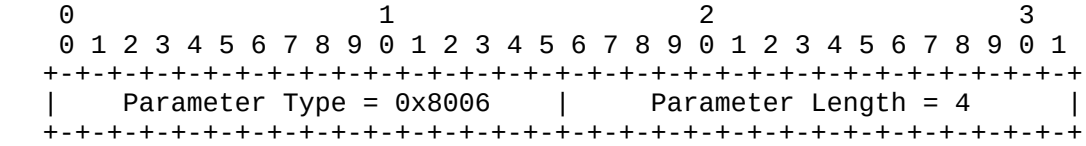
Either the CHUNKS parameter or the ALL CHUNKS parameter MUST be included once in the INIT or INIT ACK chunk if the sender wants to receive authenticated chunks. Its maximum length is 260 bytes.

The chunk types for INIT, INIT ACK, SHUTDOWN COMPLETE, and AUTH chunks MUST NOT be listed in the CHUNKS parameter. However, if a CHUNKS parameter is received then the types for INIT, INIT ACK, SHUTDOWN COMPLETE, and AUTH chunks MUST be ignored.

A CHUNKS parameter MAY contain chunk types being unknown to the receiver.

3.3. All Chunks Parameter (ALL CHUNKS)

This parameter is used to specify that all allowed chunk types are required to be authenticated before being sent by the peer.



Parameter Type: 2 bytes (unsigned integer)
This value MUST be set to 0x8006 (suggested).

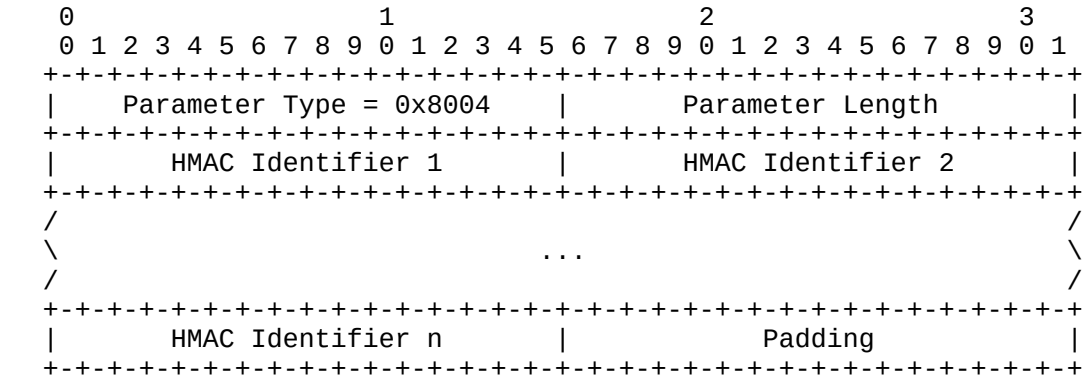
Parameter Length: 2 bytes (unsigned integer)
This value MUST be set to 4.

Either the ALL CHUNKS parameter or the CHUNKS parameter MUST be included once in the INIT or INIT ACK chunk if the sender wants to receive authenticated chunks.

The ALL CHUNKS parameter MUST NOT be included in the INIT or INIT ACK chunk, if the peer only supports [RFC4895].

3.4. Requested HMAC Algorithm Parameter (HMAC ALGO)

This parameter is used to list the HMAC Identifiers the peer MUST use.



Parameter Type: 2 bytes (unsigned integer)
This value MUST be set to 0x8004.

Parameter Length: 2 bytes (unsigned integer)
This value is the number of HMAC Identifiers multiplied by 2, plus 4.

HMAC Identifier n: 2 bytes (unsigned integer)

The values expressed are a list of HMAC Identifiers that may be used by the peer. The values are listed by preference, with respect to the sender, where the first HMAC Identifier listed is the one most preferable to the sender. Any non-deprecated HMAC Identifier MUST be listed before any deprecated HMAC Identifier.

Padding: 0 or 2 bytes (unsigned integer)
If the number of HMAC Identifiers is not even, the sender MUST pad the parameter with all zero bytes to make the parameter 32-bit aligned. The Padding MUST be 0 or 2 bytes long and it MUST be ignored by the receiver.

