Secure End-to-End Transport over SCTP

A new security extension for SCTP

D I S S E R T A T I O N

zur Erlangung des Grades
Doktor der Naturwissenschaften
(Dr. rer. nat.)
in Informatik

vorgelegt dem
Fachbereich Wirtschaftswissenschaften
Institut für Informatik und Wirtschaftsinformatik (ICB)
Universität Duisburg-Essen (Campus Essen)

von
MSc Esbold Unurkhaan
geboren am 20.01.1975 in Ulaanbaatar, Mongolei

Gutachter:

1. Prof. Dr.-Ing. Erwin P. Rathgeb
2. Prof. Dr. Klaus Echtle

Tag der mündlichen Prüfung: 24.06.2005

Abstract

In 2000, the Signaling Transport (SIGTRAN) working group of the IETF defined the Stream Control Transmission Protocol (SCTP) as a new transport protocol. SCTP is a new multi-purpose reliable transport protocol. Due to its various features and easy extensibility it is a valid option not only for already standardised applications but also in many new application scenarios. SCTP has several advantages over TCP and UDP.

The analysis of already standardised as well as potential SCTP application scenarios clearly indicates that secure end-to-end transport is one of the crucial requirements for SCTP in the future. Up to now there exist two standardised SCTP security solutions which are called TLS over SCTP [37] and SCTP over IPSec [12].

The goal of this thesis was to evaluate existing SCTP security solutions and find an optimised and efficient security solution. Several drawbacks of the standardised SCTP security solutions identified during the analysis are mainly related to features distinguishing SCTP from TCP and UDP. To avoid these drawbacks a new security solution for SCTP, called Secure SCTP (S-SCTP), is proposed which integrates the cryptographic functions into SCTP.

One main requirement was that S-SCTP should be fully compatible with standard SCTP while additionally providing strong security i.e. data confidentiality, integrity and authentication. This also means that all features, options and extensions available for standard SCTP have to be supported. Furthermore, S-SCTP should have advantages with respect to performance over all parameter ranges of SCTP and be user-friendly.

To specify the S-SCTP protocol extension several new control messages and new message parameters have been defined. Furthermore, procedures for initialisation, rekeying, and termination of secure sessions have been specified and modelled in SDL.

Based on an SCTP implementation available in our group and an open source implementation of TLS, TLS over SCTP and S-SCTP have been implemented. These implementations as well as an SCTP over IPSec configuration were used to do comparative performance studies in a lab testbed.

These experiments show that the S-SCTP concept achieves its design goals. It supports all features and current extensions of SCTP. Furthermore, it avoids the inefficiencies of the other solutions over a wide range of application scenarios and protocol parameter settings.

Keywords: SCTP, Transport Protocol, Security, TLS, IPSec, Performance Evaluation
Schlagworte: SCTP, Transportprotokoll, Sicherheit, TLS, IPSec, Leistungsbewertung

I dedicate this work to
my wife Gerelchuluun Tesee,
my father Saudavai Unurkhaan and
my mother Baatarbal Sanak

Acknowledgements

On 15 April 2001 I arrived in Essen, Germany, from Mongolia, to start my studies as a doctoral student in the Post-Graduate School (Graduiertenkolleg) ”Mathematical and Engineering Methods for Secure Data Transmission and Information Transfer”. In the three years that I have spent in Essen, I have had a wonderful time. I have worked hard on my thesis, but have also been able to learn much about German culture and language. Although I may not speak ”fließend Deutsch”, it was great for me to learn as much as I have of a new language. I have learned many, many things during my time here, but most importantly, I have developed professionally.

I begin with thanking my thesis advisor Prof. Dr.-Ing. Erwin P. Rathgeb. He first invited me to Germany and has always supported me during this period, both academically and personally.

My thanks also go to my second reviewer Prof. Dr. Klaus Echtle for his valuable comments and to Prof. Dr. Bruno Müller-Clostermann as chairman of my examination committee.

I would also like to thank Prof. Dr. Gerhard Frey and the members of the Post-Graduate School as well as the other students of the Post-Graduate School for their support.

Thanks go to my colleague Andreas Jungmaier, a great guy, who worked on the same subject and was always very helpful to me. It is so good when people can be both work colleagues and personal friends. I want to thank Thomas Dreibholz, with whom I have shared an office for three years. He too is a great guy, who also worked on SCTP, and so we were able to have interesting conversations. Next I would like to thank Dr. Avril IJsselmuiden. She came from Oxford, England, to work in our group, and was a great help in proof-reading my thesis.

I would like to thank my other wonderful colleagues: Holger Bleul, Birger Toedtmann, Nicola Altan, Anja Wiedemann and Birgit Hasel. They are really nice people with whom I enjoyed working.

Finally, I want to thank from my heart all the people who supported and helped me in Germany.

Unurkhaan Esbold

Contents

List of Figures

List of Tables

Glossary

3DES	Triple Data Encryption Standard
a_rwnd	Advertised receiver window credit
AAA	Authentication, Authorization and Accounting
AES	Advanced Encryption Standard
AH	IP Authentication Header
API	Application Program Interface
ASAP	Aggregate Server Access Protocol
ASCONF	Address Configuration Change (SCTP chunk)
ASP	Application Service Part
AUTH_DATA	Authentication Data (S-SCTP chunk)
CA	Certification Authority
CBC	Cipher Block Chaining mode
CDMA	Code Division Multiple Access
CliDHpar	Client's DH parameter (S-SCTP parameter)
CliRSApar	Client's RSA parameter (S-SCTP parameter)
COOKIE – ACK	Cookie Acknowledgement (SCTP chunk)
COOKIE – ECHO	Cookie Echo (SCTP chunk)
CRC32	Cyclic Redundancy Check - 32 bits
cwnd	Congestion Window
DES	Data Encryption Standard
DH	Diffie-Hellman key exchange algorithm
DOI	Internet IP Security Domain of Interpretation for ISAKMP
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
ENC_DATA	Encryption Data (S-SCTP chunk)
ENRP	Endpoint Name Resolution Protocol
ESP	IP Encapsulating Security Payload
FORWARD – TSN	Forward transmission sequence number (SCTP chunk)
Freeswan	Open source IPsec implementation

GMPY	General Multiprecision PYthon project
GSM	Global System for Mobile Communication
HEARTBEAT	Heartbeat (SCTP chunk)
HEARTBEAT − ACK	Heartbeat Acknowledgement (SCTP chunk)
HMAC	Keyed-Hash Message Authentication Code
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
ICV	Integrity Check Value
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange
IN	Intelligent Network
INIT	Initialization (SCTP chunk)
INIT − ACK	Initialization Acknowledgement (SCTP chunk)
IP	Internet Protocol
IPcomp	IP Payload Compression Protocol
IPsec	Internet Protocol Security
ISAKMP	Internet Security Association and Key Management Protocol
ISDN	Integrated Services Digital Network
ISUP	ISDN User Part
ITU − T	International Telecommunication Union - Telecommunication Standardization Sector
IV	Initialization Vector
KAME	IPsec open source implementation
M2PA	SS7 MTP2-User Peer-to-Peer Adaptation Layer
M2UA	Signaling System 7 (SS7) Message Transfer Part 2 (MTP2) - User Adaptation Layer
M3UA	Signaling System 7 (SS7) Message Transfer Part 3
MAC	Message Authentication Code
MD5	MD5 Message Digest algorithm
MEGACO	Media Gateway Control Protocol
MG	Media Gateway
MGC	Media Gateway Controller
MTP	Message Transfer Part
MTU	Maximum Transmission Unit
NAS	Network Access Server
NS	Name Server
OOTB	Out-of-the-blue
OpenSSL	Open source TLS implementation

OS	Operating system
OSPF	Open Shortest Path First routing protocol
PCT	Private Communication Technology
PDU	Protocol Data Unit
PE	Pool Element
PMTU	Path Maximum Transmission Unit
PPID	Protocol Payload ID
PR – SCTP	Partial Reliability Extension
PSTN	Public Switched Telephone Network
PU	Pool User
QoS	Quality of Service
RADIUS	Remote Authentication Dial-in User Service
RC4	RC4 encryption algorithm
RFC	Request for Comment
RSA	Rivest-Shamir-Adleman
RSerPool	Reliable Server Pooling
RTCP	RTP Control Protocol
RTP	Real Time Transport Protocol
RTSP	Real Time Streaming Protocol
RTT	Round Trip Time
rwnd	Receiver window
S – SCTP	Secure SCTP
S/MIME	Secure/Multipurpose Internet Mail Extensions
SA	Security Association
SACK	Selective Acknowledgement (SCTP chunk)
SAD	Security Association Database
SAP	Session Announcement Protocol
SCCP	Signalling Connection Control Part
SCP	Service Control Point
SCTP	Stream Control Transmission Protocol
SDL	Specification and Description Language
SDP	Session Description Protocol
SecLevCHD	Security Level Changed (S-SCTP chunk)
SecLevCHDAck	Security Level Changed Acknowledgement (S-SCTP chunk)
SerDHpar	Server's DH parameter (S-SCTP parameter)
SerRSApar	Server's RSA parameter (S-SCTP parameter)
SG	Signalling Gateway
SHA	Secure Hash Algorithm

SHUTDOWN	Shutdown (SCTP chunk)
SHUTDOWN – ACK	Shutdown Acknowledgement (SCTP chunk)
SHUTDOWN – COMPLETE	Shutdown Complete (SCTP chunk)
SIGTRAN	Signalling Transport
SIP	Session Initialization Protocol
SKEME	Secure Key Exchange Mechanism for Internet
SPD	Security Policy Database
SPI	Security Parameter Index
SRP	Strong Password Authentication
SS7	Common Channel Signalling System Number 7
SSCert	Secure Session Certificate (S-SCTP chunk)
SSCliKey	Secure Session Client Key (S-SCTP chunk)
SSClose	Secure Session Close (S-SCTP chunk)
SSClose_Ack	Secure Session Close Acknowledgement (S-SCTP chunk)
SSL	Secure Socket Layer
SSOpCom	Secure Session Open Complete (S-SCTP chunk)
SSOpReq	Secure Session Open Request (S-SCTP chunk)
SSOpReq_Ack	Secure Session Open Acknowledgement (S-SCTP chunk)
SSP	Service Switching Point
SSSerKey	Secure Session Server Key (S-SCTP chunk)
ssthresh	Slow Start Threshold
STP	Signal Transfer Point
SUA	Signalling Connection Control Part User Adaptation Layer
T3	SCTP retransmission timer
TCAP	Transaction Capabilities Application Part
TCB	Transmission Control Block
TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
TLS	Transport Layer Security
TSN	Transmission Sequence Number
TSVWG	Transport Area Working Group
TUP	Telephone User Part
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
VoIP	Voice over IP
X.509	X.509 certificate
X.509v3	X.509 certificate version 3

Chapter 1

Summary

Nowadays the Internet is developing rapidly and every day brings new versatile applications, which are strongly requiring a reliable and secure transport over the Internet (e.g. e-commerce, realtime delivery...). Until 2000 there were only two transport protocols: User Datagram Protocol (UDP) for unreliable transport and Transmission Control Protocol (TCP) for reliable transport. These both transport protocols still play the most important roles in the Internet, but both protocols have been defined two decades ago and now in some situations they are not sufficient for the application's requirements.

In 2000, the Signaling Transport (SIGTRAN) working group of the Internet Engineering Task Force (IETF) defined a new transport protocol which was named Stream Control Transmission Protocol (SCTP). SCTP is a new multi-purpose reliable transport protocol which due to its various features and easy extensibility is a valid option not only for already standardised applications but also in many new application scenarios. SCTP has several advantages over TCP and UDP.

SCTP is a relatively new protocol and, therefore, today only a few applications exist. But the analysis of already standardised as well as potential application scenarios clearly indicates that secure end-to-end transport is one of the crucial requirements for SCTP in the future. Up to now there exist two standardised SCTP security solutions which are called TLS over SCTP [37] and SCTP over IPSec [12].

The goal of this thesis was to evaluate existing SCTP security solutions and find an optimised and efficient security solution for the protocol. During the analysis of the standardised SCTP security solutions I discovered several drawbacks. These drawbacks are mainly related to SCTP features which are distinguishing SCTP from TCP and UDP and are, therefore, not fully supported by the solutions defined for the latter two protocols.

To avoid these drawbacks I propose a new security solution for SCTP, called Secure SCTP (S-SCTP), which integrates the crypto functions into SCTP.

One main requirement was that S-SCTP should be fully compatible with standard SCTP while additionally providing strong security i.e. data confidentiality, integrity and authentication. This also means that all features, options and extensions available for standard SCTP have to be supported. Furthermore, S-SCTP should have advantages with respect to performance over all parameter ranges of SCTP and be user-friendly.

