

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 30, 2007

M. Tuexen
Muenster Univ. of Applied Sciences
R. Stewart
P. Lei
Cisco Systems, Inc.
E. Rescorla
RTFM, Inc.
February 26, 2007

**Authenticated Chunks for Stream Control Transmission Protocol (SCTP)
draft-ietf-tsvwg-sctp-auth-08.txt**

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with [Section 6 of BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/1id-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on August 30, 2007.

Copyright Notice

Copyright (C) The IETF Trust (2007).

Abstract

This document describes a new chunk type, several parameters and procedures for 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.

Table of Contents

1.	Introduction	3
2.	Conventions	3
3.	New Parameter Types	4
3.1.	Random Parameter (RANDOM)	4
3.2.	Chunk List Parameter (CHUNKS)	5
3.3.	Requested HMAC Algorithm Parameter (HMAC-ALGO)	6
4.	New Error Cause	7
4.1.	Unsupported HMAC Identifier error cause	7
5.	New Chunk Type	8
5.1.	Authentication Chunk (AUTH)	8
6.	Procedures	10
6.1.	Establishment of an association shared key	10
6.2.	Sending authenticated chunks	11
6.3.	Receiving authenticated chunks	12
7.	Examples	14
8.	IANA Considerations	14
8.1.	A New Chunk Type	15
8.2.	Three New Parameter Types	15
8.3.	A New Error Cause	15
8.4.	A New Table For HMAC Identifiers	16
9.	Security Considerations	16
10.	Acknowledgments	17
11.	Normative References	17
	Authors' Addresses	18
	Intellectual Property and Copyright Statements	20

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.

Using TLS as defined in [RFC3436](#) [6] does not help here because it only secures SCTP user data.

Therefore an SCTP extension is presented which provides a mechanism for deriving shared keys for each association. These association shared keys are derived from endpoint pair shared keys, which are configured and might be empty, and data which 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 puts the result of an HMAC computation before the data covered by that computation. Putting it at the end of the packet would have required putting 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 SCTP extension for Dynamic Address Reconfiguration (ADD-IP) requires the usage of the extension described in this document. The SCTP Partial Reliability Extension (PR-SCTP) can be used in conjunction with the extension described in this document.

2. Conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL", when they appear in this document, are to be interpreted as described in [RFC2119](#) [3].

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)

Table 1

It should be noted that the parameter format requires the receiver to ignore the parameter and continue processing if it is not understood. This is accomplished as described in RFC2960 [5] 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 an arbitrary length random number.

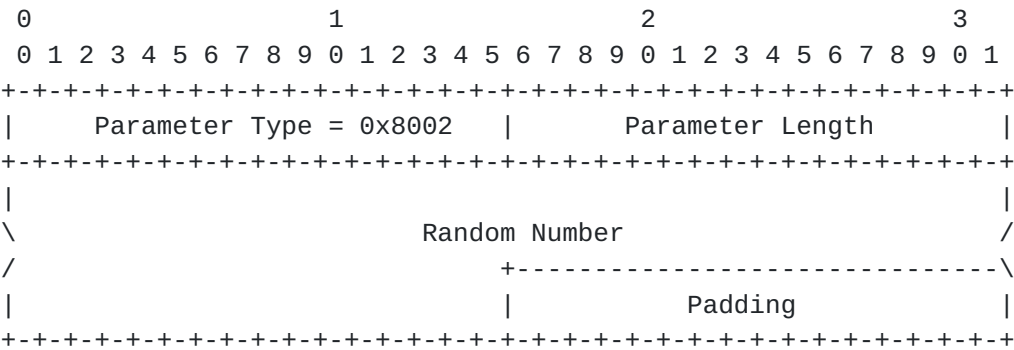


Figure 1

- 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 sent authenticated by the peer.