The HMAC ALGO parameter MUST be included once in the INIT or INIT ACK chunk if the sender wants to send or receive authenticated chunks.

Table 2 shows the currently defined values for HMAC Identifiers.

HMAC Identifier	Message Digest Algorithm
0	Reserved
1 (deprecated)	SHA-1 defined in [NIST_FIPS_180_4]
2	Reserved
3 (deprecated)	SHA-256 defined in [NIST_FIPS_180_4]
4 (suggested)	SHA-256 defined in [NIST_FIPS_180_4] with directional keys

Table 2

Every endpoint supporting SCTP chunk authentication MUST support the HMAC based on the SHA-256 algorithm with directional keys.

4. New Error Causes

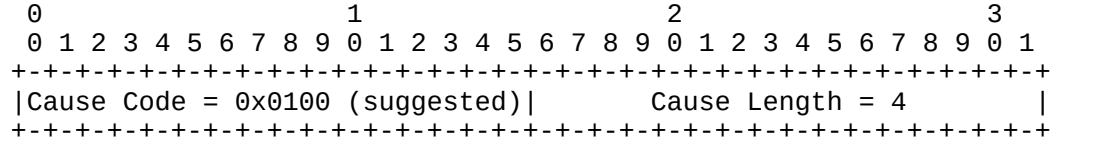
This section defines two new error causes. One that will be sent if there is a collision related to the RANDOM parameters, and one that will be sent if an AUTH chunk is received with an unsupported HMAC Identifier. Table 3 illustrates the new error cause.

Cause Code	Error Cause Name
0x0100	RANDOM Collision
0x0105	Unsupported HMAC Identifier

Table 3

4.1. RANDOM Collision Error Cause

This error cause is used to indicate a collision of Random Numbers has happened.

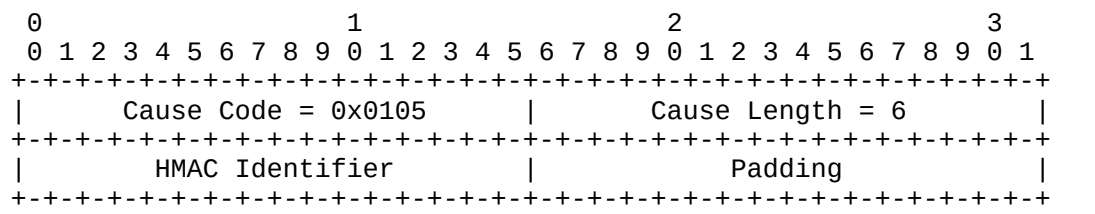


Cause Code: 2 bytes (unsigned integer)
This value MUST be set to the IANA assigned cause code for the 'RANDOM Collision' error cause. IANA is requested to assign the value 0x0100 (suggested) for this cause code

Cause Length: 2 bytes (unsigned integer)
This value MUST be set to 4.

4.2. Unsupported HMAC Identifier Error Cause

This error cause was used to indicate that an AUTH chunk has been received with an unsupported HMAC Identifier as specified in [RFC4895]. The use of this error cause is deprecated by this specification.



Cause Code: 2 bytes (unsigned integer)
This value MUST be set to 0x0105.

Cause Length: 2 bytes (unsigned integer)

This value MUST be set to 6.

HMAC Identifier: 2 bytes (unsigned integer)
This value is the HMAC Identifier which is not supported.

Padding: 2 bytes (unsigned integer)
The sender MUST pad the error cause with all zero bytes to make the cause 32-bit aligned. The Padding MUST be 2 bytes long and it MUST be ignored by the receiver.

5. New Chunk Type

This section defines the new chunk type that will be used to authenticate chunks. Table 4 illustrates the new chunk type.