In order to achieve these goals I completely defined the S-SCTP protocol extension including a message syntax, functions and procedures and modeled it in SDL (Specification and Description Language).

S-SCTP procedures include initialisation, rekeying, and termination of secure sessions. Several newly defined messages such as Secure Session Open Request, Secure Session Close, etc. are used to perform these procedures. Additionally, several new parameters for the messages had to be defined.

Our group had already developed prototype implementations of SCTP and a socket API in cooperation with an industry partner. Based on this implementation and an open source implementation of TLS, I implemented TLS over SCTP and S-SCTP. In a lab testbed, I did several experiments comparing S-SCTP with the existing security solutions.

These experiments show that the S-SCTP concept achieves its design goals. It supports all features and current extensions of SCTP and, furthermore, avoids the inefficiencies of the other solutions over the complete set of application scenarios and protocol parameter settings.

1.1 Organisation of the thesis

The thesis consists of seven chapters which are organised as follows:

- **Chapter 1:** This chapter gives a brief overview of the thesis.
- **Chapter 2:** This chapter gives an overview of SCTP, already existing SCTP application scenarios and their respective security requirements.
- **Chapter 3.** This chapter introduces SCTP and its features and extensions in detail. It also includes a comparison between SCTP, TCP and UDP. In this chapter, also the relevant security protocols TLS and IPsec are introduced.
- **Chapter 4.** This chapter provides an overview of the two already standardised SCTP security solutions which are called TLS over SCTP and SCTP over IPsec.
- **Chapter 5.** This chapter presents the new SCTP security solution named Secure SCTP (S-SCTP) which I designed based on the results obtained in chapter 4. This

chapter contains the concept, the design goals and procedures (e.g. initialisation, deletion and rekying of a secure session) of S-SCTP. All S-SCTP procedures are specified in detail in a SDL model also described in this section. In addition, required new control and data chunks are described giving their specific data structures and parameters.

- **Chapter 6.** This chapter describes the test setup for the SCTP security solutions and the results of the experiments, which I did for all three SCTP security solutions. It also describes the setup of the lab testbed which was used in the experiments for each security solution.
- **Chapter 7.** This chapter summarizes the main points and results of this thesis and indicates possible future work areas.
- **Appendix A.** Contains the SDL model of S-SCTP with detailed descriptions.
- **Appendix B.** Contains the detailed technical specification of S-SCTP.

Chapter 2

Introduction

2.1 History and overview of SCTP

Both UDP and TCP transport user data over IP. However, there are differences between the two: UDP is an unreliable transport protocol and TCP is a reliable transport protocol; UDP is suitable for multicasting and broadcasting, but TCP provides a unicast service and is unable to provide a multicast/broadcast service.

These protocols have been defined over the past 4 decades, starting in the late 1960s as a government financed research project into packet switching networks, and evolving into the most widely used form of networking today. (See Lynch [64] for a historical perspective on the development of the TCP/IP protocol suite). However, specific new applications continue to emerge, for which neither UDP nor TCP are suitable transport protocols. A new transport protocol is required to support these new applications. The Stream Control Transmission Protocol (SCTP) was developed in response to those requirements, which will be described in detail later.

Originally, SCTP was designed to transport telephony signalling messages over the Internet. Telephony signalling has very strict timing and reliability requirements, often subject to government regulations. The Signalling Transport (SIGTRAN) group of the Internet Engineering Task Force (IETF) defined the Stream Control Transmission Protocol in RFC 2960 in October 2000 [75].

The RFC2960 is not the final version of SCTP and at the time of writing, the Transport Area working group (TSVWG) of IETF is currently working on SCTP and defining new changes and extensions. The new changes or new future specifications are defined in internet draft [72]. The internet draft is renewed when new changes are created. The TSVWG also defined two new extensions which are named SCTP Partial Reliability Extension [73] and Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration [74]. These are described in the next chapter.

SCTP [75] is a general purpose, end-to-end, connection-oriented protocol that transports data in independent sequenced streams, over IP. It has several advantages over UDP and TCP. SCTP combines the message orientation of UDP with the sequencing and reliability of TCP. Due to the message orientation, SCTP applications do not have to perform record marking. The conservation of message boundaries (message delineation) is performed by SCTP. This is known as conservation of message boundaries. This is unlike TCP, which transmits a sequence of data bytes, ordered, but with no conservation of block boundaries. SCTP explicitly supports multi-homed endpoints, i.e. endpoints with more than one IP address [13]. SCTP supports multiple IP addresses to use redundant paths. When the primary path is unreachable then the redundant path is used to transport messages. This increases the reliability of the communication of two associated endpoints, which in TCP is not possible.

SCTP provides the users with flexible methods of data delivery, whereby they can request messages to be delivered by an ordered or unordered service. Several independent message streams may be multiplexed into a single connection, and received without interference from other streams. When a message is blocked, messages of other streams are unaffected. This procedure avoids head-of-line blocking.

Some attacks feasible for TCP also can happen in SCTP, for example man-in-the-middle attacks, but SCTP has some security advantages over TCP. For example, it avoids blind denial of service attacks. The SCTP association initialisation uses a four way handshake, which reduces the risk of blind attacks and IP spoofing.

Due to these features, SCTP is a more versatile transport protocol than TCP. SCTP cannot, however, provide data confidentially and authentication. If a user or application wants to use data confidentially and requires authentication, then they have to combine SCTP with security protocols such as Transport Layer Security (TLS) and Internet Protocol Security (IPsec). The combinations of SCTP with TLS and IPsec are already standardised. These are described in chapter 4.

2.2 SCTP application scenarios and their security requirements

Because SCTP is a relatively new protocol, there are, at the time of writing, not many widely used applications which use SCTP or which are standardised for use with SCTP. However, the already existing standardised and possible future applications of SCTP clearly motivate the need for secured end-to-end transport solutions for SCTP. In this section, we describe some of the major SCTP applications and comment on their security requirements.

2.2.1 Signalling Transport over SCTP

The Common Channel Signalling System Number 7 (SS7, [85]) is the protocol architecture defined by CCITT (ITU-T) for inter-exchange signalling in ISDN networks. In the past, in-band signalling techniques were used on inter-exchange trunk lines. This method uses the same physical path for both the call-control signalling and the actual connected call. However, it is inflexible, and so, out-of-band or common-channel signalling techniques were developed. A common-channel signalling network carries the user data (voice/data) and the call control traffic on different channels. Today, however, SS7 is commonly used not only for ISDN call control, but also to support remote system management, mobility management for GSM networks and Intelligent Network (IN) functionality. Figure 2.1 shows a simple example for the architecture of an SS7 network (details about SS7 can be found in [67])

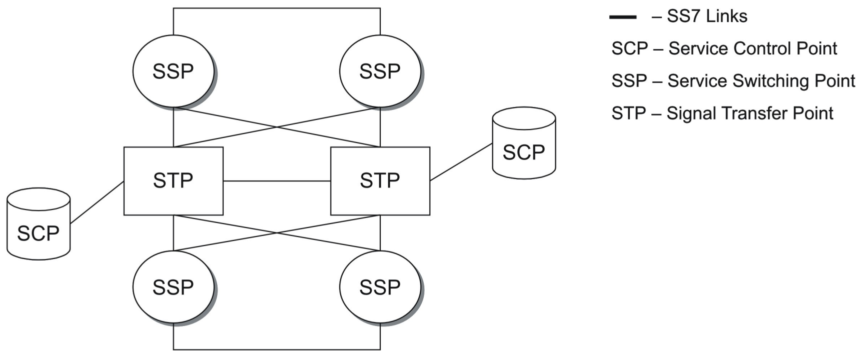

– SS7 Links
SCP – Service Control Point
SSP – Service Switching Point
STP – Signal Transfer Point
SSP
SSP
STP
STP
SCP
SCP
SSP
SSP

Figure 2.1: Simple SS7 network

The signalling points shown in figure 2.1 have following functionalities:

- SSP: Service Switching Points are switches that originate, terminate, or tandem calls. Typically, ISDN switches provide the SSP functionality.
- STP: Signal Transfer Points route signalling messages from one signalling link to another. STPs do not handle any calls, they just relay signalling messages.
- SCP: Service Control Points implement services that affect the recording, processing, transmission, or interpretation of data. SCPs typically offer functions for the

Intelligent Network (IN), e.g. 0800-services, or provide mobility support for a GSM network.

The SS7 protocol stack consists of several *parts* and figure 2.2 shows its structure in relationship to the OSI layers.

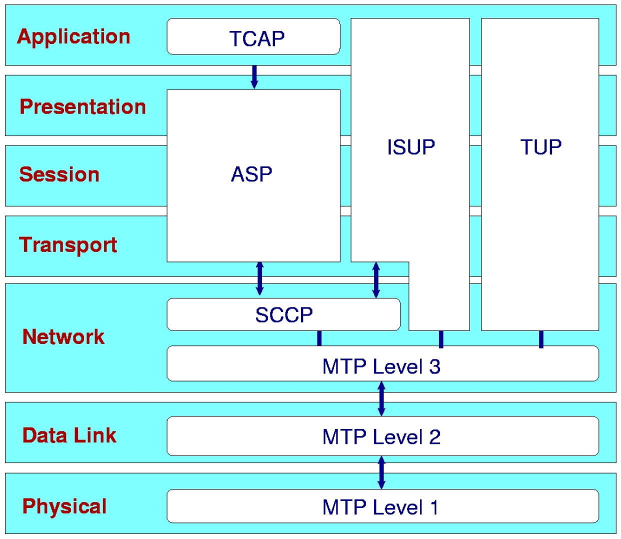


Figure 2.2: SS7 structure

MTP-1: Message Transfer Part Level 1 (MTP-1) is equivalent to the OSI Physical Layer and it defines the physical, electrical, and functional characteristics of the digital signalling link. Physical interfaces defined include E-1 (2048 kb/s; 32 64 kb/s channels), DS-1 (1544 kb/s; 24 64 kp/s channels), V.35 (64 kb/s), DS-0 (64 kb/s), and DS-0A (56 kb/s).

MTP-2: Message Transfer Part Level 2 (MTP-2) is a bit-oriented link layer protocol which together with MTP-1 provides reliable transfer of signalling messages between two directly connected Signalling Points. MTP-2 has the following functionalities: signal unit delimination/alignment, error detection/correction, signaling link initial alignment, signaling link error monitoring and flow control.

MTP-3: Message Transfer Part Level 3 (MTP-3) provides functions related to message routing (simple connectionless service) between Signalling Points and network management in the SS7 network. Every message includes routing information and MTP-3 decides upon which link to route the message. The signaling network management functions are divided into the following three categories:

- Signalling link management
- Signalling route management
- Signalling traffic management

SCCP: The Signalling Connection Control Part (SCCP) enhances MTP-3 and provides e.g connection oriented services. The users are Transaction Capabilities Application Part (TCAP) or ISDN User Part (ISUP). SCCP provides the following five classes of service:

- Class 0 Basic connectionless
- Class 1 Sequenced connectionless
- Class 2 Basic connection oriented
- Class 3 Flow control connection oriented
- Class 4 Error recovery and flow control connection oriented

Today's networks mainly use services of the classes 0 and 1, while the other classes (2, 3 and 4) are not used in today's networks. For example, the SCCP class 0 service provides the transport of TCAP messages.

ASP: The Application Service Part transfers messages between TCAP and SCCP. Until now, ASP functionality is not required in the SS7 network as TCAP uses SCCP directly.

TCAP: Transaction Capabilities Application Part (TCAP) enables the deployment of advanced intelligent network services by supporting non-circuit related information exchange between signalling points using the SCCP connectionless service. TCAP enables communication between end users.

ISUP: The ISDN User Part (ISUP) defines the protocols and procedures used to setup, manage and release trunk circuits that carry voice and data calls over the public switched telephone network (PSTN). ISUP is used for both ISDN and non-ISDN calls. Calls that originate and terminate at the same switch do not use ISUP signalling. ISDN User Part messages are carried on the signalling link by means of signal units.

TUP: The Telephone User Part (TUP) can be used for non-ISDN telephone signalling. Difference between TUP and ISDN are the message types and parameters. Otherwise TUP is compatible with the ISUP.

