Workgroup: Network Working Group
Internet-Draft: draft-hohendorf-secure-sctp-33
Published: 26 September 2023
Intended Status: Experimental
Expires: 29 March 2024
Authors: C. Hohendorf
University of Duisburg-Essen
T. Dreibholz
SimulaMet
E. Unurkhaan
Mongolian University

Secure SCTP

Abstract

This document explains the reason for the integration of security functionality into SCTP, and gives a short description of S-SCTP and its services. S-SCTP is fully compatible with SCTP defined in RFC4960, it is designed to integrate cryptographic functions into SCTP.

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 29 March 2024.

Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction

SCTP is a message oriented reliable transmission protocol which works on top of the IP-based network. It provides several advantages over other transmission protocols, such as TCP and UDP over IP. One of the advantages is multistreaming -- user data transported by individual streams. When multistreaming is used, network blocking can be avoided in certain cases (e.g. network loss). Also, SCTP supports multihoming -- the endpoints support multiple IP addresses. SCTP provides unordered delivery, so that a receiver immediately delivers user data to the upper layers upon receipt. For more details, see RFC4960 [6].

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 [2] [8] when, and only when, they appear in all capitals, as shown here.

3. A brief description of S-SCTP

S-SCTP provides security functionalities in the transport layer without the need for any other security protocols (e.g. TLS or IP-sec). Normally, a data transport over SCTP can either be secured with TLS or can be protected by IPsec. As both TLS over SCTP and SCTP over IPsec have disadvantages in certain scenarios, it is preferable to integrate cryptographic functions into SCTP.

The main issues for the security solutions TLS over SCTP RFC3436 [3] and SCTP over IPSec RFC3554 [4] is scalability with the number of streams. For N secure streams, N TLS connections have to be created, and N handshakes have to be performed. If N is small, this is not a big issue, but as N grows larger, it becomes a problem because a handshake is a slow and expensive process. So, when an application performs N handshakes, the load in terms of memory use, CPU use etc. increases linearly over time. This problem could be overcome by using IPsec. However, IPsec is not so flexible and on the other hand SCTP over IPsec has to establish new security associations (SA) for newly added IP addresses in dynamic address reconfiguration scenario. In this case, the application has to configure a new SA and to negotiate a new key exchange.

4. Key terms

This part gives the definitions of the key terms, which are used in this draft:

*Secure session: This is the session, which provides the security functionalities for an established SCTP association.

*Master secret key: S-SCTP uses two kinds of secret keys. One is the secret key for the S-SCTP packet authentication, and the other is the secret key for the data encryption and decryption.

*Cipher suite: This is the suite of cryptographic algorithms, which are used for key exchange, data encryption/decryption and the packet authentication.

*Pre-enc-data: This is the collection of the data chunks, which requires encryption. The data chunks are concatenated together and create pre-enc-data. Pre-enc-data may include the padding chunk.

*Cipher suite sequence: This is the bundle of cipher suites chosen by an endpoint from the supported cipher suites.

5. Additional chunks and parameters

Several new chunks and parameters are defined in S-SCTP. This section explains those chunks and parameters. All new chunks can be bundled with other chunks. The new parameters follow the Type-Length-Value format as defined in section 3.2.1 of RFC4960.

5.1. New type chunks and definitions

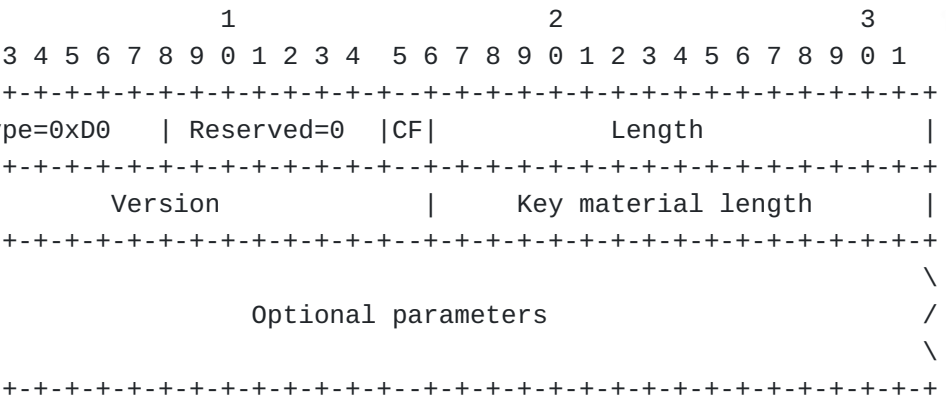
The following table shows the new chunks. All new chunks, except for the Encrypted Data (EncData) chunk, Authentication (AUTH) chunk and Padding (PADDING) chunk, are used for building the secure session and to update the master secret key. The new chunks are to be interpreted as described in Section 3.2 of RFC 4960, by the receiver.

Chunktype	Chunk name
0xD0	Secure Session Open Request Chunk
0xD1	Secure Session Certificate Chunk
0xD2	Secure Session Acknowledge Chunk
0xD3	Secure Session Server Key Chunk
0xD4	Secure Session Client Key Chunk
0xD5	Secure Session Open Complete Chunk
0xD6	Secure Session Close Chunk
0xD7	Secure Session Close Acknowledge Chunk
0xD8	Security Level Change Chunk
0xD9	Security Level Change Acknowledge Chunk
0x10	Encrypted Data Chunk
0x11	Authentication Chunk
0x12	Padding Chunk

The new parameters are defined in this section.

5.1.1. Secure Session Open request chunk (SSOpReq)

An endpoint creates the Secure Session Open Request chunk (see next table)when it wishes to establish a secure session. The chunk can be bundled with other chunks. The SSOpReq chunk can also be used to update the master secret key or cipher suite after a secure session establishment. During the association lifetime, both associated endpoints can request an update of the master secret key or cipher suite; in this case, the requesting endpoint sends the SSSOpReq chunk immediately to the other endpoint.



CF: Certificate flag: 1 bit

This flag indicates whether or not the client will send a certificate. It is set to 1 when the client sends a certificate. If a receiver receives SSOpReq chunk with CF=1 and does not receive a certificate it raises an error and terminates the secure session initialisation.

Length: 16 bits unsigned integer

The length field contains the size of the chunk in bytes, including the chunk header, version, random number length and optional parameter(s).

Version: 16 bits unsigned integer

This field indicates the S-SCTP version 1.0. The high eight bits indicate the major version, the low eight bits indicate minor version.

Key material length: 16 bits unsigned integer

This number has two meanings:

- when the handshake is made using the RSA key exchange protocol, the key material length defines the random number length, which is generated by the server and client to calculate a master secret key (see RSA parameters of the SSSerKey and SSCliKey chunks)

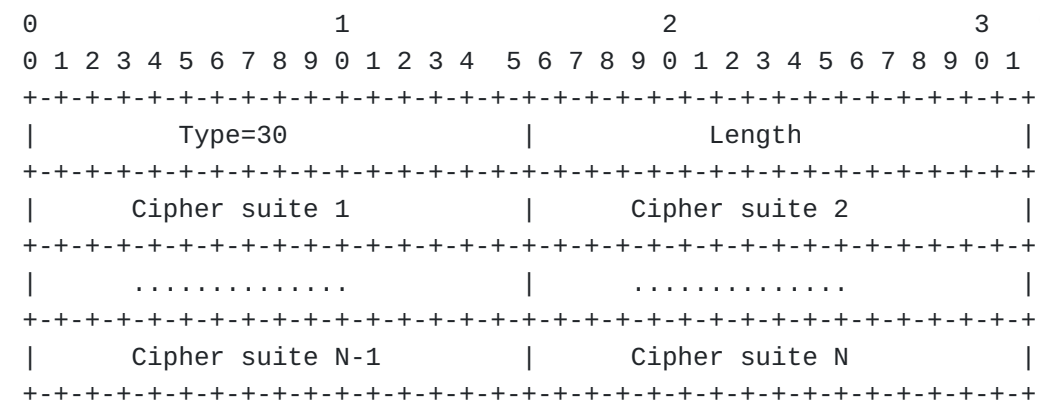
- when the handshake is made using the DH key exchange protocol, the key material length defines the DH prime number length (see DH parameters of the SSSerKey and SSCliKey chunks). For security reasons, the key material length MUST be 512 bits (default) or longer when the key exchange mechanism uses RSA, and 1024 bits (default) or longer when the key exchange mechanism uses DH. The key material length is increased in steps of 64 bits. If a user defines the key material length to be shorter than the default value, S-SCTP automatically sets it to the default.

Parameter(S):

SSOpReq chunk includes the cipher suite and compression method parameters, which are described below:

Cipher suite parameter:

This parameter contains the cipher suites, which are chosen from all supported cipher suites by the client. One of them is used for the secure session. The user can choose certain cipher suites from the cipher suites supported by the client.



Cipher suite: 16 bits unsigned integer:

This field indicates a cipher suite, which is supported by the client. The next table includes cipher suites supported in S-SCTP. Additional cipher suites can be specified.