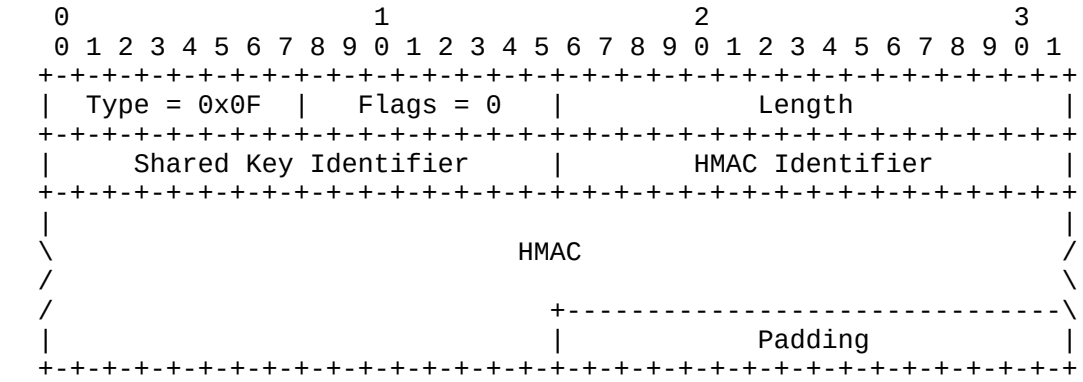
Chunk Type	Chunk Name
0x0F	Authentication Chunk (AUTH)

Table 4

It should be noted that the AUTH-chunk format requires the receiver to ignore the chunk if it is not understood and silently discard all chunks that follow. This is accomplished (as described in [RFC9260], Section 3.2.) by the use of the upper bits of the chunk type.

5.1. Authentication Chunk (AUTH)

This chunk is used to hold the result of the HMAC calculation.



Type: 1 byte (unsigned integer)
This value MUST be set to 0x0F for all AUTH-chunks.

Flags: 1 byte (unsigned integer)
SHOULD be set to zero on transmit and MUST be ignored on receipt.

Length: 2 bytes (unsigned integer)
This value holds the length of the HMAC in bytes plus 8.

Shared Key Identifier: 2 bytes (unsigned integer)
This value describes which endpoint pair shared key is used.

HMAC Identifier: 2 bytes (unsigned integer)
This value describes which message digest is being used. Table 2 shows the currently defined values.

HMAC: n bytes (unsigned integer)
This holds the result of the HMAC calculation.

Padding: 0, 1, 2, or 3 bytes (unsigned integer)
If the length of the HMAC 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.

The control chunk AUTH MUST NOT appear more than once in an SCTP packet. All control and data chunks that are placed after the AUTH chunk in the packet are sent in an authenticated way. Those chunks placed in a packet before the AUTH chunk are not authenticated. Please note that DATA chunks can not appear before control chunks in an SCTP packet.

6. Procedures

6.1. Association Shared Send and Receive Keys

6.1.1. Handling of RANDOM parameters

An SCTP endpoint willing to receive or send authenticated chunks MUST send one RANDOM parameter in its INIT or INIT ACK chunk. The RANDOM parameter MUST contain a 32-byte Random Number. The Random Number should be generated in accordance with [RFC4086].

If the length of the received Random Number is not 32 bytes, the association MUST be aborted. The ABORT chunk SHOULD contain the error cause ’Protocol Violation’.

It is unlikely but possible that an endpoint receives a packet containing an INIT chunk and all of the following three conditions are fulfilled:

1. The endpoint is in the COOKIE_WAIT or COOKIE-ECHOED state.

2. The HMAC ALGO parameter contained in the INIT chunk lists at least one non-deprecated HMAC algorithm.

3. The RANDOM parameter received in the INIT chunk is the same as the one sent in the INIT chunk it sent earlier, i.e. both Random parameter contain the same Random Number.

In this case, the endpoint MUST send a packet containing an ABORT chunk with the 'RANDOM Collision' error cause. The endpoint SHOULD retry setting up an association by sending packets with a new INIT chunk.

In any other case, an endpoint receiving a packet containing an INIT chunk with a RANDOM parameter MUST select a Random Number to be included in the INIT ACK it sends different from the Random Number received.

These rules are similar to the one for the Verification Tag in case of INIT collisions, as explained in Section 5.2.4 of [RFC9260].

If an endpoint lists at least one non-deprecated HMAC Identifier in the HMAC ALGO parameter it sends in its INIT or INIT ACK chunk, it operates not in legacy mode. Only if all HMAC Identifiers listed are deprecated, the endpoint operates in legacy mode. Operating in legacy mode means supporting SCTP authentication as specified in [RFC4895] and not as specified by this document.