SS7 over IP

In the future, telephone companies might offload voice calls from public switched telephone networks (PSTNs) to voice-over-IP (VoIP) networks because it is more cost efficient to carry voice traffic over IP networks than over circuit-switched networks. Also, IP telephony networks are expected to enable innovative multimedia services while working seamlessly with legacy telephone networks. For this purpose, the SIGTRAN working group of the Internet Engineering Task Force (IETF) is working on the transport of PSTN signalling over IP networks. The SIGTRAN working group defined the Stream Control Transmission Protocol (SCTP) [75], which is dedicated to transporting telephone signalling over IP based networks. Other SIGTRAN protocols are designed to carry signalling messages for telephony services. Figure 2.1 shows a VoIP network structure internetworking with a classical ISDN network.

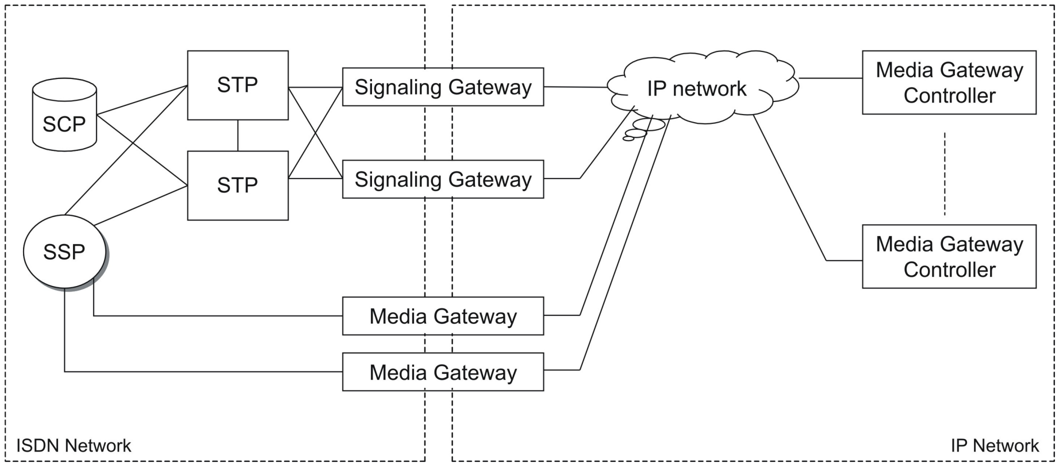


Figure 2.3: Internetworking structure of SS7 and IP network

The following functional elements are used in such a scenario:

Media Gateway: A Media Gateway (MG) terminates voice calls on inter-switch trunks from the PSTN, compresses and packetises the voice data, and delivers compressed voice packets to the IP network. For voice calls originating in an IP network, the media gateway performs the inverse functions.

Media Gateway Controller: A Media Gateway Controller (MGC) handles the registration and management of resources at the MGs. A MGC exchanges ISUP messages with central office switches via a Signaling Gateway.

Signalling Gateway: A Signalling Gateway (SG) provides transparent interworking of signalling between circuit switched and IP networks. The SG may terminate SS7 signalling or translate and relay messages over an IP network to a MGC or another SG. Because of their critical role in integrated voice networks, SGs are often deployed in groups of two or more to ensure high availability.

All three gateways may be separate physical devices or integrated in any combination.

The SIGTRAN working group defined MTP-2 User Adaptation Layer (M2UA) [48], MTP-3 User Adaptation Layer (M3UA) [71] and SCCP User Adaptation Layer (SUA) [42] to transport the SS7 messages over IP networks. Figure 2.4 shows one example usage of SIGTRAN protocols.

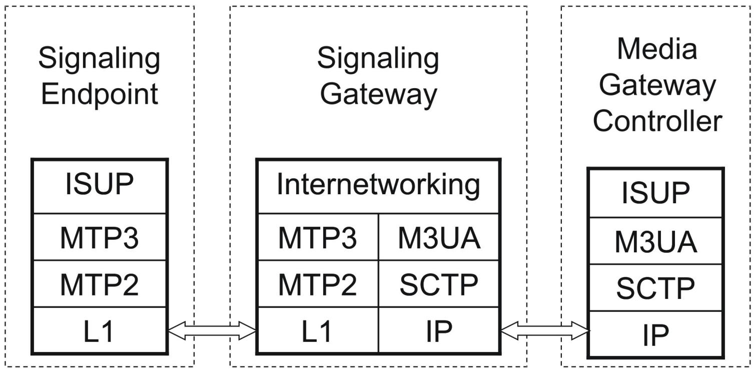


Figure 2.4: MTP3-User adaptation layer

The SIGTRAN protocols specify the means by which SS7 messages can be reliably transported over IP networks. The architecture identifies two components: a common transport protocol for the SS7 protocol layer being carried and an adaptation module to emulate lower layers of the protocol. For example, if the native protocol is MTP (Message Transfer Part) Level 3, the SIGTRAN protocols provide the equivalent functionality of MTP Level 2. If the native protocol is ISUP or SCCP, the SIGTRAN protocols provide the same functionality as MTP Levels 2 and 3. If the native protocol is TCAP, the SIGTRAN protocols provide the functionality of SCCP (connectionless classes) and MTP Levels 2 and 3. SIGTRAN protocols provide all functionality to support SS7 signalling over IP networks.

Security considerations

SS7 networks and their security rely strongly on the fact that access for end-users is not possible and that they operate within a closed, monitored and strongly regulated environment. These networks must adhere to national and international standards. SS7 transport over IP networks should be as secure as traditional, TDM-based signaling transport based on the SS7 Message Transfer Part (MTP), and provide similar performance. Since there is not yet an international standard for security mechanisms that can protect the protocol stacks defined by the IETF (e.g. M2PA [30], SUA [42], M2UA [48] or M3UA [71]), the SIGTRAN group has started to discuss security considerations for SIGTRAN protocols in [43]. The document analyzes the common threats to the SIGTRAN protocols, and makes recommendations, to which security mechanisms nodes supporting SIGTRAN based communication should implement in order to guarantee a minimum security level. All SIGTRAN protocols use SCTP as transport, and the goal is a sufficient end-to-end security. This can be achieved by deploying either one of the standard solutions described in section 3.2, namely TLS over SCTP [37] or SCTP over IPsec [12].

For SCTP over IPsec, encryption is required to protect the privacy of the data. In some cases it is necessary to protect IP layer information as well. For peer authentication and establishment of (new) security associations (SA), the Internet Key Exchange (IKE) [34] must be supported by SIGTRAN nodes. With IKE, different authentication mechanisms are possible but at least support of pre-shared keys is required.

SIGTRAN sessions that are secured by TLS over SCTP must be mutually authenticated between client and server. TLS [21] must be used on all bi-directional streams, and unidirectional streams are not allowed. This ensures that *all* data exchanged between two SIGTRAN nodes is protected. [43] explicitly allows for a session upgrade, and defines how two SIGTRAN nodes can use STARTTLS and STARTTLS-ACK messages to initiate a TLS connection.

[43] states, that for sufficient security, SIGTRAN nodes *must* properly support IPsec and *may* optionally support TLS.

2.2.2 Reliable Server Pooling (RSerPool)

The Reliable Server Pooling (RSerPool) working group of the IETF is defining an architecture [84] and protocols for the management of server pools. Figure 2.5 shows the architecture of RSerPool.

While SCTP provides redundancy at the network level, RSerPool aims at maintaining availability of services by managing sessions between redundant elements within server pools. The RSerPool architecture defines the following elements:

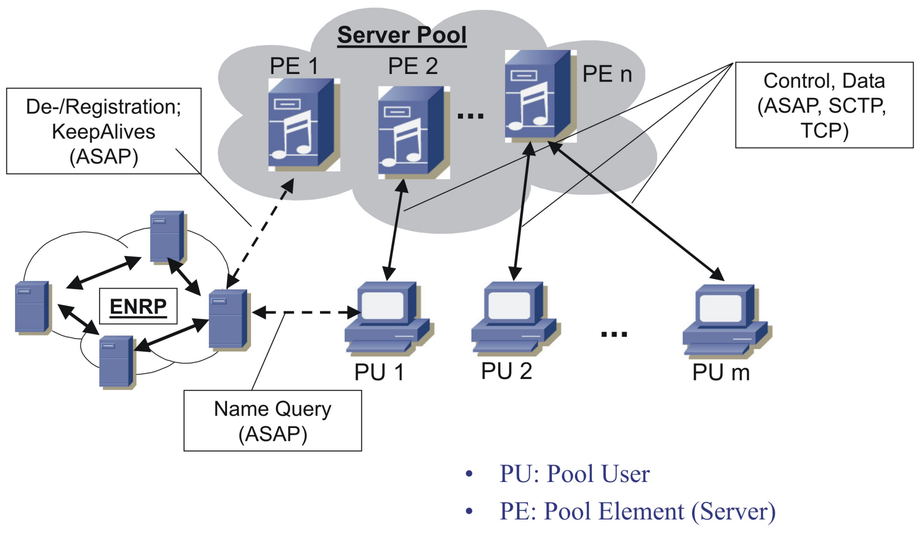

Server Pool
PE 1
PE 2
PE n
De-/Registration; KeepAlives (ASAP)
Control, Data (ASAP, SCTP, TCP)
ENRP
PU 1
PU 2
PU m
Name Query (ASAP)
• PU: Pool User
• PE: Pool Element (Server)

Figure 2.5: Architecture of Reliable Server Pooling

- Pool Elements (PEs): These represent the servers providing a service as elements of a pool. All elements within a pool provide the same service.
- Pool Users (PUs): The clients using the services being offered by a PE.
- Name Servers (NSs): These are special PEs that provide a name translation service. This service translates *pool handles* (byte vectors, with e.g. an ASCII string like “video pool”) to a set of transport addresses identifying active PEs. Moreover, NSs supervise PEs, and remove inactive PEs from their databases.

The protocol used between PEs and NSs, PUs and PEs and between PUs and NSs, is called the Aggregate Server Access Protocol (ASAP), which is currently defined in [77]. It provides registration, re-registration and de-registration of PEs with NSs, supervision of PEs using session layer keep-alive messages, and detection of NSs using server announcements via multicast (if available). For PUs it provides name translation via a NS, and a mechanism for session (re-)establishment with PEs (cf. sections 3.8 and 3.9 of [77]): In case of a PE failure, the ASAP protocol allows for transparent session failover of a PU to other PEs, where the service can be continued.

The protocol used by NSs to synchronise their name spaces and PE databases is called the Endpoint Name Resolution Protocol (ENRP) [76].

In this scenario, the communication relationships and the protocols used are as follows:

- Communication between PU and NS: This communication is used for performing name queries. The PU sends a pool handle to the NS and receives the necessary information for accessing a server in the server pool. This communication could use TCP or SCTP.

- Communication between PE and NS: This communication is used for registration and de-registration of the PE. The PE could be registered in two or more pools. This communication uses SCTP.

- Communication between PU and PE: This communication consists of two channels: control channel and data channel. The data channel is used for the service related traffic (upper layer data transmission), the control channel is used for RSerPool related traffic (to exchange RSerPool information). If both channels use different transport protocols (two TCP connections, or a combination of UDP with TCP or SCTP) they have to be tied together. The SCTP data chunk includes an 8 bit protocol payload identifier field, which indicates the message type contained in the data chunk. Based on this field, RSerPool uses an SCTP association and both data and control channels can use the same transport connection. This way, the receiver can recognise whether the data chunk belongs to a control or a data channel. The protocol payload identifier for ASAP and ENRP is standardised by Internet Assigned Numbers Authority (IANA).

- Communication between NS and NS: This connection is used to share knowledge about all server pools among all name servers in an operational scope.

- Communication between PE and PE: This is a special case of the communication between PU and PE. In this case the PU is also a PE in a server pool.

Security considerations

The document [79] analyzes the possible threats to the RSerPool architecture. For a secured communication among the above mentioned entities (namely NSs, PEs, and PUs), integrity protection and authorisation are desired security mechanisms for most communication relationships. For some of them, confidentiality may also be a desired property, but support for this is considered optional.

RSerPool strongly requires security functionality and it is mandatory to use the security protocols such as TLS or IPsec, because the following threats could occur in RSerPool:

- PE registration/de-registration flooding or spoofing: To avoid this ENRP servers have to authenticate the PE.

- PE registers with a malicious ENRP server: To avoid this PE has to authenticate the ENRP server. The first two threats, when taken together, result in mutual authentication of the ENRP server and the PE.
- Malicious ENRP server joins the ENRP server pool: To avoid this ENRP servers have to mutually authenticate.
- A PU communicates with a malicious ENRP server for name resolution: To avoid this the PU has to authenticate the ENRP server.
- Replay attack: To avoid this RSerPool needs a security protocol which provides protection from replay attacks.
- Corrupted data which causes a PU to have misinformation concerning a pool handle resolution: To avoid this RSerPool needs a security protocol which supports integrity protection.
- Eavesdropper snooping on name space information: To avoid this RSerPool needs a security protocol which supports data confidentiality.
- Flooding of Endpoint_Unreachable messages from the PU to ENRP server: To avoid this ASAP must control the number of endpoint unreachable messages transmitted from the PU to the ENRP server.
- Flooding of Endpoint_KeepAlive messages to the PE from the ENRP server: To avoid this ENRP server must control the number of Endpoint_KeepAlive messages to the PE.