Value	Cipher suite	Key exchange	Encryption	Hash
1	RSA_with_DES_CBC_MD5	RSA	DES_CBC	MD5
2	RSA_with_DES_CBC_SHA-1	RSA	DES_CBC	SHA-1
3	RSA_with_3DES_CBC_MD5	RSA	3DES_CBC	MD5
4	RSA_with_3DES_CBC_SHA-1	RSA	3DES_CBC	SHA-1
5	RSA_with_AES-128_CBC_MD5	RSA	AES-128	MD5
6	RSA_with_AES-128_CBC_SHA-1	RSA	AES-128	SHA-1
7	DH_with_DES_CBC_MD5	DH	DES_CBC	MD5
8	DH_with_DES_CBC_SHA-1	DH	DES_CBC	SHA-1
9	DH_with_3DES_CBC_MD5	DH	3DES_CBC	MD5
10	DH_with_3DES_CBC_SHA-1	DH	3DES_CBC	SHA-1
11	DH_with_AES-128_CBC_MD5	DH	AES-128	MD5
12	DH_with_AES-128_CBC_SHA-1	DH	AES-128	SHA-1
13	RSA_with_NULL_MD5	RSA	NULL	MD5
14	RSA_with_NULL_SHA-1	RSA	NULL	SHA-1
15	DH_with_NULL_MD5	DH	NULL	MD5
16	DH_with_NULL_SHA-1	DH	NULL	SHA-1
17	RSA_with_AES-192_CBC_MD5	RSA	AES-192	MD5
18	RSA_with_AES-192_CBC_SHA-1	RSA	AES-192	SHA-1
19	RSA_with_AES-256_CBC_MD5	RSA	AES-256	MD5
20	RSA_with_AES-256_CBC_SHA-1	RSA	AES-256	SHA-1
21	DH_with_AES-192_CBC_MD5	DH	AES-192	MD5
22	DH_with_AES-192_CBC_SHA-1	DH	AES-192	SHA-1
23	DH_with_AES-256_CBC_MD5	DH	AES-256	MD5
24	DH_with_AES-256_CBC_SHA-1	DH	AES-256	SHA-1

The hash algorithms, defined in cipher suites, are used only for the S-SCTP packet authentication, and not for the signature of the handshake messages. An S-SCTP implementation MUST at least support

the default cipher suite, DH_with_3DES_CBC_SHA-1 (value=0). If the SSOpReq chunk does not contain a cipher suite parameter, then:

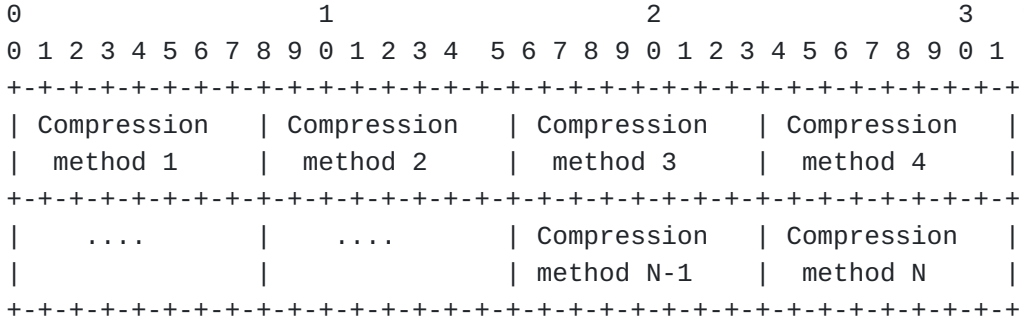
a.) S-SCTP will use the default, or:

b.) S-SCTP will use the old cipher suite.

The case "a" will be used at the beginning of the secure session. The case "b" will be used when the SSOpReq chunk is created after a secure session establishment. The signature schemes are derived from both the client and server certificates, and may be different.

Compression method parameter

This parameter contains compression methods, which are used for data compression. The data compression uses lossless compression methods. The user chooses several compression methods and sends it to the receiver. The receiver chooses one compression method.



Compression method: 8 bits unsigned char

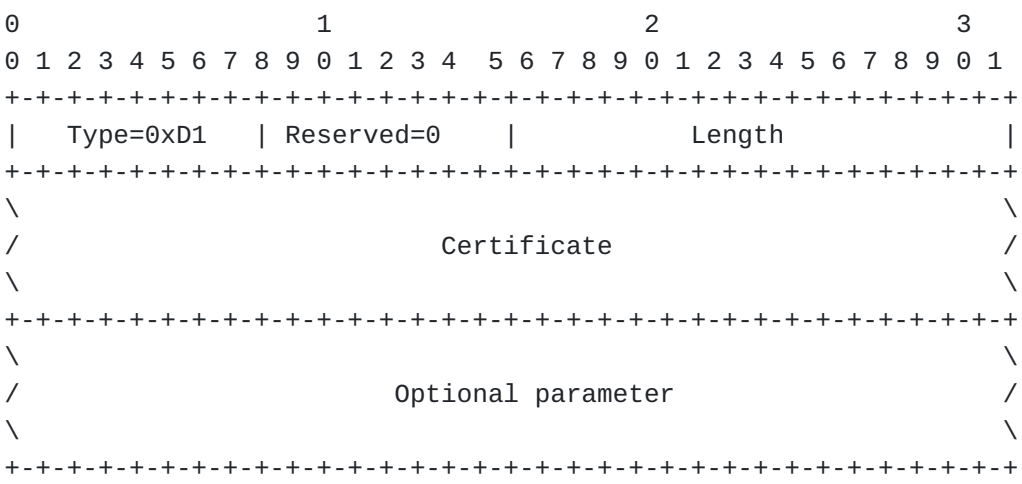
This field indicates a compression method, which is supported by the client. The next table includes compression methods supported in S-SCTP. Additional compression methods can be specified.

Value	Compression method
1	Huffman Code
2	Lempel-Ziv Code

The secure session uses null compression when the SSOpReq chunk contains no compression parameters.

5.1.2. Secure Session Certificate chunk: (SSCert)

This chunk can be sent by both endpoints. The certificate helps to authenticate the endpoint, that establishes a S-SCTP session. This chunk contains only one parameter, the certificate parameter. The SSCert chunk is optional. For security reasons, both endpoints SHOULD use a certificate to authenticate each other.



Length: 16 bits unsigned integer

The length field contains the size of the chunk in bytes, including the chunk header and parameter.

Certificate: Variable length

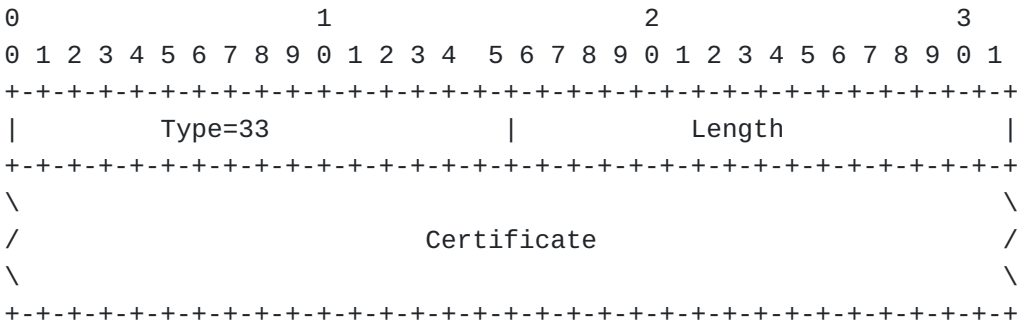
The certificate field contains the certificate of the endpoint. S-SCTP uses the X.509v3 certificate which is defined in RFC5280 [7].

Optional parameter

SSCert chunk has only one optional parameter.

Certificate parameter

The SSCert chunk uses the certificate parameter for additional certificates, when the endpoint has two or more certificates.



Certificate: Variable length

The endpoint can send two or more certificates. In this case the certificate field contains the endpoint's additional certificate. S-SCTP uses the X.509v3 certificate, which is defined in RFC5280 [7].

5.1.3. Secure Session Open Acknowledge chunk (SSOpReq_Ack)

The Secure Session Open Acknowledge chunk is sent by the server to tell the client that the secure session open request is accepted.

CF: Certificate flag: 1 bit

This flag indicates whether or not the server has a certificate. This flag is set to 1 when the server has a certificate, else it is zero.

Length: 16 bits unsigned integer

The chunk length is 8 bytes, including the chunk header, version and cipher suite field.

Version: 16 bits unsigned integer

This field indicates the S-SCTP version 1.0. The high eight bits indicate the major version, the low eight bits indicate the minor version.

Cipher suite: 16 bits unsigned integer

This field indicates the cipher suite, which is used for a secure session. The cipher suite includes necessary information for the key derivation, message encoding and MAC computation. The server uses the following rules to choose the cipher suite:

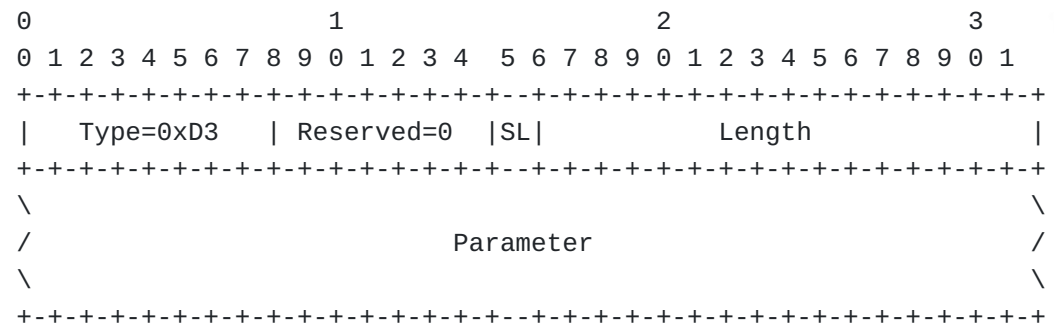
- *Client and Server do not have a certificate: Use DH key exchange.
- *Client or Server has a certificate: Use DH key exchange.
- *Client and Server have a RSA certificate: Use RSA key exchange.
- *Client and Server have a DSS certificate: Use DH key exchange.