After the association has been established, each endpoint knows its own Random Number and the peer's Random Number and these two Random Numbers are different if both endpoints are not operating in legacy mode.

6.1.2. Computation of the Local and Remote Key Vectors

An SCTP endpoint has a list of chunk types it only accepts if they are received in an authenticated way. If this list contains all chunk types except the ones for INIT, INIT ACK, SHUTDOWN COMPLETE, and AUTH chunks and the peer does not operate in legacy mode, the ALL CHUNKS parameter SHOULD be included in the INIT or INIT ACK chunk. If this is not the case, the chunk types are listed in the CHUNKS parameter, which is included in the INIT or INIT ACK chunk. The CHUNKS parameter MAY be omitted if it the list of chunk types is empty. Since this list does not change during the lifetime of the SCTP endpoint there is no problem in case of an INIT collision.

Each SCTP endpoint MUST include in the INIT and INIT ACK a HMAC ALGO parameter containing a list of HMAC Identifiers it requests the peer to use. The receiver of an HMAC ALGO parameter SHOULD use the first listed algorithm it supports. The HMAC algorithm based on SHA-256 with directional keys MUST be supported and included in the HMAC ALGO parameter. An SCTP endpoint MUST NOT change the parameters listed in the HMAC ALGO parameter during the lifetime of the endpoint.

The RANDOM parameter, the CHUNKS or ALL CHUNKS parameter, and the HMAC ALGO parameter sent by each endpoint MUST be concatenated in this sequence as byte vectors. Parameters that were not sent MUST be omitted from the concatenation process. These parameters include the parameter type, parameter length, and the parameter value, but padding is omitted; all padding MUST be removed from this concatenation before proceeding with further computation of keys. The resulting vector based on the parameters sent by an endpoint are called the local key vector and the resulting vector based on the parameters received by an endpint are called the remote key vector.

6.1.3. Derivation of Association Send and Receive Keys

Both endpoints of an association MAY have endpoint pair shared keys that are byte vectors and pre-configured or established by another mechanism. They are identified by the Shared Key Identifier. For each endpoint pair shared key, an association shared send and receive key are computed. If there are no endpoint pair shared keys configured, only one association shared send key and one association receive key is computed by using an empty byte vector as the endpoint pair shared key.

The way association shared send and receive keys are computed from the endpoint pair shared keys depends on the peer operating in legacy mode or not.

If the peer operates in legacy mode, the association shared send key and the association shared receive key are the same. They are computed by selecting the numerically smaller key vector and concatenating it to the endpoint pair shared key, and then concatenating the numerically larger key vector to that. If the key vectors are equal as numbers but differ in length, then the concatenation order is the endpoint pair shared key, followed by the shorter key vector, followed by the longer key vector. Otherwise, the key vectors are identical, and may be concatenated to the endpoint pair key in any order. The concatenation is performed on byte vectors, and all numerical comparisons use network byte order to convert the key vectors to a number. The result of the concatenation is the association shared send key and the association shared receive key.

If the peer does not operate in legacy mode, the send context is defined as the concatenation of local key vector followed by the remote key vector. The receive context is defined as the concatenation of the remote key vector followed by the local key vector. For deriving the association shared send and receive keys, a method described in Section 3.1 of [RFC5926] is used. The association shared send key is the result of using HMAC-SHA512 as the key derivation function with the endpoint pair shared key as the Master_Key, the send context as the Context and 512 as the Output_Length. The association shared receive key is computed the same way, just using the receive context as the Context. In both cases "SCTP-AUTH" is used as the Label.