To summarise, the threats listed above require security mechanisms which support authentication, integrity, data confidentiality, and protection from replay attacks. For RSerPool we need to authenticate the following:

- PU $\leftarrow$ ENRP Server (PU authenticates the ENRP server)
- PE $\leftrightarrow$ ENRP Server (mutual authentication)
- ENRP server $\leftrightarrow$ ENRP Server (mutual authentication)

RserPool does not define any new security mechanism for responding to the above threats. It will use the existing security protocols TLS and IPsec. Implementation of security mechanisms is explained in section 6.1 of [76] and [77], but these sections describe only using TLS, not IPSec. Using IPsec is still an open issue. The RSerPool Draft recommends that TLS must be implemented by ENRP servers, PEs and PUs, for reasons of interoperability. Implementations must support TLS with SCTP as described in RFC3436 [37] or TLS over TCP as described in RFC2246 [21]. When using TLS/SCTP we must ensure that RSerPool does not use any features of SCTP that are not available to an TLS/SCTP user.

2.2.3 Authentication, Authorization and Accounting (AAA)

The AAA working group of the IETF originally focused on the development of requirements for authentication, authorisation and accounting, as applied to network access. The AAA working group solicited submissions of protocols meeting these requirements, and evaluated the submissions. The AAA working group is now developing an IETF standards track protocol, based on the Diameter submission [14]. The AAA working group defined, and is also still working, on some RFC standards and internet drafts related to Diameter. A candidate protocol must meet the security requirements as documented in [15][16][24][29][31], and must be engineered and reviewed properly as developed and deployed. In the next section, I describe the Diameter Base Protocol, which is the most representative protocol of AAA.

Diameter Base Protocol

The Diameter Base Protocol is essentially an extension of the Remote Authentication Dial-in User Service (RADIUS) [60], which the AAA working group defined in RFC3588 [14]. The name ”Diameter” is really a joke that means RADIUS times two. Essentially, it means that the protocol functionalities are better than RADIUS. Diameter can also be used to authenticate and authorise users of Code Division Multiple Access (CDMA) wireless data services. Diameter is a more sophisticated protocol than RADIUS. Its base protocol is intended to provide an AAA framework for applications such as network access or IP mobility. Diameter is also intended to work in both local authentication, authorisation and accounting situations and also in roaming situations.

In the decade since AAA protocols were first introduced, the capabilities of Network Access Server (NAS) devices have increased substantially. As a result, while Diameter is a considerably more sophisticated protocol than RADIUS, it remains feasible to implement it within embedded devices, given improvements in processor speeds and the widespread availability of embedded IPsec and TLS implementations. The Diameter base protocol provides the minimum requirements needed for an AAA protocol, as required by [10]. The base protocol focuses on network access and accounting applications.

Diameter over SCTP

The Diameter base protocol requires reliable transport. The base Diameter protocol runs over TCP [58] or SCTP [75] transport protocols. The Diameter clients must support either TCP or SCTP and servers must support both. In future versions of the protocol specification, clients may have to support SCTP as well. When the client supports both TCP and SCTP, it will first try to establish an SCTP association. If this fails, it may fall back to TCP. In AAA, transport usage has several issues, for which SCTP is a better solution than TCP. For example, head of line blocking is one issue, which is avoided in SCTP but not in TCP.

More details about AAA transport profiles are defined in [11]. The following are guidelines for Diameter implementations that support SCTP:

- For interoperability: All Diameter nodes MUST be prepared to receive Diameter messages on any SCTP stream in the association.
- To prevent blocking: All Diameter nodes SHOULD utilise all SCTP streams available to the association to prevent head-of-line blocking.

In several scenarios the Diameter protocol uses agents. The agents are used in, for example, a complex network which has multiple authentication sources. In this case the agents can sort requests and forward them towards the correct destination or the agents can distribute administration of systems to a configurable group. More details about the role of Diameter agents are described in section 2.8 of [14]. The Diameter base protocol introduces the following four agents:

Relay Agent: Relay Agents relay messages between Diameter nodes. Every message has a routing information and it includes the Diameter application identifier and a server identifier. Relay Agents are inserting or removing routing information from messages.

Proxy Agent: Proxy Agents route messages like the Relay Agents mentioned above. But the Proxy Agents use the Diameter Routing Table and decide based on predefined policies. Proxy Agents don't need to support all Diameter applications.

Redirect Agent: Redirect Agents allow servers and clients to communicate directly returning the information necessary, for Diameter agents to communicate directly. The redirect agents do not modify messages and never relay requests.

Translation Agent: Translation Agents provide translation between Diameter and another AAA protocol, such as RADIUS.

Security considerations

The authentication of each connection and authorisation of sessions are very important parts of the Diameter protocol. Also because Diameter requires confidentiality, transmission level security must be used on each connection (TLS or IPsec or some kind of transmission-level security). Therefore, each connection is authenticated, replay and integrity protected and confidential on a per-packet basis. Diameter clients have to support IPsec, and may support TLS. Diameter servers have to support both TLS and IPsec. The Diameter protocol must not be used without any security mechanism (TLS or IPsec). In RFC3588 [14] it is suggested that IPsec can be used primarily at the edges and in intra-domain traffic. Pre-shared keys are used between a Network Access Server (NAS) or a local AAA proxy when the NAS devices don't support certificates. It is also suggested that inter-domain traffic would primarily use TLS. See section 13 of [14] for a more detailed explanation of the Diameter protocol security mechanism.

2.2.4 Session Initialization Protocol (SIP)

The Session Initialization Protocol (SIP) is defined by the SIP working group of IETF. The SIP working group first defined the SIP protocol [65] in 1999, and in 2002 defined a new edition of the SIP protocol in RFC3261 [65]. The new edition of the SIP the protocol added the use of SCTP in transporting SIP messages.

SIP allows integration/convergence of telephony services, IP-based voice calls and IP-based data transmission over an IP network. Thus, it can be used to create a unified infrastructure for both voice and data services. It is an efficient, and scalable IP-based signalling protocol to establish a communication session. SIP is a key component of VoIP technology. For voice calls in the pure-IP domain, SIP represents the signalling protocol used to establish a connection between two voice-enabled devices, whether they are SIP-enabled IP phones, adapters for analog phones, or PC-based voice communication software. So SIP is an integral part of end-to-end IP-based voice communications. SIP is the protocol which will enable voice networks to become multimedia networks.

SIP is an application layer control protocol that is used in multimedia sessions such as Internet telephony calls. The role of SIP is to establish, modify and terminate multimedia sessions. SIP can also invite participants to already existing sessions, such as multicast conferences. Media can be added to (and removed from) an existing session. SIP supports five facets of establishing and terminating multimedia communications:

- Determination of the communication system to be used.
- Determination of the user availability of the called party to engage in communications.

- Determination of the media and media parameters to be used.
- Establishment of session parameters at both called and calling party.
- Call handling including transfer and termination of sessions, modifying session parameters, and invoking services.

SIP can be used with other protocols to build a complete multimedia architecture. Typically, these architectures will include protocols such as:

- the Real-time Transport Protocol (RTP) [68] for transporting real-time data and providing Quality of Service (QoS) feedback;
- the RTP Control Protocol (RTCP) [68] for providing feedback on the quality of the data distribution and controlling sessions in multicasting;
- the Real-Time streaming protocol (RTSP) [69] for controlling delivery of streaming media;
- the Media Gateway Control Protocol (MEGACO) [20] for monitoring controlling gateways to the Public Switched Telephone Network (PSTN);
- the Session Description Protocol (SDP) [32] for describing multimedia sessions;
- and the Session Announcement Protocol (SAP) [33] to announce a multicast session.

However, the basic functionality and operation of SIP does not depend on any of these protocols. Since SIP messages and the sessions they establish can pass through entirely different networks, SIP cannot, and does not, provide any kind of network resource reservation capabilities. The nature of the services provided make security particularly important. SIP provides a suite of security services, which include denial-of-service prevention, authentication (both user to user and proxy to user), integrity protection, and encryption and privacy services.

SIP over SCTP

As already stated, SIP is an application layer protocol and works independently from the transport protocol used. It can run over both reliable and unreliable transport protocols such as UDP and TCP. Usually, SIP procedures are defined for transport over UDP and TCP. However, SCTP is also a suitable transport protocol for SIP, as it provides most of the features of UDP and TCP. There are advantages in using SCTP instead of TCP or UDP for SIP and these are described below.

Advantages of SIP over SCTP, compared to SIP over UDP

1. Fast retransmission: By using the selective acknowledgement (SACK) mechanism, SCTP [75] can quickly detect the loss of a packet. When the receiver detects segments received out of order it sends a new SACK for every incoming data packet. [1] As a consequence, the sender will resend lost messages quickly.

2. Congestion Control: SCTP provides congestion control over the association.

3. Transport layer fragmentation: SCTP performs transport layer fragmentation. IP fragments messages for UDP when the message is bigger than MTU. When one fragment of the UDP data is lost then the whole UDP packet is lost. In this case, SCTP needs to retransmit only the lost data fragment.

Advantages of SIP over SCTP, compared to SIP over TCP

1. Head-of-line blocking: By using independent streams and unordered delivery, SCTP avoids head-of-line blocking. This is very important for SIP, when multiple transactions are running over one TCP connection and one message of the particular transactions is lost, then all transactions will be stopped. With head-of-line blocking, other streams are not affected.

2. Multihoming: SCTP provides support for multihoming which improves fault tolerance. When a primary network (path) fails, SCTP can use another network (path) and continue to transport messages.

Security considerations

The SIP protocol requires fundamental security services to:

- preserve the confidentiality and integrity of messaging,

- to prevent replay attacks or message spoofing,

- to provide for the authentication and privacy of the participants in a session and to prevent denial-of-service attacks.

SIP message bodies each require the security services of confidentiality, integrity and authentication. Full encryption of messages provides the best security but SIP messages have some fields, which must be visible (i.e. unencrypted) to proxies. For example, request Uniform Resource Identifier (URI) and route. More details about security mechanisms of SIP are given in section 26 of [65].

[1] In the normal case it sends one SACK for every two incoming data packets.

2.3 Security related consequences for SCTP

All applications described above strongly require a security mechanism to protect sensitive data. The security consideration parts of SCTP application documents so far discuss the use of TLS or IPsec as security protocols. Both security solutions (TLS over SCTP and SCTP over IPsec) are already standardised, but there are currently no implementations supporting these security solutions. However, both TLS and IPsec don't support all SCTP features.

The limitations of the standardised security solutions are mainly due to the fact, that the primary design goal for them was to combine already existing protocols without modifications. A qualitative evaluation of the limitations of the standardised security solutions (see section 4) clearly shows that either modifications and extensions are necessary for the standard protocols or new solutions have to be found in order to fully support SCTP. For example, TLS cannot support some SCTP features (e.g. unordered delivery) because it was designed with reliable ordered TCP transport in mind.

Most problems of the standardised SCTP security solutions are related to SCTP features and extensions which distinguish SCTP from TCP and UDP. It is crucial that a security protocol used for SCTP applications supports all SCTP features and extensions. For example, in a complete multimedia architecture using SIP with RTP and RTCP, the partially reliable delivery extension (described in 3.1.4) of SCTP is used for realtime delivery.

In this project, I have chosen to define a new integrated solution which allows to fully exploit all potential optimisations without restrictions.

Chapter 3

Description of relevant protocols

3.1 Stream Control Transmission Protocol

The Stream Control Transmission Protocol (SCTP) is a message-oriented, general-purpose, reliable transport protocol which was defined by the IETF in [75] in October 2000. SCTP provides numerous advantages over UDP [57] and TCP [58]. SCTP is a reliable transport protocol as TCP, but its API [78] provides UDP-like functionalities. For instance, SCTP combines the message orientation of UDP with the sequencing and reliability of TCP. SCTP is a connection-oriented protocol and its connections are called ”associations”. An SCTP association can provide a 1-to-many (one connection to many applications) relationship unlike TCP which only provides a 1-to-1 relationship. Multiple applications using one SCTP association transport data in independent sequenced streams. A stream is a unidirectional logical channel transporting a sequence of messages. The messages are included in chunks. A chunk is a SCTP information unit. SCTP can bundle several chunks into one packet if the messages are short and can also fragment messages into several chunks if they are too big. One main SCTP advantage over TCP is that SCTP supports multiple IP addresses. That means that an SCTP endpoint, which has multiple IP addresses, can use multiple paths. A path is a uni-directional network route between associated SCTP endpoints.