Compression method: 16 bits unsigned char

This field indicates the compression method, which is used for data compression in the secure session.

5.1.4. Secure Session Server Key chunk (SSSerKey)

This chunk includes the parameter which is used for the key exchange algorithm. The Server Key Exchange chunk consists of the chunk header and one parameter. The parameter type depends on the key exchange algorithm.



Security level (SL): 2 bits

This 2-bit value indicates a server's security level of the reserved flags.

Length: 16 bit unsigned integer

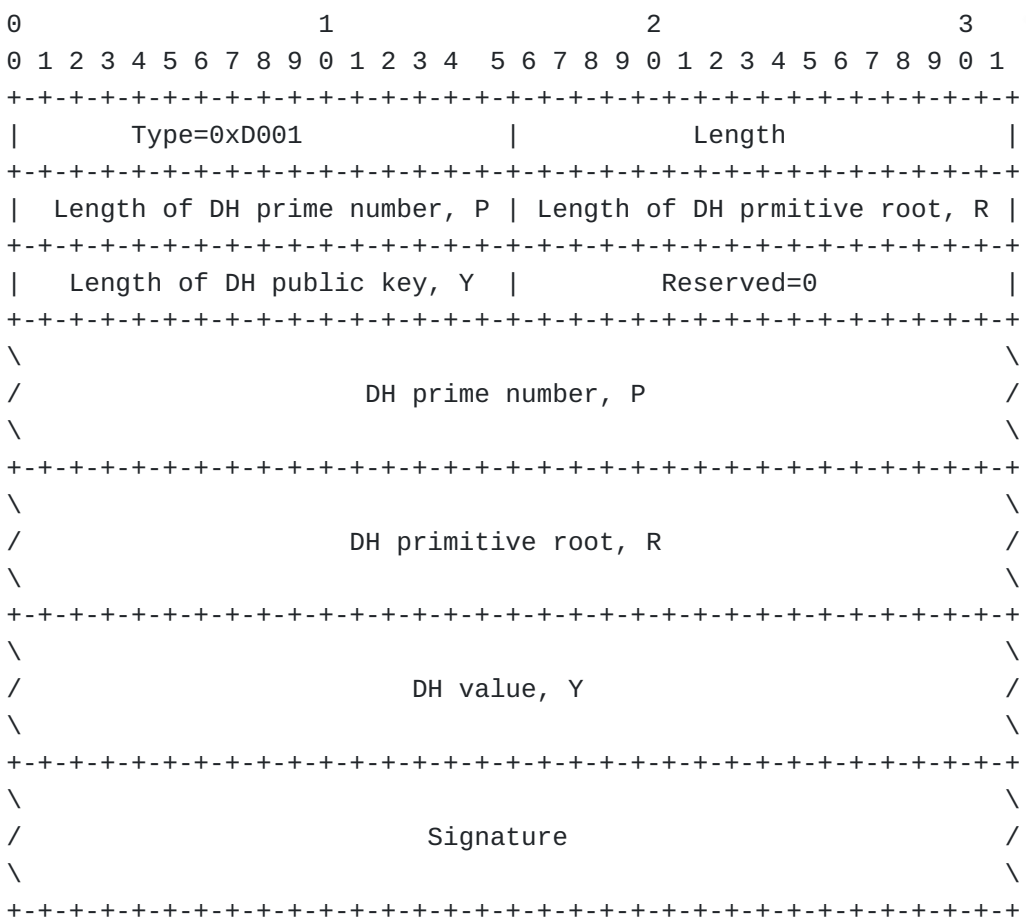
The length field contains the size of the chunk in bytes, including the chunk header and parameter.

Parameters:

The following two parameters define the key exchange protocols. Their parameter formats are shown in the next two tables. When a user specifies a new cipher suite with a new key exchange algorithm, then they must define a new parameter.

Diffie-Hellman parameter

The SSSerKey chunk uses this parameter when the handshake is done via the DH key exchange algorithm.



Length: 16 bit unsigned integer

The length field contains the size of the parameter in bytes, including the parameter header, length of DH prime number, length of DH primitive root, length of DH public key, reserved, DH prime number, DH primitive root, DH public key and signature.

Length of DH prime number, P: 16 bits unsigned integer

This field contains the size of the DH prime number.

Length of DH primitive root, Q: 16 bits unsigned integer

This field contains the size of the DH primitive root. The size of the prime number is equal R, where R is a random number defined in the SSOpReq chunk.

Length of DH value, Y: 16 bits unsigned integer

This field contains the size of the DH public key.

DH value, P: Variable length

This is the prime number of the DH key exchange protocol.

DH value, Q: Variable length

This is the primitive root of the prime number P.

DH value, Y: Variable length

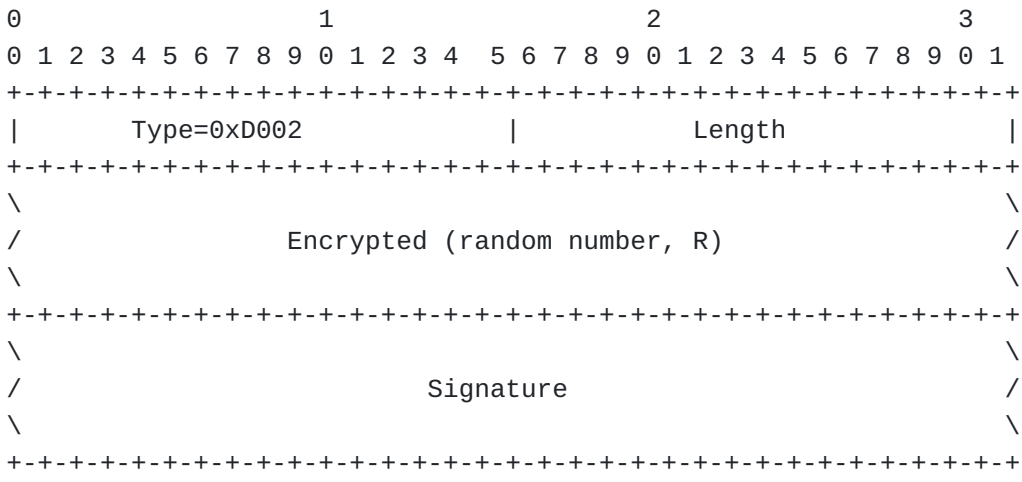
This is the public key of the DH key exchange protocol.

Signature: Variable length

The field contains the signature which is derived from the chunk header and the whole parameter except the signature field. The signature computation uses the signature algorithm which is defined in the server certificate. If the server does not have a certificate, this field does not exist in the parameter.

RSA parameter

The SSSerKey chunk uses this parameter when the handshake uses the RSA key exchange algorithm.



Length: 16 bits unsigned integer

The length field contains the size of the parameter in bytes, including the parameter header, the encrypted random number and the signature.

Encrypted (Random number, R): Variable length

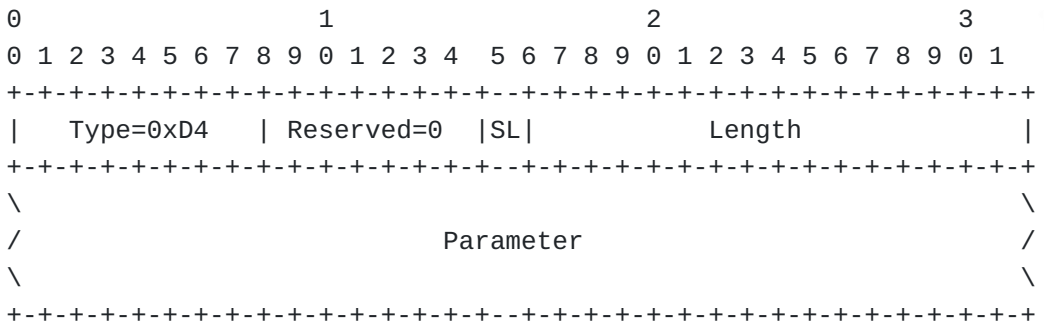
The random number is used to generate the secret keys for user data encryption and authentication. The random number encryption uses the client public key.

Signature: Variable length

The field contains the signature, which is derived from the chunk header and the whole parameter except the signature field. The signature computation uses the signature algorithm which is defined in the server certificate.

5.1.5. Secure Session Client Key chunk (SSCliKey)

This chunk includes the parameters which are used for the key exchange algorithm. The Secure Session Client Key Exchange chunk consists of the chunk header and one parameter. The parameter format depends on the key exchange algorithm.



Security level (SL): 2 bits

This 2-bit value indicates a client's security level.

Length: 16 bit unsigned integer

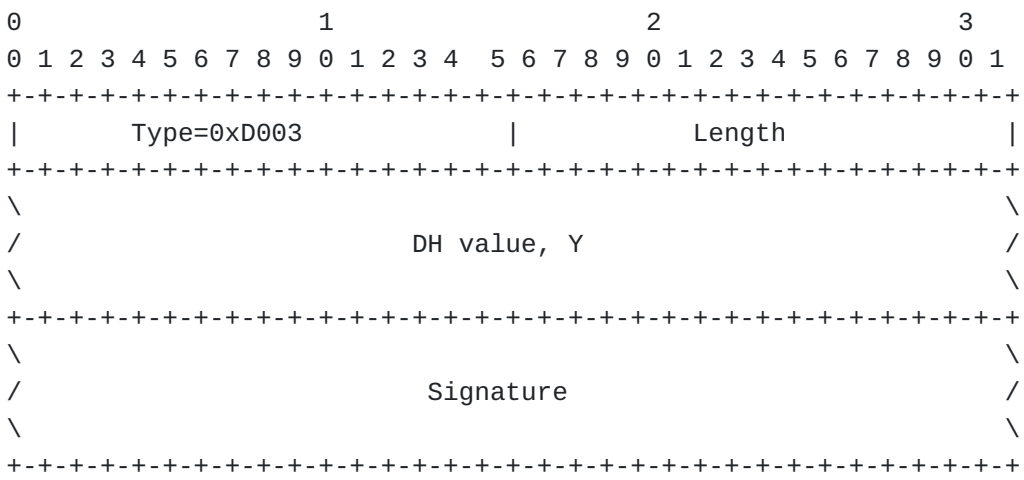
The length field contains the size of the chunk in bytes, including the chunk header and parameter.