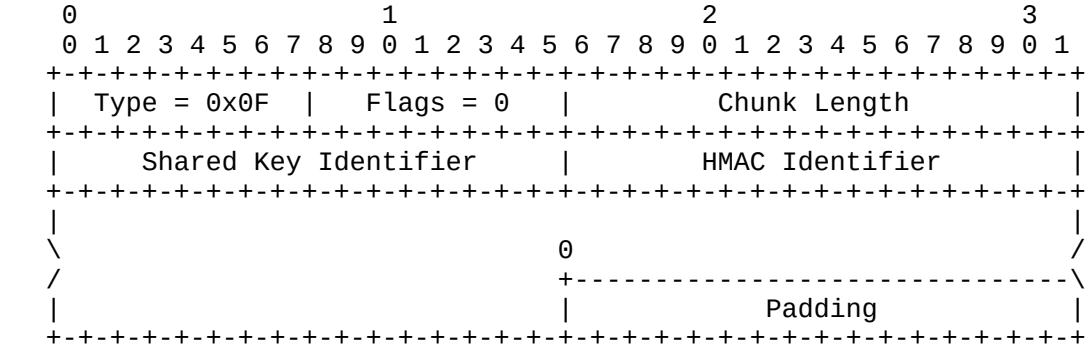
6.2. Sending Authenticated Chunks

All chunks, that have been requested by the peer to be sent authenticated, MUST be sent authenticated. The other chunks MAY be sent authenticated when possible. If endpoint pair shared keys are configured, one of them MUST be selected for authentication.

To send chunks in an authenticated way, the sender MUST include these chunks after an AUTH chunk. This means that a sender MUST bundle chunks in order to authenticate them.

If the endpoint has no endpoint pair shared key for the peer, it MUST use Shared Key Identifier zero with an empty endpoint pair shared key. If there are multiple endpoint pair shared keys the sender selects one and uses the corresponding Shared Key Identifier.

The sender MUST calculate the Message Authentication Code (MAC) (as described in [RFC2104]) using the hash function H as described by the HMAC Identifier and the shared association send key K based on the endpoint pair shared key described by the Shared Key Identifier. The ’data’ used for the computation of the AUTH-chunk is given by the AUTH chunk with its HMAC field set to zero (as depicted below) followed by all the chunks that are placed after the AUTH chunk in the SCTP packet.



Please note that all fields are in network byte order and that the field that will contain the complete HMAC is filled with zeroes. The length of the field shown as zero is the length of the HMAC described by the HMAC Identifier. The padding of all chunks being authenticated MUST be included in the HMAC computation.

The sender fills the HMAC into the HMAC field and sends the packet.

6.3. Receiving Authenticated Chunks

The receiver has a list of chunk types that it expects to be received only after an AUTH-chunk. This list has been sent to the peer during the association setup. It MUST silently discard these chunks if they are not placed after an AUTH chunk in the packet.

The receiver MUST use the HMAC algorithm indicated in the HMAC Identifier field. If this algorithm was not specified by the receiver in the HMAC ALGO parameter in the INIT or INIT ACK chunk during association setup, the AUTH chunk and all the chunks after it MUST be silently discarded. Implementations compliant with [RFC4895] SHOULD send an ERROR chunk with the error cause defined in Section 4.2. An SCTP implementations following this specification MUST silently discard ERROR chunks with the error cause defined in Section 4.2.

If an endpoint with no shared key receives a Shared Key Identifier other than 0, it MUST silently discard all authenticated chunks. If the endpoint has at least one endpoint pair shared key for the peer, it MUST use the key specified by the Shared Key Identifier if a key has been configured for that Shared Key Identifier. If no endpoint pair shared key has been configured for that Shared Key Identifier, all authenticated chunks MUST be silently discarded.

The receiver now performs the same calculation as described for the sender. It uses the shared association receive key K based on the endpoint pair shared key described by the Shared Key Identifier. If the result of the calculation is the same as given in the HMAC field, all the chunks following the AUTH chunk are processed. If the field does not match the result of the calculation, all the chunks following the AUTH chunk MUST be silently discarded.

It should be noted that if the receiver wants to tear down an association in an authenticated way only, the handling of malformed packets should not result in tearing down the association.

An SCTP implementation has to maintain state for each SCTP association. In the following, we call this data structure the SCTP transmission control block (STCB).

When an endpoint requires COOKIE ECHO chunks to be authenticated, some special procedures have to be followed because the reception of a COOKIE ECHO chunk might result in the creation of an SCTP association. If a packet arrives containing an AUTH chunk as a first chunk, a COOKIE ECHO chunk as the second chunk, and possibly more chunks after them, and the receiver does not have an STCB for that packet, then authentication is based on the contents of the COOKIE ECHO chunk. In this situation, the receiver MUST authenticate the chunks in the packet by using the RANDOM parameters, CHUNKS or ALL CHUNKS parameters and HMAC_ALGO parameters obtained from the COOKIE ECHO chunk, and possibly a local shared secret as inputs to the authentication procedure specified in Section 6.3. If authentication fails, then the packet is discarded. If the authentication is successful, the COOKIE ECHO and all the chunks after the COOKIE ECHO MUST be processed. If the receiver has an STCB, it MUST process the AUTH chunk as described above using the STCB from the existing association to authenticate the COOKIE ECHO chunk and all the chunks after it.