3.1.1 Format of SCTP messages

SCTP endpoints communicate by exchanging protocol data units (PDUs) that contain a fixed size common header plus a possibly variable number of chunks. A typical PDU is shown in figure 3.1.

Chunks may either transport user data (data chunks) or control information (control

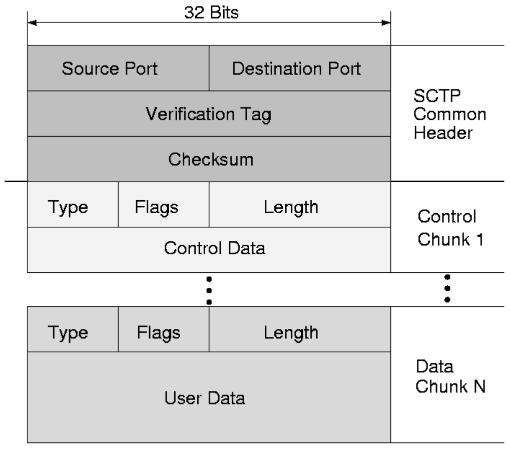


Figure 3.1: SCTP packet format with common header and chunks

chunks). Control chunks are used for establishing/releasing associations, selectively acknowledging data, supervising the status of network paths, reporting errors, or optional protocol extensions. When both control and data chunks are transported in a SCTP packet, control chunks are always placed before data chunks. Chunks may be bundled into an SCTP packet, which must remain smaller than the Maximum Transmission Unit (MTU) of the underlying network path.

Every SCTP packet begins with a common header, which contains two 16-bit port numbers (source and destination), a verification tag, and a CRC32-C checksum [1] [80]. The verification tag is a 32 bit random value that is exchanged between the endpoints when the association is set up. The checksum is 32 bits, (unlike in TCP, which only uses a 16 bit checksum). Upon receipt of an SCTP packet, an endpoint first verifies the checksum, and then drops the packet if the checksum is incorrect.

All chunks have a common chunk header and a chunk specific content, which depends on the chunk type. The chunk headers include an 8 bit chunk type field. At the time of writing, more than 200 values are still reserved for new chunk types. The chunk concept makes SCTP more flexible, because users can define new control chunks using the reserved values. Some SCTP control chunks have parameters which are mandatory or optional and they have a common format shown in figure 3.2. The length of parameters is variable and the values depend on the parameter type.

[1]The redundant information is the remainder of a polynomial division rather than a sum. However, this term is commonly used in the standards and will also be used throughout this thesis.

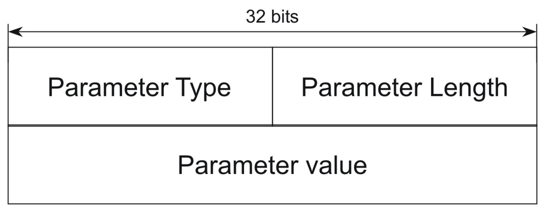


Figure 3.2: The format of a parameter

3.1.2 Main features of SCTP

In this section I will describe the main features of SCTP which distinguish SCTP from TCP, and in later chapters will relate to the security considerations.

Multi-Streaming

The multi-streaming mechanism separates and transmits user data on multiple SCTP streams. These streams are capable of independent, sequenced delivery. Message loss in a particular stream will only hinder delivery within that particular stream, and no others within the association. The figure 3.3 shows one example of multi-streaming.

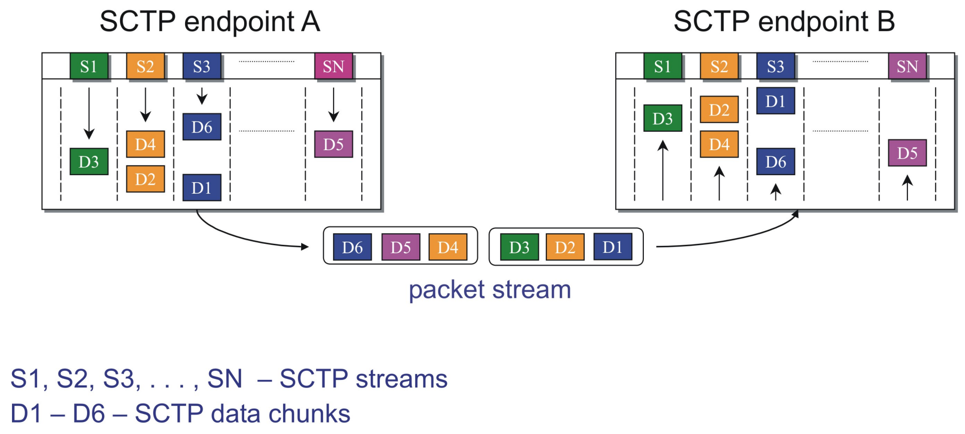

SCTP endpoint A
SCTP endpoint B
S1
S2
S3
SN
D1
D2
D3
D4
D5
D6
packet stream
S1, S2, S3, . . . , SN – SCTP streams
D1 – D6 – SCTP data chunks

Figure 3.3: Multiplexing of multiple message streams within one association

With multi-streaming, SCTP eliminates the unnecessary blocking that often occurs in TCP transmissions. In TCP, a stream is defined as a sequence of bytes that conform to strict in-sequence delivery. In-sequence delivery results in a major drawback known as head of the queue blocking, where messages within a stream are not allowed to bypass each other.

Flexible delivery

SCTP provides flexible delivery for messages, by using both ordered and unordered delivery methods. The ordered delivery means that the receiver transmits received messages to the user application in the same order in which they were submitted to the sender. Every data chunk includes a 32 bit Transmission Sequence Number (TSN) which helps to provide ordered delivery. Unordered delivery means that the receiver directly delivers a received message to the user application without reordering. The data chunk includes a flag which indicates whether the message requires ordering or not. The receiver checks this flag and directly sends the message to the user when the flag is set. Figure 3.4 shows an example of the ordered and unordered delivery.

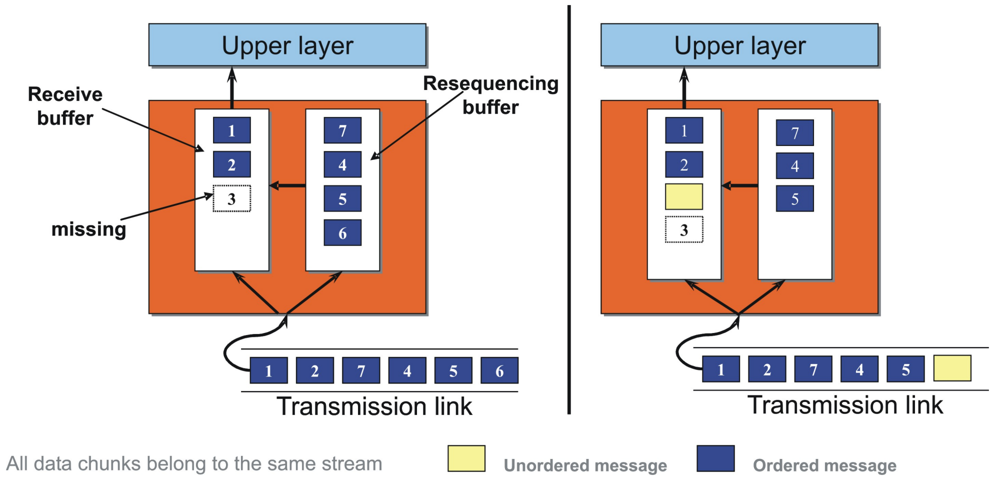


Figure 3.4: Flexible message delivery

Figure 3.4 shows ordered delivery on the left side and unordered delivery on the right side. In ordered delivery, if one data chunk is missing then the next chunks must wait for delivery to a user application in a resequencing buffer until the missing chunk is received. In our example, data chunk 3 is missing and data chunks 4 to 7 are waiting. However, in the unordered delivery case, the data chunk with unordered delivery flag set doesn't wait in the resequencing buffer, but is delivered directly to the receive buffer.

Bundling

SCTP uses bundling when assembling packets for transmission. Bundling means grouping chunks together into a single SCTP packet. This can include both control and data chunks. The benefit of this is an increase in bandwidth efficiency, due to a reduction of the overheads imposed by SCTP and IP headers, and by a reduction of the number of Selective Acknowledgments (SACKs) needed by a receiver (SACK chunk is described in section 3.2.6 of [66]). The bundling mechanism is defined in section 1.3.5 of [75]. SCTP has some rules for organising (bundling) the packet. These are:

- The common header is always located at the beginning of the packet.
- The bundling mechanism is optional. However, when bundling chunks into the packet, the mechanism must pay attention to MTU size. The packet size must be less than the MTU size of the path used.
- If the packet includes both control and data chunks, the control chunks are inserted into the packet after the common header, with the data chunks following.
- Some chunks (INIT, INIT-ACK and SHUTDOWN-COMPLETE) cannot be bundled. So, if the packet includes one of these chunks, SCTP must not insert other chunks into the packet.
- Data chunks must not be bundled with an ABORT chunk, only control chunks may be bundled with an ABORT chunk. If the packet includes control chunks and an ABORT chunk, then the control chunks must be inserted before the ABORT chunk.

User data fragmentation

SCTP fragments a user message if the size of the user message to be sent causes the outbound SCTP packet size to exceed the current path MTU. The data chunk has flags to indicate whether the chunk is carrying a complete message or a part of a fragmented message. In this case, it will also indicate whether it is carrying the beginning, middle or end part of the message. The chunks carry transmission sequence numbers to allow the message to be reassembled in the correct order.

Multi-Homing

An SCTP multi-homed endpoint can control and aggregate multiple IP addresses, unlike a TCP endpoint, which can only have a single IP address.

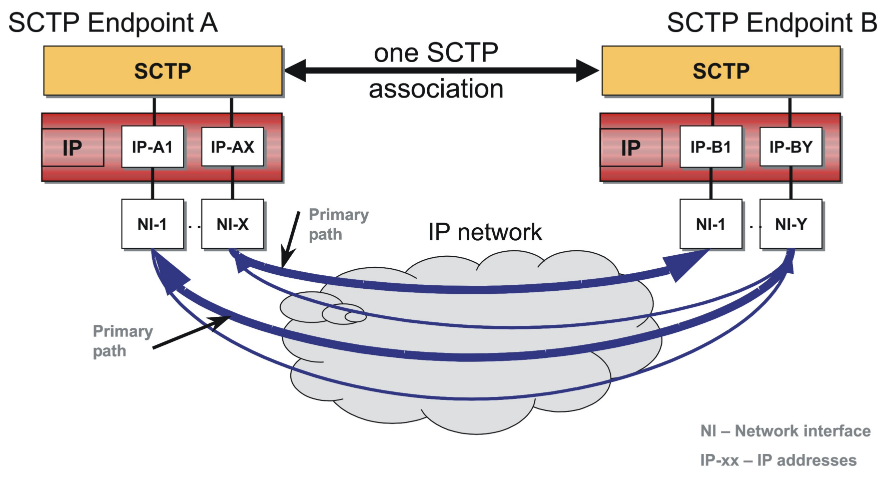


Figure 3.5: SCTP endpoints supporting multiple addresses

In figure 3.5, the endpoint A has X IP addresses and the endpoint B has Y IP addresses. During the initiation of an association, SCTP endpoints exchange lists of IP addresses. SCTP data delivery uses a path. A path is unidirectional network route between associated SCTP endpoints. Each endpoint can have multiple paths, one path is selected as the primary path (see figure 3.5) and carries the main load of the user data traffic. The primary paths from A to B and from B to A could be different. Other paths are only used for data retransmissions and HEARTBEAT chunks (HEARTBEAT chunk is described in section 3.2.7 of [66]). Should the primary path become unreachable, an endpoint may send data to another, active address, and report the failure to its user, who can subsequently choose a new primary path. Todays operating systems (e.g. Windows, Linux, FreeBSD ...) make routing decisions in the kernel space and they choose one route per destination. This means an application cannot, in general, predefine which specific route is to be used to reach the peer in a multi-homing scenario. In addition, all packets sent to the same destination address are delivered via the same route irrespective of their source address. Therefore, per default there are Y paths from A to B and X paths from B to A in the above scenario, i.e. there exist X+Y paths between A and B. However, an application can explicitly define source and destination address of a packet that it sends and a user can manually configure an extended routing table. In this case the operating system's routing decision follows the extended routing table and the packet is forwarded via the respective sending interface which has the source address given in the packet. Moreover, a receiver sends its reply to the source address where the packet came from. Thus, it is possible to create several routes per destination and by doing so, the number of paths used between A and B can be extended to X*Y.