Parameters:

Two new parameters are defined here that can appear in the SSCliKey chunk. Their parameter formats are shown in the next two tables.

Diffie-Hellman parameter

The SSCliKey chunk uses this parameter when the handshake uses the DH key exchange algorithm.



Length: 16 bit unsigned integer

The length field contains the size of the parameter in bytes, including the parameter header, the DH public key and the signature.

DH value, Y: Variable length

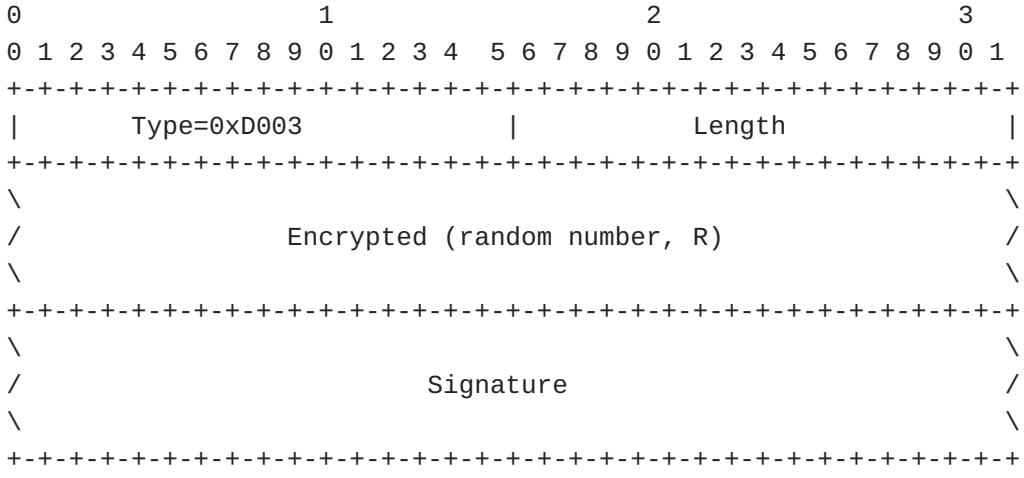
This field contains the public key of the DH key exchange protocol.

Signature: Variable length

The field contains a signature which is derived from the chunk header and the whole parameter except the signature field. The signature computation uses the signature algorithm defined in the client certificate. If the client does not have a certificate, then this field does not exist in the parameter.

RSA parameter

The SSCliKey chunk uses this parameter when the handshake uses RSA key exchange algorithm.



Length: 16 bits unsigned integer

The length field contains the size of the parameter in bytes, including the parameter header, the encrypted random number and a signature.

Encrypted (Random number): Variable length

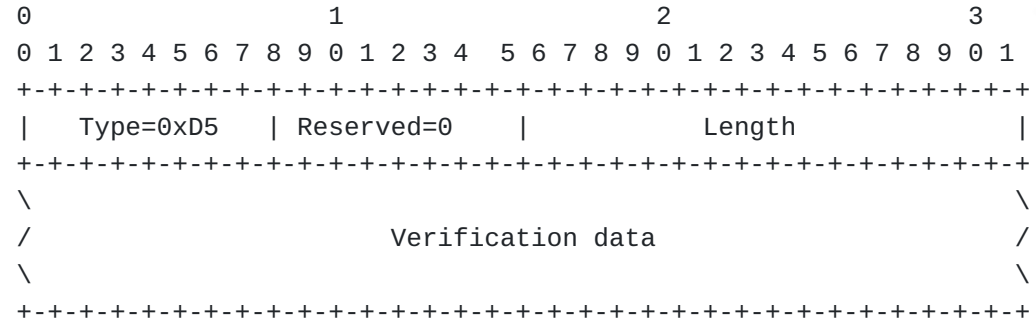
This field contains the random number, encrypted by the server's public key, which is used to generate the master secret key for encryption and authentication.

Signature: Variable length

The field contains the signature which is derived from the chunk header and the whole parameter except the signature field. The signature computation uses the signature algorithm defined in the server certificate.

5.1.6. Secure Session Open Complete chunk (SSOpCom)

This chunk is the last chunk of the handshake and it indicates the completion of the secure session establishment. After receiving this chunk the endpoint verifies the verification data which is contained in the chunk. The secure session procedure is complete when the verification is successful. Otherwise the secure session will be closed.



Length: 16 bits unsigned integer

The length field contains the size of the chunk in bytes, including the chunk header and verification data.

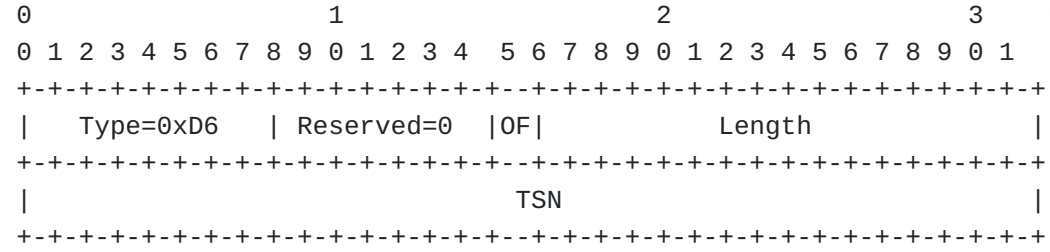
Verification data: Variable length

The verification data contains a hashed value which is an output of the HMAC function. The HMAC uses the authentication secret key, which is individually generated by the endpoints. The HMAC input contains all received secure session handshake chunks of the current endpoint. Both endpoints compute the hash value individually and

exchange it using the SSOpCom chunk. The receiver computes the hash value using the same algorithm as the sender, and compares it with the verification data. If the receiver's computed value is the same as the sender's verification data, then the receiver assures that the secure session open is successfully completed. If it is not the same, then the receiver sends an ERROR message to the sender, and immediately closes the secure session.

5.1.7. Secure Session Close chunk (SSClose)

This chunk indicates a request to close the current secure session. The sender MUST NOT send any encrypted or authenticated chunks after it has sent this chunk.



Outstanding flag (OF): 1 bit

This flag indicates that the endpoint has sent the SSClose chunk and has no outstanding secured data.

Length: 16 bits unsigned integer

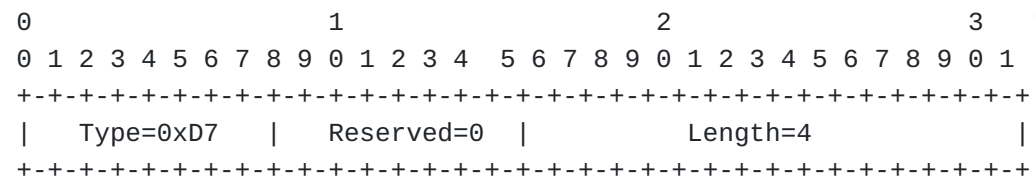
The length field contains the size of the chunk in bytes, including the chunk header and TSN.

Transmission sequence number (TSN): 16 bits unsigned integer

This is the transmission sequence number of the data chunk that was last encrypted and sent. The TSN helps the server to detect outstanding EncData chunks.

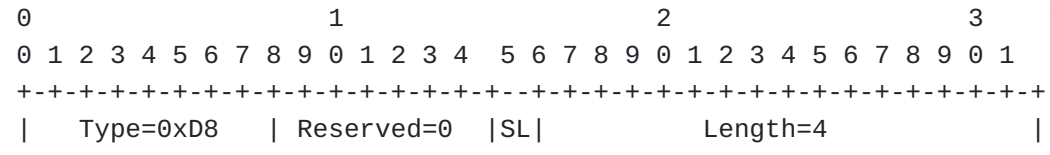
5.1.8. Secure Session Close Acknowledge chunk (SSClose_Ack)

This chunk is used to acknowledge the receipt of the secure session close chunk. When the endpoint receives the secure session close chunk, it immediately stops sending encrypted or authenticated chunks. The Secure Session Close Acknowledge chunk only consists of the chunk header.



5.1.9. Security Level Changed chunk (SecLevCHD)

This chunk is used to convey the other associated endpoint of the endpoint's new security level. The endpoint sends SecLevCHD chunk every time it selects a new security level. The endpoint uses the new selected security level when it receives the Security Level Changed Acknowledged chunk. The sender MUST NOT send a new SecLevCHD chunk when an unacknowledged SecLevCHD chunk exists.