If the receiver does not find an STCB for a packet containing an AUTH chunk as the first chunk and does not find a COOKIE ECHO chunk as the second chunk, it MUST use the chunks after the AUTH chunk to look up an existing association. If no association is found, the packet MUST be considered as out of the blue. The out of the blue handling MUST be based on the packet without taking the AUTH chunk into account. If an association is found, it MUST process the AUTH chunk using the STCB from the existing association as described earlier.

Requiring ABORT chunks and COOKIE ECHO chunks to be authenticated makes it impossible for an attacker to bring down or restart an association as long as the attacker does not know the association shared key. But it should also be noted that if an endpoint accepts

ABORT chunks only in an authenticated way, it may take longer to detect that the peer is no longer available. If an endpoint accepts COOKIE ECHO chunks only in an authenticated way, the restart procedure does not work, because the restarting endpoint most likely does not know the association shared key of the old association to be restarted. However, if the restarting endpoint does know the old association shared key, it can successfully send the COOKIE ECHO chunk in a way that it is accepted by the peer by using this old association shared key for the packet containing the AUTH chunk. After this operation, both endpoints have to use the new association shared key.

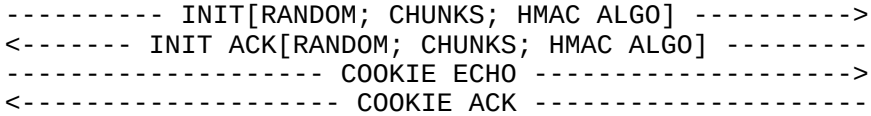
If a server has an endpoint pair shared key with some clients, it can request the COOKIE_ECHO chunk to be authenticated and can ensure that only associations from clients with a correct endpoint pair shared key are accepted.

Furthermore, it is important that the cookie contained in an INIT ACK chunk and in a COOKIE ECHO chunk MUST NOT contain any endpoint pair shared keys.

7. Examples

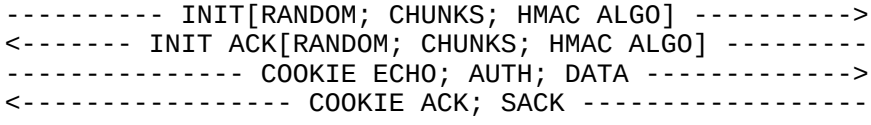
This section gives examples of message exchanges for association setup.

The simplest way of using the extension described in this document is given by the following message exchange.



Please note that the CHUNKS and ALL CHUNKS parameters are optional in the INIT and INIT ACK chunks.

If the server wants to receive DATA chunks in an authenticated way, the following message exchange is possible:



Please note that if the endpoint pair shared key depends on the client and the server, and is only known by the upper layer, this message exchange requires an upper layer intervention between the

processing of the COOKIE ECHO chunk and the processing of the AUTH and DATA chunk at the server side. This intervention may be realized by a COMMUNICATION-UP notification followed by the presentation of the endpoint pair shared key by the upper layer to the SCTP stack, see for example Section 11 of [RFC9260]. If this intervention is not possible due to limitations of the API (for example, the socket API), the server might discard the AUTH and DATA chunk, making a retransmission of the DATA chunk necessary. If the same endpoint pair shared key is used for multiple endpoints and does not depend on the client, this intervention might not be necessary.

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 observe the HMAC algorithms used for sending and receiving of AUTH chunks.

Please note that this section is informational only.

A socket API implementation based on [RFC6458] is, by means of the existing SCTP_AUTHENTICATION_EVENT event, extended to provide the event notification whenever a new HMAC algorithm is used in a received AUTH chunk.