Multi-homing has obvious advantages over single homed connections in case of network failures if configured correctly.

Retransmission of lost packets

The loss of data is a major problem in an IP network. This problem is also relevant for SCTP. But, with the help of the retransmission mechanism, SCTP solves this problem and provides a reliable service. The sender starts the retransmission timer (this timer is named T3) every time a DATA chunk is sent out and is waiting for a Selective Acknowledgement (SACK) chunk. The SACK chunk is sent from the receiver to acknowledge the received DATA chunks.

It includes a cumulative Transmission Sequence Number (TSN), zero or more Gap Ack Blocks and an indication of duplicated TSNs. The Cumulative TSN is the highest TSN received without gaps. The Gap Ack Blocks indicate received DATA chunks which have TSNs higher than the cumulative TSN. Figure 3.6 shows a simple example of data transmission between two endpoints with data chunk losses.

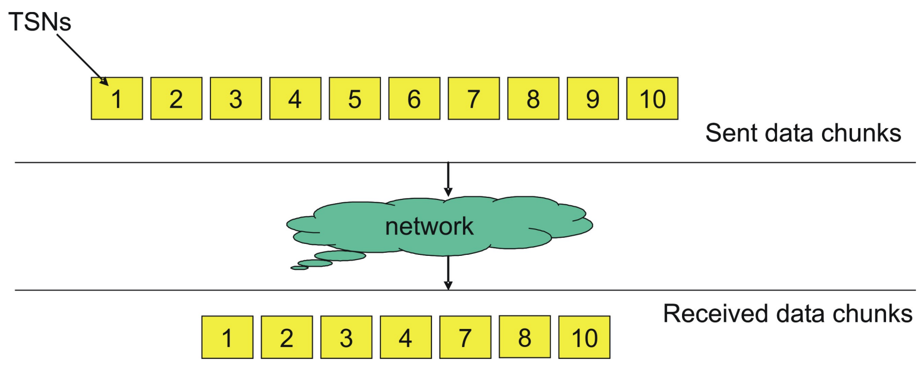


Figure 3.6: Data chunks sent between two endpoints

A Gap Ack Block indicates the start and end offset TSNs of received chunks relative to the current cumulative TSN. The received DATA chunk's TSN in a Gap Ack Block is calculated by following:

TSN = cumulative TSN + offset TSN

In our example the cumulative TSN is equal to 4 and the sender receives a SACK with two Gap Ack Blocks. The start TSN for the first Gap Ack Block is equal 3 and the corresponding end TSN is equal 4 such that this block acknowledges the chunks 7 and 8 (from 4+3=7 to 4+4=8) also indicating that chunks 5 and 6 are missing. The second Gap Ack Block start TSN is equal to 6 and its end TSN is also equal 6 thus acknowledging chunk 10 (4+6=10) and reporting chunk 9 as missing. When two or more DATA chunks which have same TSN are received then the SACK chunk contains this TSN and marks it as duplicated TSN.

If the SACK chunk doesn't arrive before the T3 timer expires then the sender resends the data. The T3 timer values are updated when a DATA chunk is retransmitted. The sender stops the T3 timer when all outstanding DATA chunks are acknowledged. After arrival of a SACK chunk, that advances the CTSNA, if there are still outstanding DATA chunks, then the sender restarts the T3 timer. For more details about handling retransmission timers, see page 151-158 of [66]. The sender marks a TSNwhich is missing in the received SACK chunk. If this missing TSN is not acknowledged by one of the next three SACK chunks then SCTP uses a fast retransmission mechanism. The receiver sends a SACK chunk after reception of two packets containing DATA chunks to reduce the overhead of a message. But in the fast retransmission mechanism the receiver sends a SACK chunk after every received data packet. More details about the fast retransmission mechanism are defined in section 7.2.4 of [75].

Flow and congestion control

SCTP data transmission is controlled by congestion and flow control functions (see [7]) which are similar to TCP. Before the initialisation of an association, both associating endpoints exchange the advertised receive window credit (a_rwnd) intial values. The a_rwnd is used for flow control and it is dynamically updated by the sender. The value of a_rwnd is used to indicate the amount of user data which is allowed to be transmitted to the peer.

One other parameter of flow control is the receiver window (rwnd). The rwnd indicates the receiver's available space in the inbound buffer. By using this parameter, the sender knows how much data to send. After initialisation rwnd is equal to the received a_rwnd. When receiving any SACK chunk, the sender calculates a new value for rwnd. Calculation of a new rwnd value is defined in section 6.2.1 of [75]. The congestion control function uses a congestion window (cwnd) and a slow start threshold (ssthresh) parameter. The parameters cwnd and ssthresh are per path, and not per association. The congestion control function includes a slow start algorithm and congestion avoidance algorithm methods. The slow start algorithm is used when ssthresh is larger than or equal to cwnd. In other cases, congestion avoidance is used. After an association initialisation cwnd is intialised to two MTU and ssthresh is initialised to an arbitrary value, but in most implementations it is equal rwnd.

The purpose of the slow start algorithm is to determine the available capacity of the unknown network. At the beginning of the transfer, the endpoints send small amounts of data. If there is available capacity, the sender can ramp up the amount of data transferred. The purpose of congestion avoidance is to increase cwnd by one MTU every round trip time (RTT). An endpoint reduces ssthresh and cwnd values when it detects packet losses, or if the T3 timer expires. The calculation of the new value is defined in section 7.2.3 of [75].

3.1.3 SCTP association management

This section will describe how SCTP sets up and tears down an association, and how it transfers user data over that association. Figure 3.7 shows the complete SCTP state diagram, which is described in [75].

Association setup

Before any data transfer can take place, two endpoints must successfully complete a ”four way handshake”. This is the process which initiates an association (similar to TCP’s ”three way handshake”).

The basic setup process is shown in Figure 3.8. The process becomes more complicated when an application restart is initiated, or when two endpoints attempt a simultaneous set up. These two scenarios, although infrequent, are possible, and SCTP is able to cope with them when they arise.

The initialisation begins when the endpoint A builds and sends an INIT chunk. Then the endpoint A moves from the Closed state to the Cookie-Wait state. There are many parameters which have to be included, and these are described in detail in [66]. Interestingly, the INIT and INIT-ACK chunks contain similar parameters to initiate some mechanisms on both sides. These include:

- An INITIATION tag. This is used for the verification of the common header, after the association is established. It protects the packet from blind attacks by allowing a receiver to identify a packet, and discard it if the verification tag is unknown.

- Input and Output stream numbers. A user application creates an input and output stream number at the start of the association (these may be different). These numbers remain constant throughout the lifetime of the association. The output stream number must be less than the input stream number – if not, the endpoint aborts the association.

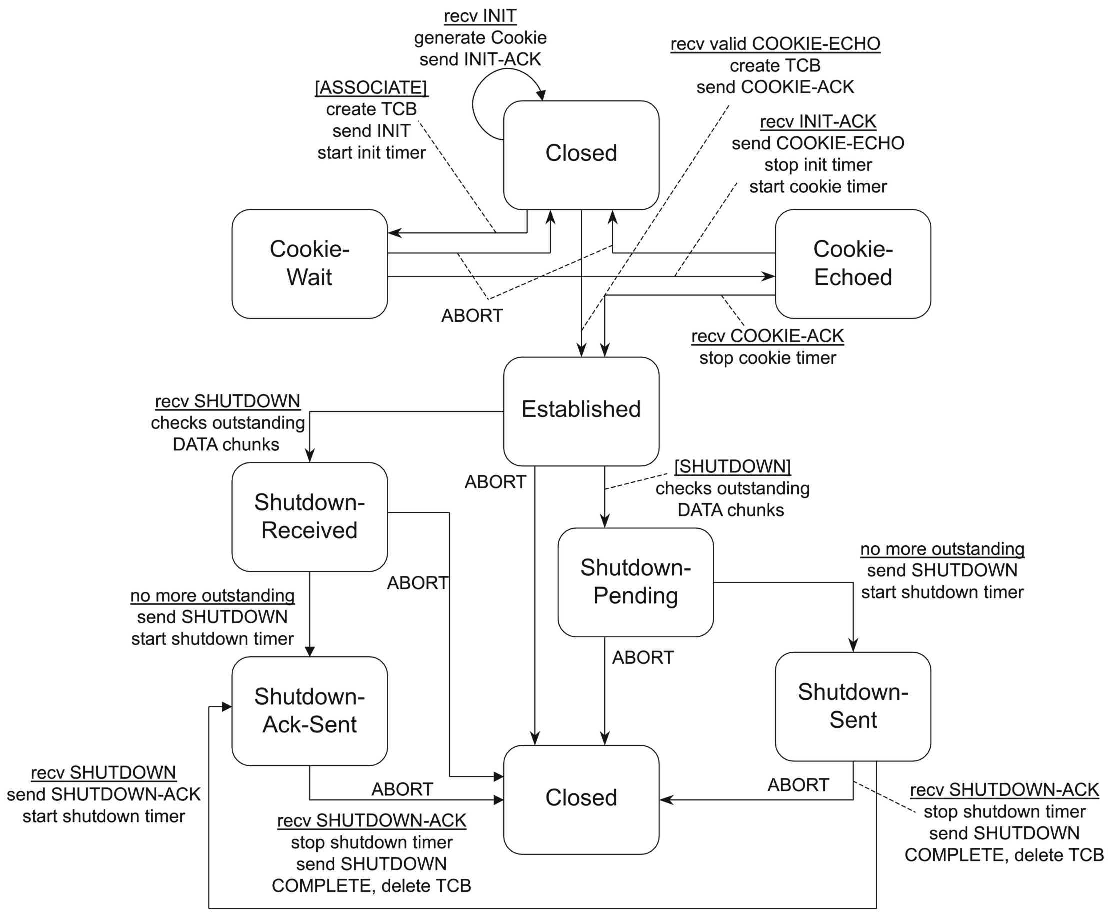


Figure 3.7: SCTP state machine

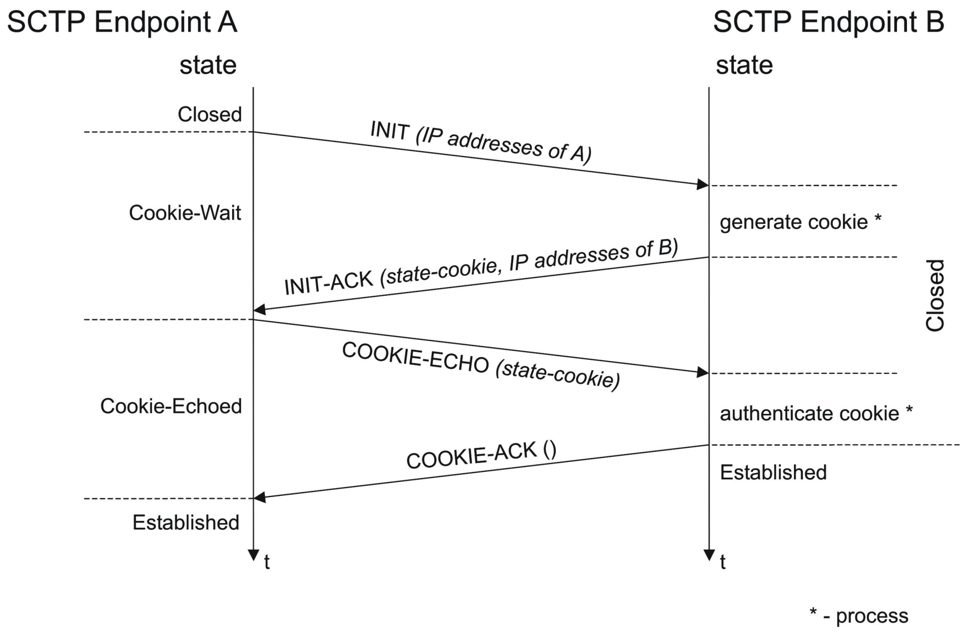


Figure 3.8: SCTP initialisation diagram

- Initial Transmission Sequence Number (TSN). The TSN provides a means for reliably transmitting user data. The initial value is set in here.
- Advertised Receiver Window Credit (a_rwnd) Field. This is the flow control window size, and indicates how much buffer space is being allocated to the association.