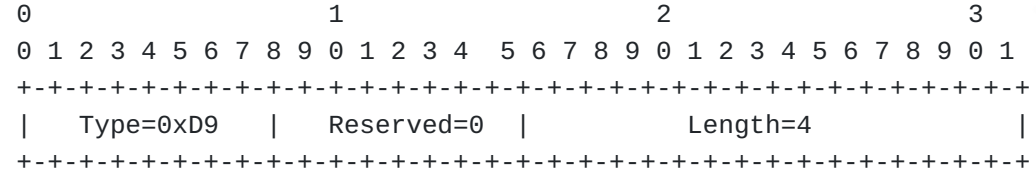


Security level (SL): 2 bits

This 2-bit value indicates the sender's new security level.

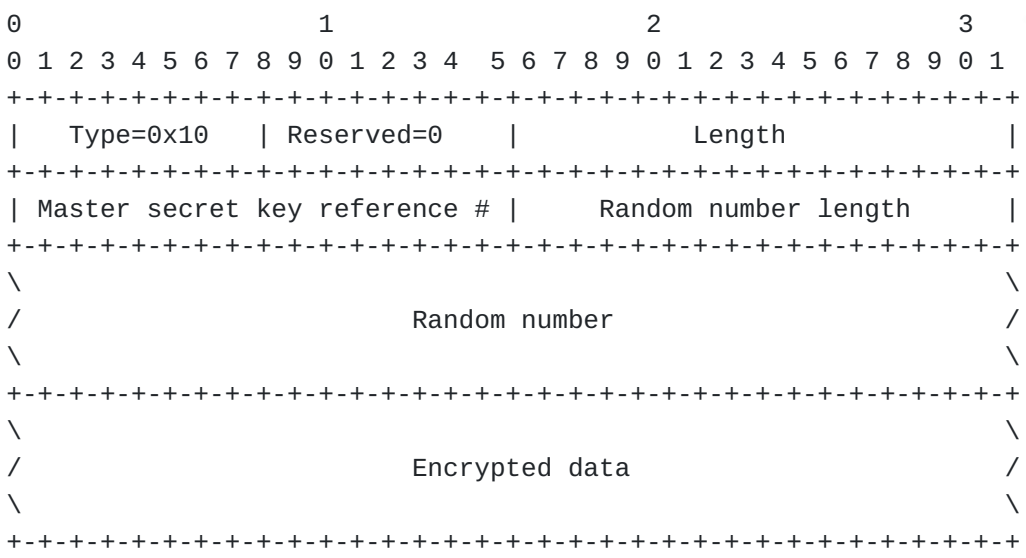
5.1.10. Security Level Changed Acknowledged chunk (SecLevCHD_Ack)

This chunk is used to acknowledge the receipt of the SecLevCHD chunk. When the endpoint receives the SecLevCHD chunk, it immediately sends the SecLevCHD_Ack chunk.



5.1.11. Encrypted Data Chunk (EncData)

Each S-SCTP packet includes at most one encrypted data chunk, and the packet could also include several (normal, unencrypted) data chunks. The encrypted data chunk may contain one or several data chunks. The EncData chunk includes a padding chunk when it is needed.



Length: 16 bits unsigned integer

The length field contains the size of the chunk in bytes,including the chunk header and encrypted data.

Master secret key reference number: 16 bits unsigned integer

The association can be updated by changing the master secret key several times during the association lifetime. The association keeps old secret keys. The number of the kept old secret keys depends on the implementation. This field helps to identify which key (old or new) has been used for encryption. That means the endpoint is able to receive messages, which were encrypted with an old key, after the update of a master secret key.

Random number length: 16 bits unsigned integer

This field contains the size of the random number which is defined below.

Random number: Variable length

This field indicates the random number which is used as initialisation vector of the cipher block chaining (CBC) mode for encryption.

Encrypted data: Variable length

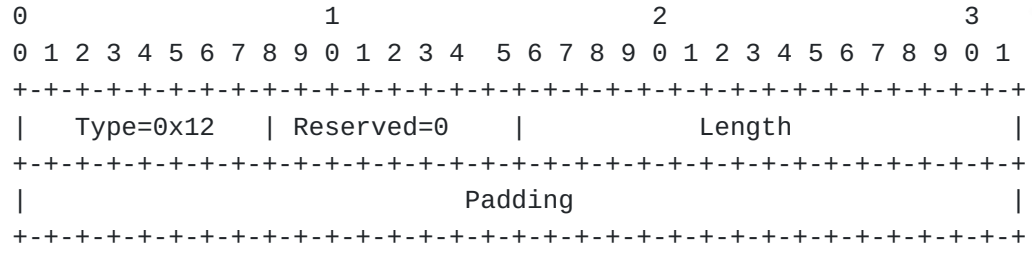
Contains a byte vector that consists of the encrypted data chunks. Before encryption, the chunk(s) MUST fulfil the following conditions. If encryption is performed using the DES or 3DES algorithm, the total size of the chunk(s) MUST be a multiple of 8 bytes. If encryption is performed using the AES algorithm, the total size of the chunk(s) MUST be a multiple of 16 bytes. If the total

size of the chunk(s) is not a multiple of 8 bytes or 16 bytes, the sender MUST add appropriate padding at the chunk's end.

5.1.12. Padding chunk (PADDING)

This padding chunk is used with an EncData chunk. The symmetric encryption algorithms use a block oriented encryption of the user data. For example DES uses 64 bit blocks, and AES uses 128 bit blocks. Before encryption, the user data which has to be encrypted has to be formatted according to the required block size. If the last block is not completely full, padding has to be added. If less than 4 bytes padding are required, the block is filled up will all zeros. The receiver can calculate the number of padding bytes based on the length field of the original data chunks. If 4 bytes or more are required, a padding chunk of appropriate length is added.

The algorithms split user data into blocks when the data length is greater than the block size. The blocks MUST be full. If 104 bits are to be encrypted using DES algorithm with 64 bit block size, user data is split into two blocks of 64 and 40 bits. The second block must be padded with 24 bits up to the full block size of 64 bits.



Length: Variable length

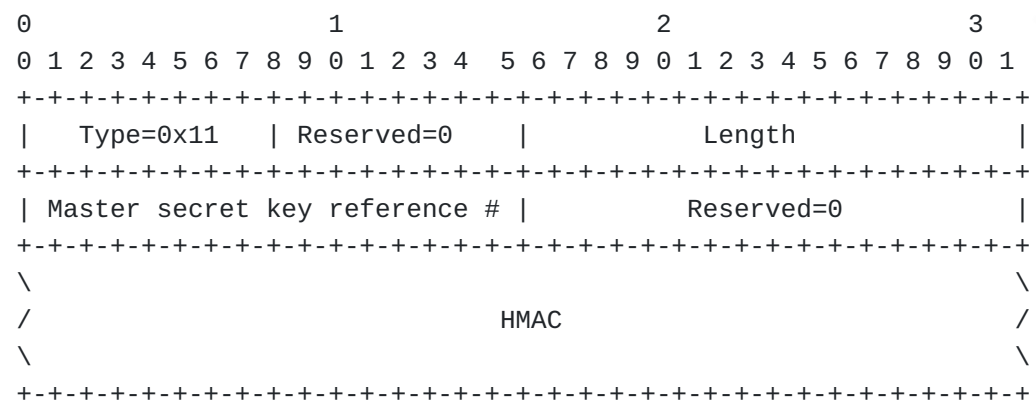
This field indicates the padding size. The padding follows the padding chunk. The length includes the padding chunk and padding.

Padding: Variable length

The padding is a random number. The random number generation is implementation specific.

5.1.13. Authentication chunk (AUTH)

This chunk is dedicated to the authentication of an S-SCTP packet. S-SCTP inserts this chunk into the packet when the security level requires authentication.



Length: 16 bits unsigned integer

The length field contains the size of the chunk in bytes, including the chunk header, master secret key reference, reserved field and MAC.

Master secret key reference number: 16 bits unsigned integer

The association can update the secret keys several times during the association lifetime. The association keeps old secret keys. The number of the kept old secret keys depends on the implementation. This field identifies the key which is used for authentication. If the endpoint receives a message which was authenticated by an old key, this message can still be authenticated after an update of the master secret key.

HMAC: Variable length

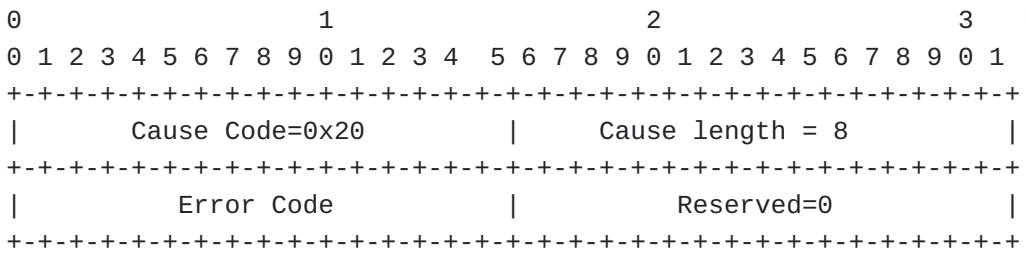
This field contains the authentication code for the SCTP packet. The message authentication uses the HMAC algorithm defined in RFC 2104. The hash function used in the HMAC algorithm is derived from the negotiated cipher suite, which was chosen by the server.

6. New Error Cause

This part explains the new error causes defined for S-SCTP. When a secure session or certificate failure is detected in the secure session open process, the S-SCTP association immediately stops the process. However, the association continues (without any security functionality). When the secure session failure happens during an update of the master secret key the association stops the update operation and closes the secure session. The following table shows four general failure groups.

Cause Code Value	Cause Code
0x20	Secure Session failure
0x21	Secure Session Certificate failure
0x22	Secure Session Decryption failure
0x23	Secure Session Authentication failure
0x24	Secure Session Decompression failure

6.1. Secure Session failure



If any error happens in the secure session open and update process, endpoints alert their peers with these error codes. The next table shows error codes for what can happen.