Two new IPPROTO_SCTP-level socket options with the name SCTP_EXPOSE_HMAC_IDENT_CHANGES and SCTP_SEND_HMAC_IDENT are defined as described below. The first socket option enables the monitoring of HMAC algorithms used in received AUTH chunks via the SCTP_AUTHENTICATION_EVENT event. The second socket option is used to query the HMAC algorithm used for sending AUTH chunks.

Support for the SCTP_SEND_HMAC_IDENT and SCTP_EXPOSE_HMAC_IDENT_CHANGES socket options also needs to be added to the function sctp_opt_info().

8.1. Extending the SCTP_AUTHENTICATION_EVENT event

Section 6.1.8 of [RFC6458] defines the SCTP_AUTHENTICATION_EVENT event, which uses the following structure:

```
struct sctp_authkey_event {
    uint16_t auth_type;
    uint16_t auth_flags;
    uint32_t auth_length;
    uint16_t auth_keynumber;
    uint32_t auth_indication;
    sctp_assoc_t auth_assoc_id;
};
```

This document updates this structure to:

```
struct sctp_authkey_event {
    uint16_t auth_type;
    uint16_t auth_flags;
    uint32_t auth_length;
    uint16_t auth_identifier; /* formerly auth_keynumber */
    uint16_t auth_reserved; /* Avoid hole in structure */
    uint32_t auth_indication;
    sctp_assoc_t auth_assoc_id;
};
```

It renames auth_keynumber to auth_identifier. auth_identifier just replaces auth_keynumber in the context of [RFC6458]. In addition to that, the SCTP_AUTHENTICATION_EVENT event is extended to also indicate when a new HMAC Identifier is received and this reporting is explicitly enabled as described in Section 8.2. In this case auth_indication is SCTP_AUTH_NEW_HMAC and the new HMAC identifier is reported in auth_identifier.

8.2. Expose HMAC Identifier Usage (SCTP_EXPOSE_HMAC_IDENT_CHANGES)

This option allows the application to enable or disable the reception of SCTP_AUTHENTICATION_EVENT events when a new HMAC Identifier has been received in an AUTH chunk (see Section 8.1). This read/write socket option uses the level IPPROTO_SCTP and the name SCTP_EXPOSE_HMAC_IDENT_CHANGES. By default these events are not reported to provide backwards compatibility.

The following structure is used to enable or disable the reporting of newly received HMAC Identifiers in AUTH chunks:

```
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, the application may fill in an association identifier or SCTP_{FUTURE|CURRENT|ALL}_ASSOC.

assoc_value: Newly received HMAC Identifiers are reported if, and only if, this parameter is non-zero.

8.3. Get the HMAC Identifier being Sent (SCTP_SEND_HMAC_IDENT)

During the SCTP association establishment a HMAC Identifier is selected that is used by an SCTP endpoint when sending AUTH chunks. An application can access the result of this selection by using this read-only socket option, which uses the level IPPROTO_SCTP and the name SCTP_SEND_HMAC_IDENT.

The following structure is used to access HMAC Identifier used for sending AUTH chunks:

```
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, the application fills in an association identifier. It is an error to use SCTP_{FUTURE|CURRENT|ALL}_ASSOC in assoc_id.

assoc_value: This parameter contains the HMAC Identifier used for sending AUTH chunks.

9. IANA Considerations

[NOTE to RFC-Editor: "RFCXXXX" is to be replaced by the RFC number you assign this document.]

[NOTE to RFC-Editor: The requested values for the cause code are tentative and to be confirmed by IANA.]

IANA is requested to perform the following updates to SCTP-parameters (http://www.iana.org/assignments/sctp-parameters):

* In the Chunk Types Registry replace in the entry for the AUTH Chunk Type the reference to [RFC4895] by a reference to RFCXXXX.

* In the Parameter Types Registry replace in the entry for the Random, the Chunk List, and the Requested HMAC Algorithm Parameter Chunk Parameter Type the reference to [RFC4895] by a reference to RFCXXXX.

* In the Error Cause Codes Registry replace in the entry for the Unsupported HMAC Identifier Cause Code the reference to [RFC4895] by a reference to RFCXXXX. Additionally, mark the assignment as deprecated.

* In the Hashed Message Authentication Code (HMAC) Identifiers Registry replace in the Reference section the reference to [RFC4895] by a reference to RFCXXXX.