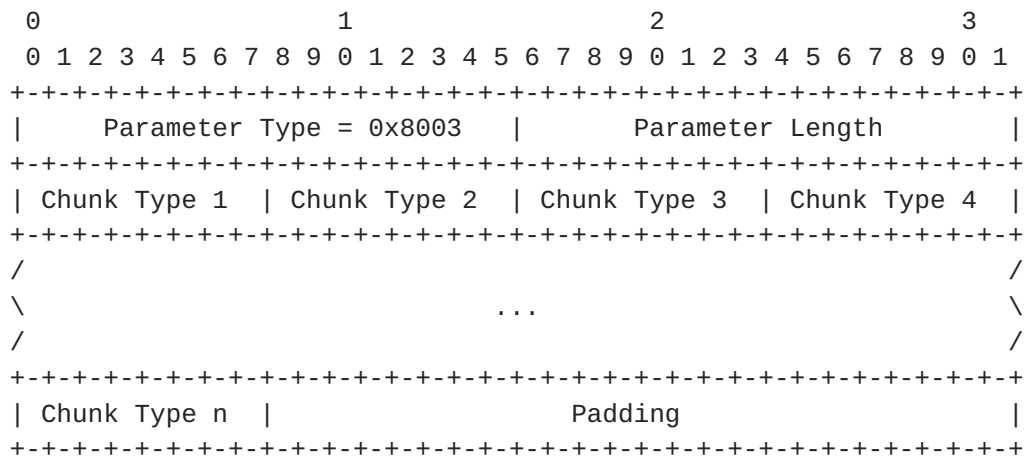


Figure 2

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.

The 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.

3.3. Requested HMAC Algorithm Parameter (HMAC-ALGO)

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

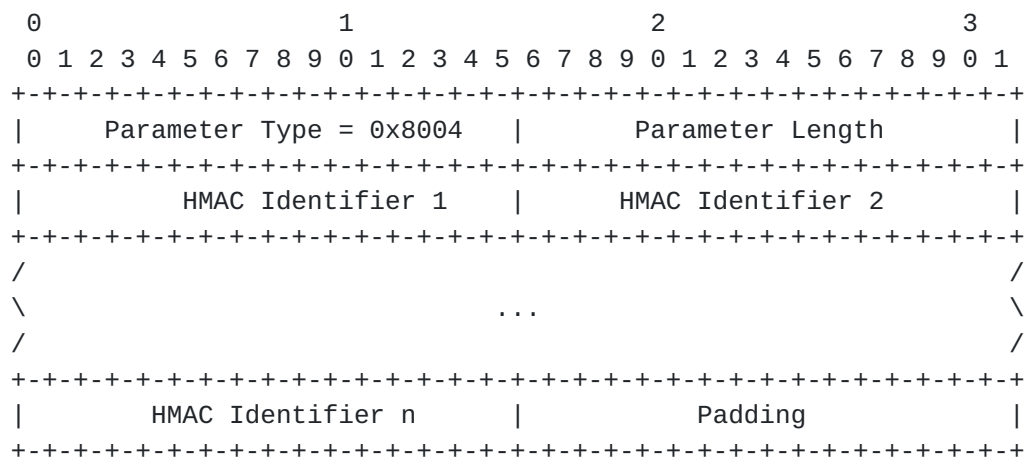


Figure 3

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.

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.

[illegible]

Figure 4

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 [RFC2960](#) [5] [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.

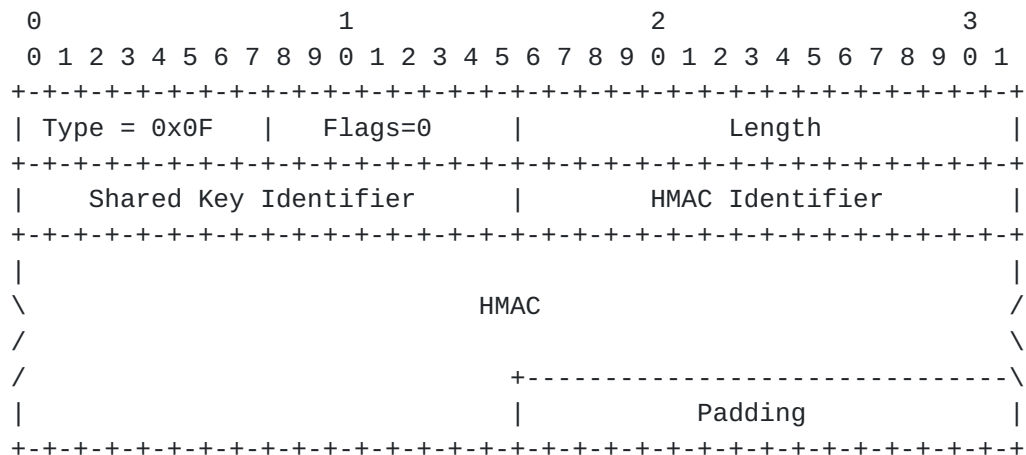


Figure 5

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 hold 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 which 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. Establishment of an association shared key