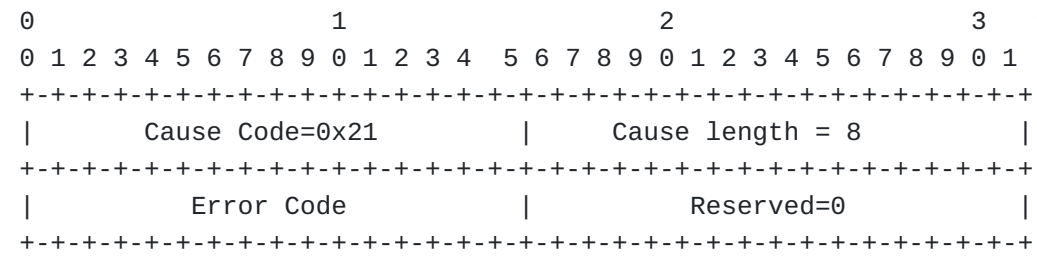
Error Code Value	Error Code
0	Timer expired
1	Signature failure
2	Secure Session Open Complete failure

*Timer expired: The sender starts a timer when it sends the secure session message. When the sender receives no response from the receiver before the timer expires, it sends this error code.

*Signature failure: Some secure session chunks include a signature, which identifies and protects the secure session message. If the receiver checks the signature and cannot identify the chunk, this error code is used in the error chunk.

*Secure Session Open Complete failure: This chunk is a very important part of the secure session. Both server and client individually compute the master secret and HMAC secret keys. Both sides check these secret keys and parameters (i.e. secure session chunks exchanged before, source and destination ports). If these keys are not identical, an error chunk is sent containing this error code.

6.2. Secure Session Certificate failure



The certificate failure signals that an error has occurred in processing the certificates. The next table shows error codes for what can happen.

Error Code Value	Error Code
0	No certificate
1	Bad certificate
2	Certificate expired
3	Unknown certificate

*No certificate: This error happens when the sender sets the CF flag and the receiver does not receive the certificate.

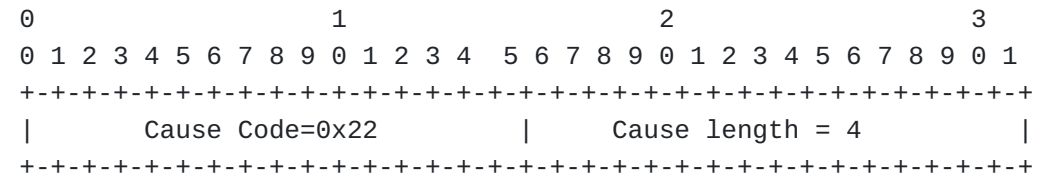
*Bad certificate: The signature of the certificate is bad and the certificate could not be verified.

*Certificate expired: The certificate is no longer valid.

*Unknown certificate: The received certificate a X.509v3 certificate.

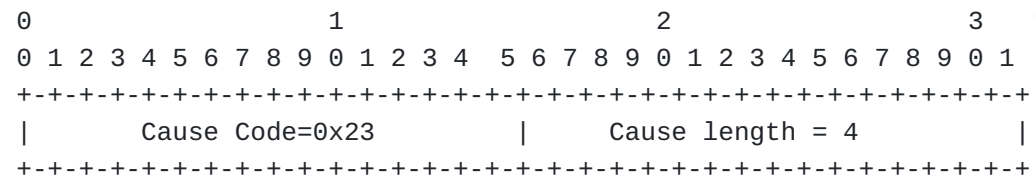
6.3. Decryption failure

This error happens when the EncData chunk cannot be decrypted or the data chunk(s) cannot be identified after decryption. The receiver discards the EncData and increases a counter by 1. This counter counts errors. If the number of errors reaches a limit, the secure session is terminated. The limit of the errors depends on the implementation.



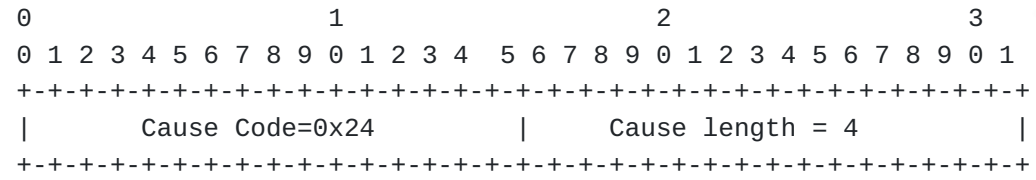
6.4. Authentication failure

In the event of a HMAC error, the packet is discarded by the receiver. To check for an error, the receiver computes the HMAC and compares it to the HMAC field of the packet. If they do not match, an error is reported back.



6.5. Decompression failure

This error happens when the compressed chunk(s) cannot be decompressed or the data chunk(s) cannot be identified after decompression. The receiver discards the decompressed chunk(s).



7. S-SCTP packet format and security levels

S-SCTP has four different security levels, which cover privacy settings of an S-SCTP association. The S-SCTP application can change the security levels at any time during the security session lifetime.

- Security level 0: This is the null security level. S-SCTP does use neither data chunk encryption nor authentication. The S-SCTP packet is the same as the SCTP packet (this level is fully compatible to SCTP).

- Security level 1: This security level requires packet authentication but does not use encryption. Every outgoing packet (including the SCTP common header) is authenticated.

- Security level 2: In this security level, data chunks may be encrypted. When an S-SCTP packet contains an encrypted data chunk, it MUST include an AUTH chunk as well. That means every chunk and the packet header are authenticated. When a packet includes only unencrypted data chunks or control chunks or both unencrypted data chunks and control chunks, the packet will not be authenticated.

*Security level 3: This is the highest security level. S-SCTP requires both encryption and authentication. Every outgoing chunk is encrypted and the packet is authenticated.

Both endpoints can use different security levels, e.g. the association can use security functions only for one direction, e.g. from server to client. In this case the server uses security level 3 and the client uses security level 0. The transmission control block (TCB) of the association includes the security level as a new parameter.

8. S-SCTP data format

S-SCTP sorts data chunks before bundling them into the outgoing SCTP packet. The data chunks are sorted according to whether they have to be encrypted or not. The chunks belonging to the encryption group are concatenated and encrypted into an EncData chunk. May be a PADDING chunk is inserted into the encryption group. Insertion of a PADDING chunk is done depending on data length and encryption block size.

An assortment of encrypted and non-encrypted chunks are bundled in the packet. The control chunk(s) are placed first in the packet when bundled with other chunks. Finally, an AUTH chunk may be added to the packet.

HMAC computation is performed over all chunks and the SCTP common header with a 0 checksum. The checksum is then computed over the complete packet (including AUTH chunk). The HMAC length depends on the hash function in the cipher suite. In every security level, the SCTP packet construction is slightly different. In security level 0 the packet format is same as the SCTP packet format.

9. Procedures

In this section an explanation of the procedures of secure session: initialisation, termination, update and etc., is given.

9.1. Establishment of a secure session

The following process is used to establish the S-SCTP secure session. The handshake process runs in parallel with the data transmission. The secure session start and close is controlled by the user. The user can establish and close a secure session at any time during the association lifetime. Each time a secure session is established, a new set of keys is generated. It is not possible to create a new secure session when a secure session already exists. The following describes secure session establishment, which makes use of a handshake timer and retransmissions in case packets are lost during transmission. S-SCTP uses a four-way handshake. After

all messages of one of the connection "legs" have been sent, client or server starts a RTO.hand (handshake retransmission time out) timer. For example, the secure session certificate is the last handshake message of the first leg. The sender waits for a response from the receiver until the RTO.hand timer expires. The sender stops the RTO.hand timer when it receives the expected message(s). If the RTO.hand timer expires before all expected messages have been received, the sender retransmits the handshake message(s).

The retransmission uses the following algorithm. The RTO.hand timer gets a value from RTO of the path where the message is sent to, which is defined in RFC4960. Before a retransmission, the sender checks RTN.hand.max (handshake maximum retransmission number). This initial value is dependent upon specific implementations. The suggested value for RTN.hand.max is Path.Max.Retrans (see RFC 4960).

RTN.hand.max should be a constant parameter. We introduce a counter for the number of retransmissions, and if that counter exceeds the parameter RTN.hand.max, the timer expired error message is sent to the peer. If a retransmission is required then S-SCTP uses the same retransmission rules as defined in RFC4960. If the receiver receives a retransmission of a handshake message that was already received, the message last received MUST be dropped. The endpoint discards the message(s) when they are unexpected. A secure session initialisation begins when one of the associated endpoints sets the security level to a value higher than 0. The endpoint starting a secure session initialisation is called client and the other associated endpoint is called server.

*The client sends the SSOpReq chunk to the server. If the client has a certificate, it sets the CF flag of the SSOPReq chunk to 1. The client sends the SSCert chunk immediately after the SSOpReq chunk. The SSCert chunk can be bundled with the SSOpReq chunk or with other chunk(s). When the CF flag is set to 0, the client sends only the SSOpReq chunk.