The next step is that the endpoint B sends an INIT-ACK chunk as the response to the INIT chunk. The INIT-ACK chunk contains similar values as included in the INIT chunk and also an important mandatory parameter, called state-cookie. This is ultimately intended for the endpoint B itself. The endpoint B generates a state-cookie, which includes a Keyed-Hashing value for Message Authentication (HMAC) [41]. The HMAC is created from a timestamp and all information that was included in the INIT and INIT-ACK chunks. The endpoint B generates a secret key when it starts up. This key is used for generating the HMAC securing ”state-cookies” which are used when new associations with this endpoint are established. The purpose of the state-cookie is that the endpoint B stores all association related information in the state-cookie and sends it back to A so it doesn’t have to save any of the association related information after sending back the INIT-ACK. Therefore, endpoint doesn’t have to allocate any resources for a pending association. This is preventing DoS attacks which often occur in TCP connection initialisation. The endpoint B remains in the Closed state.

After receiving an INIT-ACK chunk, the endpoint A sends a COOKIE-ECHO chunk. The COOKIE-ECHO chunk includes the state-cookie, which had been sent by the endpoint B. Now the endpoint A moves to the Cookie-Sent state.

The endpoint B chocks the state-cookie after receiving the
COOKIE-ECHO chunk by verifying wiht its own secret key that the cookie had originally been sent by endpoint B and that is has not been modified. If the state-cookie is valid, the endpoint B gets all association related information from state-cookie, creates an SCTP association and sends a COOKIE-ACK chunk to the endpoint A. This chunk includes only a chunk header. The sender can bundle DATA chunks with COOKIE-ACK chunk in one packet. Then the endpoint B moves to the Established state.

The endpoint A moves to the Established state when it receives the COOKIE-ACK chunk.

An attacker (man in the middle) can read the content of the cookie (sent in clear text) and use it for potential subsequent attacks. He can also completely block the communication, of course. However, due to the HMAC, he cannot modify the information, e.g. to hijack the association by forging the addresses.

Closing down an association

SCTP has two methods to close an association - a graceful shutdown and an abortive shutdown. The graceful shutdown of SCTP is similar to that of TCP, except that, unlike TCP, SCTP doesn't support a ”half-closed” state where one side is closed, and the other side is open and still able to send data.

Graceful Shutdown SCTP uses a three-way handshake for the graceful shutdown. The upper layer protocol of one endpoint indicates to the SCTP layer that it wishes to tear down the association. At this point, the SCTP protocol marks the association as closing and stops accepting data from the upper layer (see figure 3.9).

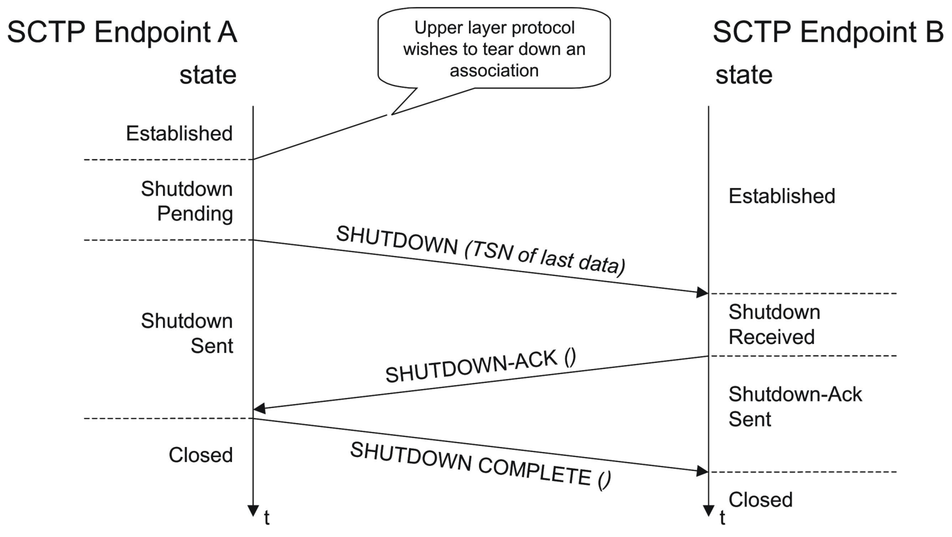


Figure 3.9: SCTP graceful shutdown diagram

If there is still data left to transmit, or unacknowledged transmitted data, it now enters a state of Shutdown-Pending, and will remain there until it has finished sending all data to the upper layer, or has received acknowledgements for previously sent, but unacknowledged, chunks. When these conditions are met, the endpoint sends a SHUTDOWN chunk to the other association endpoint, and it enters a Shutdown-Sent state. The SHUTDOWN chunk contains the TSN of the last data chunk sent, which allows the other endpoint to calculate how many more DATA chunks should be received.

If however, there is no data left to send, and there are no unacknowledged chunks, the endpoint will send a SHUTDOWN chunk immediately and go into the Shutdown-Sent state.

Upon receipt of a SHUTDOWN chunk, a receiver enters the Shutdown-Received state, and notifies the upper layer protocol that a shutdown has been received. It also stops accepting new messages from the upper layer.

Once all user messages have been acknowledged, the receiver will send a SHUTDOWN-ACK and enter the Shutdown-Ack-Sent state. It then starts a shutdown timer. If the timer expires, it should resend the SHUTDOWN-ACK.

When the sender receives the SHUTDOWN-ACK, it finally sends a SHUTDOWN-COMPLETE chunk and enters the Closed state.

When the SHUTDOWN-COMPLETE chunk is received, the receiver then stops its shutdown timer and also enters the Closed state to complete the graceful shutdown.

Abortive Shutdown This shutdown is used as an unreliable, best-effort attempt to notify endpoints that the association is stopping. It may be used either by the application or by SCTP. The application might use it if, for example, it fails and causes an ABORT to be sent, or when it wants communication to end immediately. SCTP might use it when, for example, it has insufficient resources or the operating system fails.

The mechanism is simple – if an endpoint wishes to close the association, it simply sends an ABORT chunk to its peer endpoint and enters the Closed state. The peer endpoint, upon receipt of the ABORT, verifies the information and then also moves to the Closed state. The ABORT chunk may contain information as to why the association is being suddenly closed, and if so, the receiving endpoint may pass this information up to its application layer.

If the peer endpoint doesn't receive the ABORT chunk, the sender doesn't know about it. So, it thinks the association is closed, whilst the peer endpoint thinks the association is open. In this situation, when the sender receives any message from the peer, it is treated as an out-of-the-blue (OOTB) packet (described in section 8.4 of RFC2960 [75]), and a new ABORT is sent to the peer.

Path and peer monitoring

Using HEARTBEAT chunks allows SCTP to periodically monitor the reachability of idle destination addresses. A destination address is considered idle when no chunk which can be used to update the round trip time (RTT) measurement of the address has been sent to the address within the current heartbeat period of that address. The period is implementation-specific.

A HEARTBEAT chunk contains a chunk header and sender-specific Heartbeat Info parameters. The sender-specific Heartbeat Info field should normally include information about the sender's current time at which the HEARTBEAT chunk is sent, and the 0destination transport address to which the HEARTBEAT chunk is sent. The receiver returns a HEARTBEAT-ACK chunk in response. The receiver copies the sender-specific Heartbeat Info field of the HEARTBEAT chunk to the HEARTBEAT-ACK chunk. When the sender receives the HEARTBEAT-ACK chunk it calculates the RTT. If the sender doesn't receive a HEARTBEAT-ACK chunk, then it determines that the destination address is not available. The user can turn off and on sending the HEARTBEAT.

In multihoming, if the primary path fails another available path is selected as the new primary path and the communication continues without interruption. All data lost due to path failure is retransmitted along the new primary path. More failover scenarios are explained and evaluated in [8].

Path Maximum Transmission Unit (PMTU) discovery

For PMTU, SCTP uses the same MTU as TCP, described in RFC1191 [47]. This defines the maximum number of bytes of an IP datagram that can be transferred in a single unit over an IP network. If a datagram exceeds this value, it is either dropped, or fragmented depending on the setting of a ”don't fragment” bit in the IP header.

PMTU Discovery is a mechanism used by an endpoint to discover the current PMTU of a path. Since this value may be specific for a certain path, the mechanism has to be performed for each path in a multihoming case.

3.1.4 SCTP extensions

To date, the Transport working group (TSVWG) of the IETF has defined two new SCTP extensions which are explained in more detail in the following sections. These extensions further distinguish SCTP from TCP and they are critical points for standardised SCTP security solutions, as will be explained in chapter 4.3.

Partially reliable delivery (PR-SCTP)

If this extension [73] is supported by both sides of an SCTP association, it can be used to provide a partially reliable transport service over an SCTP association. The partially reliable delivery means that some transported messages can be lost without or with only limited retransmission. This may be useful, for example, in real time audio or video transmission, when retransmissions would arrive too late, anyway.

This extension defines a new control chunk and a new parameter. The new Forward-TSN-Supported parameter consists of only a common parameter header, and it indicates the endpoint is supporting the PR-SCTP extension. This parameter is used only with INIT and INIT-ACK chunks. The PR-SCTP extension works when both associated endpoints support this extension.

The PR-SCTP extension defines the new Forward Cumulative TSN (FORWARD-TSN) control chunk. The FORWARD-TSN chunk includes the new cumulative TSN. One endpoint sends the FORWARD-TSN control chunk to the other endpoint, which then updates the SACK's cumulative TSN. Any outstanding DATA chunks with a TSN less than the SACK's cumulative TSN, can be dropped. SCTP does not retransmit these dropped messages.

Dynamic address reconfiguration

This is a new SCTP extension [74], which allows an endpoint to add or drop an IP address and set a new primary address on existing associations. This is particularly useful in mobile networking, where a mobile endpoint is frequently changing its position. For example, when a mobile endpoint leaves a network area and enters a new network area, it is able to add the new address and to remove the old address on the existing association, without terminating the association [9]. Both associated endpoints can send a request to add or to drop an IP address and set a new primary address. The endpoint accepts the dynamic address reconfiguration request coming from the other endpoint, without any conditions.

This extension is defined in [74] and it defines two new control chunks and 6 new parameters. The new Address Configuration Change (ASCONF) chunk requests a change of configuration and it must be acknowledged. The ASCONF chunk consists of a chunk header, a serial number and parameters. The 32 bit serial number is used for acknowledgement and is returned within the Address Configuration Acknowledgment (ASCONF-ACK) chunk. The purpose of the new ASCONF-ACK chunk is to acknowledge the ASCONF chunk and its parameters. The parameters can be divided into two groups (apart from the adaptation layer indication parameter). The first group requests and includes add or drop IP address and sets a new primary address parameter. The second group responds to request parameters and includes error cause indication and success indication parameters. Each sent request parameter has to be answered. Every request parameter includes a 32 bit correlation ID to determine the parameter and this correlation ID is returned within the response parameter. The adaptation layer indication parameter is dedicated for the communication of peer upper layer protocols.

3.1.5 Qualitative comparison of TCP, UDP and SCTP

TCP limitations

TCP suffers from a type of head-of-line blocking which makes it unsuitable for telephony. This is because telephony requires reliable delivery, but not necessarily sequenced delivery. The nature of TCP is that it provides guaranteed, ordered delivery: any missing data blocks delay delivery of incoming data, until the missing data has been retransmitted and safely delivered. This can introduce a delay that doesn't conform to the delay requirements of telephony signalling traffic.

Telephony signalling is message oriented, whilst TCP is byte oriented. This means that applications would need to add their own message boundary markers, and also have to ensure that messages are delivered in a timely fashion, if necessary using the ”push” facility.

TCP doesn't support multi-homed IP hosts. Without link or path redundancy, the transmission service can be too prone to failures to meet the high availability requirements of telephony signalling.

The security of TCP is not good enough to meet the high requirements of a telephony service. The main focus is usually placed on resistance to denial of service attacks, as TCP is fairly vulnerable to some denial of service attacks, such as SYN flooding ([25] p.9).

UDP limitations

UDP is a unreliable protocol, i.e. it doesn't provide guarantees that the data transmitted will be delivered to the destination, or if delivered, that it will be in the correct sequence. UDP also has no feedback mechanism which means that it is unable to detect the state of the network, and therefore cannot respond to network congestion by throttling back on transmission. The continued transmission of data into an already congested network means that the network may become more congested, and hence lead to data loss. This then gives another level of unreliability to the UDP service and makes it unsuitable for telephony signalling, because it cannot meet the very strict reliability requirements.

One approach is to place more functionality into the application in order to overcome the limitations of UDP. However, this is not generally considered a suitable solution due to the added complexity this would create for the application. If placed in the application, the complexity has to be added on a per application basis and is not generally usable which is a clearly suboptimal solution in an environment with many applications.