  Replace each reference to [RFC4895] by a reference to RFCXXXX for all currently registered Message Digest Algorithms.

A new chunk parameter type has to be assigned by IANA. This requires an additional line in the "Chunk Parameter Types" registry for SCTP:

ID	Chunk Parameter Type	Reference
0x8006 (suggested)	All Chunks	[RFCXXXX]

Table 5: New Entry in Chunk Paramter Types Registry

A new error cause code has to be assigned by IANA. This requires an additional line in the "Error Cause Codes" registry for SCTP:

Value	Cause Code	Reference
0x0100 (suggested)	RANDOM Collision	[RFCXXXX]

Table 6: New Entry in Error Cause Codes Registry

A new HMAC Identifier has to be assigned by IANA. This requires an additional line in the "Hashed Message Authentication Code (HMAC) Identifiers" registry for SCTP:

HMAC Identifier	Message Digest Algorithm	Reference
4 (suggested)	SHA-256 with directional keys	[RFCXXXX]

Table 7: New Entry in HMAC Identifier Registry

10. Security Considerations

Without using endpoint pair shared keys, this extension only protects against modification or injection of authenticated chunks by attackers who did not capture the initial handshake setting up the SCTP association.

If an endpoint pair shared key is used, even a true man in the middle cannot inject chunks, which are required to be authenticated, even if he intercepts the initial message exchange. The endpoint also knows that it is accepting authenticated chunks from a peer who knows the endpoint pair shared key.

The establishment of endpoint pair shared keys is out of the scope of this document. Other mechanisms can be used, like using TLS or manual configuration.

When an endpoint accepts COOKIE ECHO chunks only in an authenticated way the restart procedure does not work. Neither an attacker nor a restarted endpoint not knowing the association shared key can perform an restart. However, if the association shared key is known, it is possible to restart the association.

Because SCTP already has a built-in mechanism that handles the reception of duplicated chunks, the presented solution makes use of this functionality and does not provide a method which improves the protection against replay attacks. Of course, this only works within each SCTP association. Therefore, a separate association shared key is used for each SCTP association to handle replay attacks covering multiple SCTP associations.

Each endpoint presenting a list of more than one element in the HMAC ALGO parameter must be prepared for the peer using the weakest algorithm listed.

When an endpoint pair uses non-NULL endpoint pair shared keys and one of the endpoints still accepts a NULL key, an attacker who captured the initial handshake can still inject or modify authenticated chunks by using the NULL key.

11. Acknowledgments

The authors wish to thank Eric Rescorla for being a coauthor of [RFC4895], which is the basis of this document.

The authors wish to thank David Black, Sascha Grau, Russ Housley, Ivan Arias Rodriguez, Irene R端ngeler, Nagesh Shamnur, and Magnus Westerlund for their invaluable comments on [RFC4895].

Finally, the authors wish to thank Timo V旦lker and Magnus Westerlund for his invaluable comments on this document.

12. References

12.1. Normative References

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/info/rfc2104>.

[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>.

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

[RFC5926] Lebovitz, G. and E. Rescorla, "Cryptographic Algorithms for the TCP Authentication Option (TCP-AO)", RFC 5926, DOI 10.17487/RFC5926, June 2010, <https://www.rfc-editor.org/info/rfc5926>.

[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>.

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

[NIST_FIPS_180_4] Dang, Q. H. and NIST, "Secure Hash Standard", NIST Federal Information Processing Standards Publications 180-4, DOI 10.6028/NIST.FIPS.180-4, July 2015, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>.

12.2. Informative References

[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>.

[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>.

Authors' Addresses

Michael T端xen
M端nster Univ. of Applied Sciences
Stegerwaldstr. 39
48565 Steinfurt
Germany
Email: tuexen@fh-muenster.de

Randall R. Stewart
15214 Pendio Drive
Bella Collina, FL 34756
United States of America
Email: randall@lakerest.net

Peter Lei
Netflix, Inc.
8735 West Higgins Road
Suite 300
Chicago, IL 60631
United States of America
Email: peterlei@netflix.com

Hannes Tschofenig
Email: hannes.tschofenig@gmx.net