*The server receives a SSOpReq chunk and checks the CF flag. If the CF flag is set to 1, the server waits for the SSCert chunk. Upon receipt, the server checks the certificate and if there is a problem with it, the server stops the handshake and goes to an error state, aborts secure session setup and reports the cause to its peer. It there is no error, the server chooses the cipher suite and sends the SSOpReq_Ack chunk with CF=1 flag to the client when the server has a certificate. The server immediately sends the certificate and the SSSerKey chunks after the SSOpReq_Ack chunk. All three chunks may be bundled together or with other chunks. The server sends only the SSOpReq_Ack chunk with the SSSerKey chunk if CF=0. Before sending the server key exchange chunk, the server generates key material. The server

starts the update master secret key operation when it receives the SSOpReq chunk after secure session establishment. If the server receives the SSCert chunk before the SSOpReq chunk, it stores the SSCert chunk and waits until it receives the SSOpReq chunk. But the server drops a second SSCert chunk.

*The client receives the handshake messages and checks the certificate in the SSSerKey chunk. If the client detects any errors, it stops the handshake and goes to an error state, aborts secure session setup and reports the cause to its peer. The client generates key material and sends the SSCliKey chunk to the server. The client sends the SSOpCom chunk immediately after the client key exchange chunk. Before sending the handshake-finished chunk, the client computes the encryption secret and MAC secret keys.

*The server receives the SSCliKey chunk and computes the master secret and the MAC secret keys. It then computes the SSOpCom chunk and sends it to the client. Finally, the server checks the SSOpCom chunk of the client. If the server detects any error, it reports a secure session open complete error and closes the handshake. The secure session is established only when both sides detect no errors. The server is ready for secure transmission when it detects no errors, but the client must wait for the SSOpCom chunk of the server. When this is received, the client checks it and reports to the peer a secure session open complete error if any error is detected before aborting secure session setup. The handshake may run simultaneously with normal SCTP data transmission. If the client receives encrypted or authenticated data chunks before it receives the server's SSOpCom chunk, then those chunks MUST be discarded.

When both associated endpoints request the initialisation of a secure session simultaneously (both endpoints send an SSOpReq message), both ignore the received SSOpReq message and wait a random time before resending the SSOpReq message. Each endpoint generates the random time independently. The random number must be small, e.g. 120 seconds maximum.

9.2. Choice of cipher suite and compression method

This section explains how to choose the cipher suite and compression method which are used for the secure session. Each endpoint maintains an ordered list of supported cipher suites (cipher suite list). The ordering in the list indicates the preference with which a cipher suite should be used (first in the list have higher preference). The order in the list is defined by the retrospective S-SCTP user.

S-SCTP users on both sides can allow all cipher suited in the list when establishing a secure session or limit the allowed cipher suites to a subset. The complete list or the selected subset can be indicated to the server in the SSOpReq. If the complete list is sent, the default cipher suite list must be located first in the list. The server uses the following rules to choose the cipher suite to be used for the secure session:

The server chooses the default cipher suite, if the SSOpReq chunk does not contain any cipher suite.

The server gets the first cipher suites from SSOpReq chunk and server's cipher suite sequence. When both cipher suites are identical the server chooses this cipher suite for the secure session. Otherwise, the server takes its first cipher suite and looks for a match in the cipher suite sequence of the client. When there is no matche, the server takes the client's first cipher suite and searches for match in its cipher suite sequence. S-SCTP checks the first cipher suite in the SSOpReq chunk against all cipher suites in the cipher suite list of the server. If no match is found, all subsequent cipher suites in the SSOpReq are checked sequentially in the order they appear in the SSOPReq until a match is found. The first cipher suite supported by both endpoints is chosen. When two cipher suites match each other then this cipher suite is selected for the secure session. If not, the server looks, its second cipher suite, for a match in the cipher suite sequence of the client. Furthermore, the server uses the same mechanism to look a cipher suite for the secure session.

The server chooses the default cipher suite, when the cipher suites in the SSOpReq chunk are not supported by the server.

Both client and server also maintain a list of compression methods. The choice of the compression mechanism works similarly to the cipher suite selection mechanism described above. S-SCTP uses a NULL compression method as default compression method.

9.3. Data transfer

Before transporting the packet over the network, S-SCTP takes the following steps. First, it checks the security level. If the security level is:

*0, jump to step "d"

*1, jump to step "c"

*2, check the user data. If the user data requires encryption, jump to step "a" . If the user data does not require encryption, jump to step "c"

*3, jump to step "a"

a) S-SCTP sorts data chunks in two groups, which are encrypted and unencrypted. The encrypted group consists of those data chunks requiring encryption. The unencrypted group consists of those data chunks not requiring encryption. If the secure session's security level is set to 3, all chunks are sorted into the encrypted group.

b) The data chunks in the encrypted group are concatenated. After this, S-SCTP calculates the padding chunk and inserts the padding chunk on the last position into pre-enc-data if necessary. The Pre-enc-data size MUST be smaller than the current MTU. If the pre-enc-data is bigger than the current MTU, S-SCTP must create two pre-enc-datas. Every pre-enc-data is encrypted and stored in the encryption data field of the EncData chunk.

c) SCTP builds the packet according to the security level and inserts the AUTH chunk in the last position in the packet.

d) S-SCTP sends the packet.

9.4. Closing of a secure session

The termination of a secure session begins when one of the endpoints sends the secure session close chunk. This chunk includes the last encrypted data TSN and OF. The endpoint (sender) stops the encryption or authentication of all chunks or packets after it has sent the secure session close chunk. But normal (unsecured) data transfer will continue. The endpoint then waits until it receives the SSClose_Ack chunk. After receiving the SSClose_Ack chunk, the association clears the TCB parameters belonging to the secure session. The receiver (other endpoint) immediately stops encryption and authentication of all chunks or packets after it receives the secure session close chunk. Before sending the SSClose_Ack, the receiver waits for outstanding data (encrypted or authenticated data), which are the receiver's unacknowledged data chunks and sender's data chunks that have a TSN less than the last encrypted data TSN in the SSClose chunk. If the receiver does not receive the outstanding data chunks before RTO.hand timer expires, the S-SCTP association closes the secure session and outstanding data chunks will be dropped. The receiver ignores the last TSN of SSClose chunk and waits only for the receiver's unacknowledged data chunks when SSClose chunk's OF=1. The SSClose and SSClose_Ack chunks may be bundled with other chunks. If the sender does not receive the acknowledge chunk, the client follows the standard retransmission rule for messages. After the termination of the secure session, the TCB parameters belonging to the secure session MUST be set to zero. If the SCTP association begins to close the current association, the

SSClose chunk is sent. If the SCTP association creates an ABORT chunk, the secure session closes immediately and the TCB parameters belonging to the secure session MUST be set to zero.

9.5. Generation of the Master secret key

Secret key generation uses the 3DES_CBC algorithm. Both server and client compute the master secret key separately. The key material is split into 64 bit blocks. Every block will be input to the 3DES_CBC encryption. The key material is as follows:

*If the secure session key exchange algorithm uses DH, the key material consists of the DH's secret key.

*If the secure session key exchange algorithm uses RSA, the key material consists of random numbers of both client and server.

9.6. Update of the master secret key

A secure update mechanism of the secret keys is a very important requirement for a secure session. The secret keys consist of the master secret key, which is used for data chunk encryption, and the HMAC secret key, which is used for packet authentication. If an association exists for a long time, the S-SCTP association needs to update the secret keys. Both the client and the server can request an update of the secret keys. A three way handshake, called an abbreviated handshake, is used to update the master secret keys. All actions of the handshake are encrypted by the current master secret key. The current security level does not affect the packets, which contain the handshake messages. The key update handshake works similar to the first establishment handshake (e.g. the endpoints start an RTO.hand timer when sending handshake chunks). Format and function of the chunks used to update keys are the same as for the handshake. When an endpoint receives a SSOpReq chunk (after a secure session establishment) it begins to update secret keys. Both the server and client key exchange chunks always use the RSA key exchange algorithm. The random numbers in SSSerKey and SSCliKey chunks are encrypted by the current master secret key. The following describes the method used to update the master secret key:

The client generates a random number and sends the SSopReq chunk with the SSCliKey chunk. The key material length in the handshake request chunk may be equal to 0. If not, the number indicates the size of the new key material. If 0, both sides will use the key material length which was used in the last handshake. The server sends the SSop_Ack, the SSSerKey and the SSOpCom chunks immediately after receiving the SSOpReq and the SSCliKey chunks. After receiving the handshake messages from the server, the client computes a new master secret key and checks the SSOpCom chunk of the server. If it

detects any error, the client closes the secure session and reports an error to the peer. The client computes the SSOpCom chunk and sends it to the server. After sending the SSOpCom chunk the client is ready to use the new master secret key. The server receives the SSOpCom chunk of the client and checks the new keys. If it detects any error, the server closes the secure session and reports an error to the peer. Before receiving the client's SSOpCom chunk, the server discards any encrypted or authenticated chunk that make use of the new master secret key.

The encrypted and unencrypted user data transmission works in parallel with the update operation. After the update operation, the new master secret key is used for data encryption and authentication. When both client and server receive an SSOpReq chunk simultaneously, the client ignores the server's SSopReq chunk and the server accepts the client's SSOpReq chunk. The next steps are the same as for the secure session initialisation.

The new master secret key generation uses the same algorithm as described above. The secure session includes one parameter which is called secure session lifetime. This parameter is used to initialise a timer which indicates the secure session secret key's lifetime in seconds. When the timer expires, the association automatically updates the secret keys. The user can define this parameter. If the user does not define it, the parameter assumes a default value. This default value depends on the implementation. The implementation MUST define secure session's lifetime initial value. We suggest a value of 600 seconds for the lifetime as a compromise between security and overhead.

9.7. Random number generation

As the security of S-SCTP depends on the quality of the random number generator, we suggest to use one according to RFC4086 [5].