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](#) [7]. If the random number is not 32 byte long the association MUST be aborted. The ABORT chunk SHOULD contain the error cause 'Protocol Violation'. In case of INIT collision, the rules governing the handling of this random number follow the same pattern as those for the Verification Tag, as explained in [section 5.2.4 of RFC2960](#) [5]. Therefore each endpoint knows its own random number and the peer's random number after the association has been established.

An SCTP endpoint has a list of chunks it only accepts if they are received in an authenticated way. This list is included in the INIT and INIT-ACK and MAY be omitted if it is empty. Since this list does not change during the lifetime of there is no problem in case of 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 a HMAC-ALGO parameter SHOULD use the first listed algorithm it supports. The HMAC algorithm based on SHA-1 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.

Both endpoints of an association MAY have endpoint pair shared keys which 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 key is computed. If there is no endpoint pair shared key only one association shared key is computed by using an empty byte vector as the endpoint pair shared key.

The RANDOM parameter, the CHUNKS parameter and the HMAC-ALGO parameter sent by each endpoint are concatenated as byte vectors. 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. Parameters which were not sent are simply omitted from the concatenation process. The resulting two vectors are called the two key vectors.

From the endpoint pair shared keys and the key vectors the

association shared keys are computed. This is performed 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 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 key.

6.2. Sending authenticated chunks

Endpoints MUST send all requested chunks authenticated where this has been requested by the peer. The other chunks MAY be sent authenticated or not. If endpoint pair shared keys are used, 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 0 with an empty endpoint pair shared key. If there are multiple endpoint shared keys the sender selects one and uses the corresponding Shared Key Identifier.

The sender MUST calculate the MAC as described in [RFC2104](#) [2] using the hash function H as described by the MAC Identifier and the shared association 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 shown in Figure 6) followed by all chunks that are placed after the AUTH chunk in the SCTP packet.

```

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Type = 0x0F | Flags=0 | Chunk Length |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Shared Key Identifier | HMAC Identifier |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|
\                                /
/                                +-----\
|                                | Padding |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```


Figure 6

Please note that all fields are in network byte order and that the field which will contain the complete HMAC is filled with zeroes. The length of the field shown as 0 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 which 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 chunks after it MUST be discarded and an ERROR chunk SHOULD be sent with the error cause defined in [Section 4.1](#).

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 based on Figure 6. If the result of the calculation is the same as given in the HMAC field, all 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 an 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 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 chunks after the COOKIE-ECHO MUST be processed. If the receiver has a 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 chunks after it.

If the receiver does not find a STCB for a packet containing an AUTH chunk as the first chunk and not 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 end-point most likely does not know the association shared key of the old association to be restarted. However, if the restarting end-point does know the old association shared key he can send successfully 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 end-points 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 client 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 end-point 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.

```
----- INIT[RANDOM; CHUNKS; HMAC-ALGO] ----->
<----- INIT-ACK[RANDOM; CHUNKS; HMAC-ALGO] -----
----- COOKIE-ECHO ----->
<----- COOKIE-ACK -----
```

Please note that the CHUNKS parameter is optional in the INIT and INIT-ACK.

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

```
----- INIT[RANDOM; CHUNKS; HMAC-ALGO] ----->
<----- INIT-ACK[RANDOM; CHUNKS; HMAC-ALGO] -----
----- COOKIE-ECHO; AUTH; DATA ----->
<----- COOKIE-ACK; SACK -----
```

Please note that if the endpoint pair shared key depends on the client and the server and that it 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 10 of RFC2960](#) [5]. 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. IANA Considerations

[NOTE to RFC-Editor:

"RFCXXXX" is to be replaced by the RFC number you assign this document.

]

This document (RFCXXX) is the reference for all registrations described in this section. All registrations need to be listed in the document available at sctp-parameters [9]. The suggested changes are described below.

8.1. A New Chunk Type

A chunk type for the AUTH chunk has to be assigned by IANA. It is suggested to use the value given in Table 4. This requires an additional line in the "CHUNK TYPES" table of sctp-parameters [9]:

CHUNK TYPES

ID Value	Chunk Type	Reference
-----	-----	-----
15	Authentication Chunk (AUTH)	[RFCXXXX]

8.2. Three New Parameter Types

Parameter types have to be assigned for the RANDOM, CHUNKS, and HMAC-ALGO parameter by IANA. It is suggested to use the values given in Table 1. This requires two modifications of the "CHUNK PARAMETER TPYES" tables in sctp-parameters [9]: The first change is the addition of three new lines to the "INIT Chunk Parameter Types" table:

Chunk Parameter Type	Value
-----	-----
Random	32770 (0x8002)
Chunk List	32771 (0x8003)
Requested HMAC Algorithm Parameter	32772 (0x8004)

The second required change is the addition of the same three lines to the to the "INIT ACK Chunk Parameter Types" table.

8.3. A New Error Cause

An error cause for the Unsupported HMAC Identifier error cause has to be assigned. It is suggested to use the value given in Table 3. This requires an additional line of the "CAUSE CODES" table in sctp-parameters [9]:

VALUE	CAUSE CODE	REFERENCE
-----	-----	-----
261 (0x0105)	Unsupported HMAC Identifier	RFCXXXX

8.4. A New Table For HMAC Identifiers

HMAC Identifiers have to be maintained by IANA. Four initial values should be assigned by IANA as described in Table 2. This requires a new table "HMAC IDENTIFIERS" in sctp-parameters [9]:

HMAC Identifier	Message Digest Algorithm	REFERENCE
-----	-----	-----
0	Reserved	RFCXXXX
1	SHA-1	RFCXXXX
2	Reserved	RFCXXXX
3	SHA-256	RFCXXXX

For registering at IANA a new HMAC Identifier in this table a request has to be made to assign such a number. This number must be unique and a message digest algorithm usable with the HMAC defined in [RFC2104](#) [2] MUST be specified. The "Specification Required" policy of [RFC2434](#) [4] MUST be applied.

9. Security Considerations

Without using endpoint 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 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 end-point 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 has already a mechanism built-in that handles the

reception of duplicated chunks, the presented solution makes use of this functionality and does not provide a method to avoid replay attacks by itself. Of course, this only works within each SCTP association. Therefore a separate 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 that the peer uses 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.

10. Acknowledgments

The authors wish to thank David Black, Sascha Grau, Russ Housley, Ivan Arias Rodriguez, Irene Ruengeler, and Magnus Westerlund for their invaluable comments.

11. Normative References

- [1] Rivest, R., "The MD5 Message-Digest Algorithm", [RFC 1321](#), April 1992.
- [2] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", [RFC 2104](#), February 1997.
- [3] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [4] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", [BCP 26](#), [RFC 2434](#), October 1998.
- [5] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and V. Paxson, "Stream Control Transmission Protocol", [RFC 2960](#), October 2000.
- [6] Jungmaier, A., Rescorla, E., and M. Tuexen, "Transport Layer Security over Stream Control Transmission Protocol", [RFC 3436](#), December 2002.
- [7] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", [BCP 106](#), [RFC 4086](#), June 2005.

- [8] National Institute of Standards and Technology, "Secure Hash Standard", FIPS PUB 180-2, August 2002,
<<http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf>>.
- [9] <<http://www.iana.org/assignments/sctp-parameters>>

Authors' Addresses

Michael Tuexen
Muenster Univ. of Applied Sciences
Stegerwaldstr. 39
48565 Steinfurt
Germany

Email: tuexen@fh-muenster.de

Randall R. Stewart
Cisco Systems, Inc.
4875 Forest Drive
Suite 200
Columbia, SC 29206
USA

Email: rrs@cisco.com

Peter Lei
Cisco Systems, Inc.
8735 West Higgins Road
Suite 300
Chicago, IL 60631
USA

Phone:

Email: peterlei@cisco.com

Eric Rescorla
RTFM, Inc.
2064 Edgewood Drive
Palo Alto, CA 94303
USA

Phone: +1 650-320-8549

Email: ekr@rtfm.com

Full Copyright Statement

Copyright (C) The IETF Trust (2007).

This document is subject to the rights, licenses and restrictions contained in [BCP 78](#), and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in [BCP 78](#) and [BCP 79](#).

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).