9.8. HMAC algorithm

S-SCTP uses the HMAC algorithm which is defined in RFC2104 [1] for the packet authentication.

10. HMAC algorithm

ULP-to-SCTP primitives deliver upper layer requests to S-SCTP. The following part describes new ULP-to-SCTP primitives and thus enhances the section 10 of RFC4960. All new ULP-to-SCTP primitives described below are defined in the ssctp.h header file.

INITSECSESS: This primitive initialises a new secure session.

Format: {initSecSess(secure session ID, key material length, cipher suites list, compression methods list, certiticate(s)) --> result}

*secure session ID: This parameter identifies a secure session.

*key material length: This defines the key material length which is used in the SSOPReq chunk.

*cipher suite list: Eligible cipher suites for a new secure session.

*compression method list: Eligible compression methods for a new secure session.

*certificate(s): Local endpoint certificate(s).

SETSECLEVEL: This primitive sets a new security level for an existing secure session.

Format: {setSecLevel(secure session ID, security level) --> result}

*secure session ID: local handle to the secure session

*security level: This parameter indicates the new security level

GETSECLEVEL: This primitive gets the current security level of a secure session.

Format: {getSecLevel(secure session ID) --> security level}

*secure session ID: local handle to the secure session

SENDSEC: This primitive sends secure data via S-SCTP.

Format: {sctp_send_enc(association id, buffer address, byte count, context, stream id, life time, destination transport address, unorder flag, no-bundle flag, payload protocol-id, encryption flag, compression flag) --> result}

Every parameter, except the encryption and compression flags, defined in this function is the same as the corresponding parameter defined in the SEND function of RFC4960 section 10.

*encryption flag: This flag defines if a current user data message needs encryption or not.

*compression flag: This flag defines if a current user data message needs compression or not.

GETSECSTATUS: This primitive gets the security status of an association. The security status indicates if the SCTP association is using a secure session or not.

Format: {setSecStatus(association ID) --> status}

*association ID: local handle to the SCTP association.

SETSECSESSTTL: This primitive sets a new lifetime for a secure session.

Format: {setSecSessTTL(secure session ID, Time) --> result}

*secure session ID: local handle to the secure session.

*time: The new lifetime in seconds.

SHUTSECSESS: This primitive deletes a secure session.

Format: {shutSecSess(secure session ID) --> result}

*secure session ID: local handle to the secure session.

*security level: This parameter indicates the new security level.

11. S-SCTP to ULP

S-SCTP defines new notifications to deliver information to the upper layer. The notifications extend the section 10.2 of RFC4960 [6]. All new notifications are defined in the ssctp.h header file.

SECSESSUP:

This notification indicates that S-SCTP is ready to send or receive secure data ({secsessUpNotif()}).

SECSESSDOWN:

This notification indicates that an association has lost a secure session ({secsessdownNotif()}).

SECSESSREKEY:

This notification indicates that a secure session updated the secret keys ({secsessrekeyNotif()}).

Additional changes had to be made in the socket API implementation to access the new sctplib functions described above. A user calls the same socket API functions as in standard SCTP to send and receive user data, but has to set an additional encryption flag

(MSG_ENC) to request encryption of user data. Also, a compression flag (MSG_COMP) has to be set in ext_send, ext_sendto, ext_sendmsg to request compression of user data. S-SCTP compression performs per user message not per chunk or per packet. In the SCTP DATA chunk, a new flag is defined, which indicates if the data is compressed or not. On the receiver side there are no changes.

12. Transmission Control Block (TCB) extension

A SCTP TCB contains parameters which are related to an association (e.g. an association id, port number, IP address list...). S-SCTP defines several parameters which are related to a secure session and it extends the TCB defined in section 12 of RFC4960.

Security level:

This parameter contains the association's current security level.

Second security level:

This is the security level of the associated second endpoint.

Key material length:

The size of the key material, which was last used for key generation.

Secure session status:

This parameter indicates whether the association is using a secure session or not.

Secure session lifetime:

This parameter indicates the lifetime of the secret keys of a secure session.

Server indication:

This parameter indicates if an endpoint is server or client. If the parameter is equal to 1 then it is a server, otherwise it is a client.

Secure session ID:

This parameter indicates the local secure session ID.

Master secret key reference:

This is an "array of secret data" collection and every array element includes the following parameters.

*Selected cipher suite: This parameter indicates the encryption and authentication algorithms that are used in a secure session.

*Selected compression: This parameter indicates the compression method that is used in a secure session.

*Encryption key: This is a secret key which is used for encryption.

*Authentication key: This is a secret key which is used for authentication.

This information is used in EncData and AUTH chunks.

13. Socket API extensions for Secure SCTP

S-SCTP defines new socket options for the ext_setsockopt() and ext_getsockopt() socket functions to initialise, delete and rekey a secure session. A user calls the ext_setsockopt or ext_getsockopt functions with a new option. It is not necessary to define new socket API functions, as this is a more standard socket API fashion. The following paragraphs describe the new socket options.

SSCTP-INIT:

This socket option is used to initialise or update a secure session. The following structure is used to access these parameters.

```
struct ssctp_init {
       uint16_t  secsessID;
       uint16_t  key_length;
       uint8_t   num_cipher;
       uint8_t   *cipher_suites;
       uint8_t   num_comp;
       uint8_t   *comp_methods;
       uint8_t   *certificate;
};
```

*secsessID: This parameter indicates a current secure session ID.

*key_length: This parameter defines the length of a key material.

*num_cipher: This parameter defines the number of cipher suites.

*cipher_suites: This parameter includes a list of cipher suites.

*num_comp: This parameter defines the number of compression methods.

*comp_methods: This parameter includes a list of compression methods.

*certificate: This parameter includes a certificate of the endpoint.

SSCTP-SECLEVEL:

This socket option is used to set and get a secure session security level. The following structure is used to access and modify these parameters.

```
struct ssctp_seclevel {
       uint16_t  secsessID;
       uint8_t  seclevel;
};
```

*secsessID: This parameter indicates a current secure session ID. This parameter MUST be zero when beginning a secure session initialisation.

*seclevel: This parameter contains a new security level before socket write access or contains the current security level after socket read access.

SSCTP-SECSTATUS:

This socket option is used to get the secure session status and secure session ID when a secure session exists. The following structure is used to access these parameters.

```
struct ssctp_secstatus {
       uint16_t secsessID;
       uint8_t sec_status;
};
```

*secsessID: This parameter contains the current secure session ID. This parameter MUST be zero when a secure session does not exist.

*sec_status: This parameter contains a security status. This parameter MUST be zero when a secure session does not exist. This parameter is equal to 1 when a secure session exists.

SSCTP-SECSESSTTL:

This socket option is used to set and get the secure session lifetime. The following structure is used to access and modify these parameters.

```
struct ssctp_secsessTTL {
       uint16_t secsessID;
       uint16_t secsessTTL;
};
```

*secsessID: This parameter indicates the current secure session ID.

*secsessTTL (seconds): This parameter contains a new secure session lifetime before socket write access or contains a current secure session lifetime after socket read access.

SSCTP-CLOSE:

This socket option is used to close an existing secure session. The following structure is used to access these parameters.

```
struct ssctp_secclose {
       uint16_t  secsessID;
};
```

*secsessID: This parameter contains the current secure session ID.

14. Testbed Platform

A large-scale and realistic Internet testbed platform with support for the multi-homing feature of the underlying SCTP protocol is NorNet. A description of NorNet is provided in [9], some further information can be found on the project website [10].

15. Security Considerations

Security has been described in the previous sections.

16. IANA Considerations

This document introduces no additional considerations for IANA.

17. References

17.1. Normative References

[1] 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>.

[2] 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>.

[3] Jungmaier, A., Rescorla, E., and M. Tuexen, "Transport Layer Security over Stream Control Transmission Protocol", RFC 3436, DOI 10.17487/RFC3436, December 2002, <https://www.rfc-editor.org/info/rfc3436>.

[4] Bellovin, S., Ioannidis, J., Keromytis, A., and R. Stewart, "On the Use of Stream Control Transmission Protocol (SCTP) with IPsec", RFC 3554, DOI 10.17487/RFC3554, July 2003, <https://www.rfc-editor.org/info/rfc3554>.

[5] 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>.

[6] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <https://www.rfc-editor.org/info/rfc4960>.

[7] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[8] 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>.

17.2. Informative References

[9] Dreibholz, T. and E. G. Gran, "Design and Implementation of the NorNet Core Research Testbed for Multi-Homed Systems", Proceedings of the 3nd International Workshop on Protocols and Applications with Multi-Homing Support (PAMS) Pages 1094-1100, ISBN 978-0-7695-4952-1, DOI 10.1109/WAINA.2013.71, 27 March 2013, <https://www.simula.no/file/threfereedinproceedingsreference2012-12-207643198512pdf/download>.

[10] Dreibholz, T., "NorNet – A Real-World, Large-Scale Multi-Homing Testbed", 2022, <https://www.nntb.no/>.

Authors' Addresses

Carsten Hohendorf
University of Duisburg-Essen, Institute for Experimental Mathematics
Ellernstraße 29
45326 Essen
Germany

Email: hohend@iem.uni-due.de

Esbold Unurkhaan
Mongolian University of Science and Technology
Bayanzurkh duureg, 2-nd khoroo
313/49 Ulaanbaatar
Mongolia

Email: esbold@csms.edu.mn

Thomas Dreibholz
Simula Metropolitan Centre for Digital Engineering
Pilestredet 52
0167 Oslo
Norway

Email: dreibh@simula.no
URI: https://www.simula.no/people/dreibh