The advantages of SCTP over TCP and UDP

SCTP has two major advantages over TCP and UDP:

1. The support for multi-homed hosts.
   This allows a single SCTP association to achieve link/path redundancy, by running over multiple links/paths. By using this, fast failover can be achieved with barely any interruption of the data transfer.

2. The support for multiple streams in a single SCTP association.
   This overcomes the head of the line blocking problem of TCP. Message ordering is only enforced within each subflow, and thus there is no chance that messages can block across subflows.

The other characteristics of SCTP and its enhancements over TCP and UDP are summarised in Table 3.1.

Protocol Feature	SCTP	TCP	UDP
State required at each endpoint	yes	yes	no
Reliable data transfer	yes	yes	no
Congestion control and avoidance	yes	yes	no
Message boundary conservation	yes	no	yes
Path MTU discovery and message fragmentation	yes	yes	no
Message bundling	yes	yes	no
Multihomed host support	yes	no	no
Multi-stream support	yes	no	no
Unordered data delivery	yes	no	yes
Security cookie against SYN flood attack	yes	no	no
Built-in heartbeat	yes	no	no

Table 3.1: Feature comparison of SCTP, TCP and UDP

3.2 Standard security protocols

In this section I will describe the existing standard security protocols which can be used for secure transport with SCTP.

3.2.1 Secure Socket Layer and Transport Layer Security

The Transport Layer Security protocol (TLS) provides privacy and data integrity between two communicating applications. In this section, the background of its development is described. However, in order to do that, we must first talk about the Secure Socket Layer protocol (SSL).

Secure Socket Layer (SSL)

The motivation for the development of SSL was to provide secure electronic commerce and other Web transactions in the Internet. Before the development of SSL, web commerce transactions were insecure with sensitive data such as credit card numbers and personal information being insecurely sent over an unknown path, perhaps from one side of the world to the other. Clearly, this is not an environment wherein e-commerce would be expected to flourish.

Netscape Communications were thinking about these types of problems when developing their first web browser. They designed SSL to address security problems. The first and second versions, SSL 1.0, and SSL 2.0 were completed five months apart in 1994. Version 1.0 was not released, and version 2.0 was released with Netscape Navigator. SSL 2.0 was weak and easy to break. It used the same key for encryption and authentication, and the key length was a maximum of 40 bits. In 1995, Microsoft developed Private Communication Technology (PCT) as a minor enhancement to address the weaknesses of SSL 2.0. PCT was also incorporated into SSL 3.0. PCT has, in general, better security properties than SSL. Whilst the development of SSL was open, and the web community contributed significantly to the development, the protocol actually belonged to Netscape, (they hold a US patent, patent number 5657390 on it [18]). In 1996, the IETF took over the responsibility for further development of SSL. At this point it was renamed to Transport Layer Security protocol (TLS) and the first version was released in January 1999 [21]. The development can be seen in Figure 3.10

TLS is, essentially, another version of SSL. There are some differences, but they are fewer than, for example, the differences between SSL 3.0 and SSL 2.0 [83].

Transport Layer Security (TLS) protocol overview

Although the original application of SSL was for web transactions, TLS is a general protocol suitable for securing a wide variety of network traffic, e.g. file transfer, remote object access, email, remote terminal and directory access.

TLS is a suite of protocols: the Handshake protocol, the Record Layer protocol, the Alert protocol and the Change Cipher protocol. These are organised as shown in Figure 3.11. The Record Layer protocol accepts messages from the higher layer protocols, frames them as necessary, and passes them to the transport layer protocol (e.g. TCP) for transmission. The TLS protocol requires a strictly ordered delivery of messages.

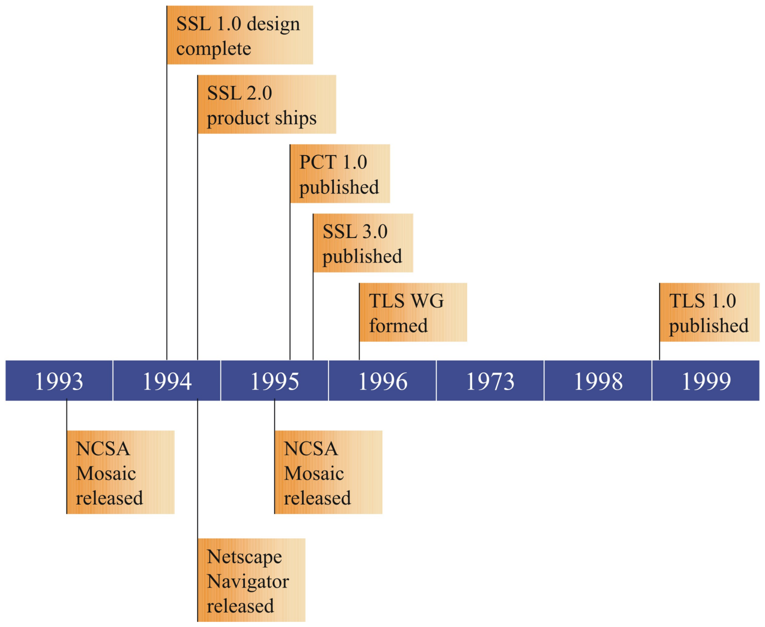


Figure 3.10: The development of SSL

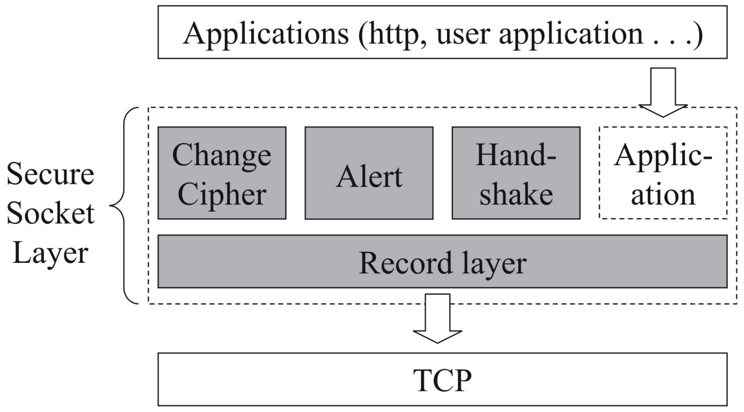


Figure 3.11: The organisation of the TLS protocols

Record Layer Protocol. This protocol provides the format for the other protocols to encapsulate their data (as shown in Figure 3.11). It secures application data, using the keys created during the Handshake protocol. It is responsible for securing application data and verifying its integrity and origin. It manages the following:

- Fragmentation of outgoing messages into manageable blocks when necessary (maximum block size is 16384 bytes), and reassembly of incoming messages.
- Compression of outgoing blocks and decompression of incoming blocks (optional).
- Application of a message authentication code (MAC) to outgoing messages, and verification of incoming messages, using the MAC.
- Encryption of outgoing messages and decryption of incoming messages.

When the protocol processing is finished, the outgoing encrypted data is passed down to the transport layer for onward transmission.

Handshake Protocol. This protocol is used to set up a connection between two applications, and it must be completed before any data can be transferred. It uses the Record Layer protocol to encapsulate its messages. Multiple handshake messages may be encapsulated into a single Record Layer message.

The Handshake protocol provides negotiation of cryptographic keys and algorithms, which are used for data encryption and authentication. The message sequence (defined in [21]) is described in Figure 3.12.

The protocol has a number of messages. Each handshake message has a single byte which defines the type of message. The Handshake protocol allows a client and server to start a new connection, or to resume a previous session. For a new connection, new security parameters are negotiated, whereas for a resumed connection, an abbreviated handshake is used.

The client sends a ClientHello message when it first connects to a server. The ClientHello message includes a cipher suite list, compression methods and protocol version. This is the usual opening message in a handshake negotiation. After receiving the ClientHello message the server sends a ServerHello message in response. The ServerHello message includes selected cipher suite, selected compression method and a protocol version. If the server does not accept the negotiation, it does not send this message. After sending the ServerHello message the server sends the messages described below:

- Certificate message: This message contains an X.509 certificate. The certificate authenticates the entities.

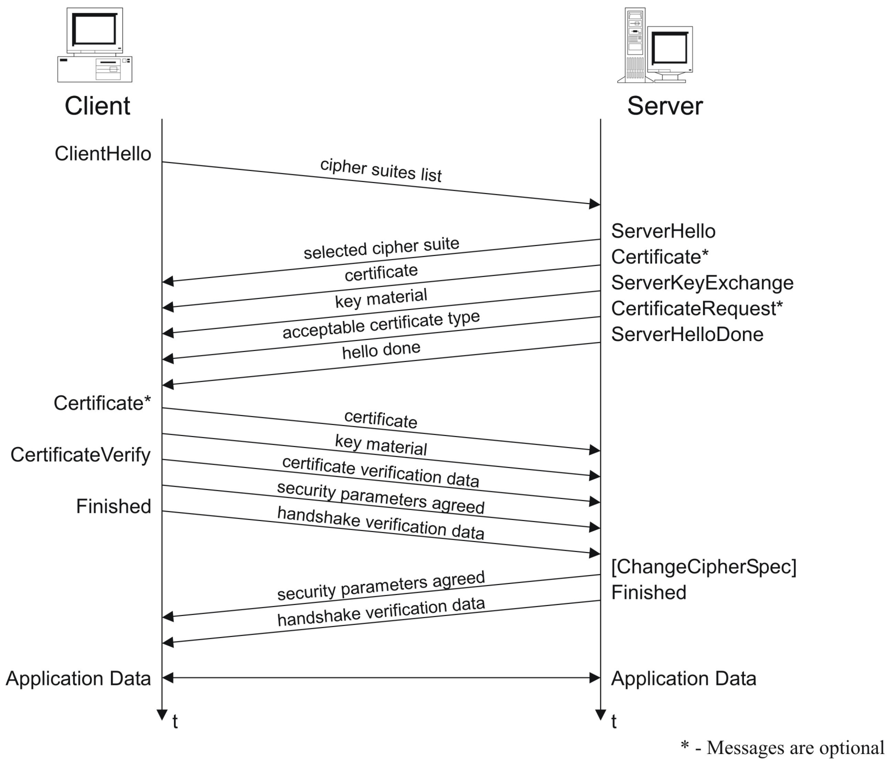


Figure 3.12: TLS handshake

- ServerKeyExchange message: This message conveys cryptographic information to allow the client to communicate the premaster secret and the cipher suite selected from cipher suites list in the ClientHello message.

- CertificateRequest: A non-anonymous server can optionally request a certificate from the client, if appropriate for the selected cipher suite.

- ServerHelloDone message: This message means that the server has finished sending messages to support the key exchange, and the client can proceed with its phase of the key exchange.

After receiving all of these messages, the client sends a Certificate message if the server has requested a certificate. If the server doesn't request a certificate, then the client sends a ClientKeyExchange message including key material, which is self-generated. The client sends several more messages to the server, which are described below:

- CertificateVerify: This message is used to provide explicit verification of a client certificate. This message is only sent following a client certificate that has signing capability.
- Finished message: This message is the final handshake message and indicates that the negotiation is finished and that the connection is protected with the new negotiated algorithms, keys and secrets. Both server and client verify the Finished message individually and they validate the negotiation if no errors are detected.

Finally, the server sends ChangeCipherSpec and Finished messages to the client. Now both sides are ready to send secure data.

TLS has one more message named HelloRequest, which is not described above. This message is sent by the server, when it wants to restart a negotiation. It is not often used, but it provides an extra level of security for a server. For example, if a connection has been open for an unacceptably long period, the server may use HelloRequest to force a new negotiation of keys.

Alert Protocol. TLS uses this protocol to signal an error or caution condition to the other endpoint in the communication. The Alert protocol defines several specific errors. Fatal Alerts indicate that there are significant problems with the communication, and require that both parties terminate the connection. Warning Alerts are not quite as drastic, and in this case, a system may allow the connection to continue.

ChangeCipherSpec Protocol. This is the simplest protocol in TLS, with only one message. The ChangeCipherSpec message is sent by both the client and server to notify the receiving party that subsequent records will be protected under the newly negotiated CipherSpec and keys.

TLS features

The main features of TLS are:

- Application independent protocol: As TLS is located at the upper boundary of the transport layer, it is an application independent protocol which is easy to use in combination with many different applications.
- Data authentication: TLS provides data origin authentication. This is a very important function in most application scenarios.

- Data confidentiality: TLS provides data confidentiality (encryption) in addition to authentication if required.

- Flexible encryption and authentication: This is a very useful feature for users. The TLS protocol is flexible and allows users to choose the encryption and authentication algorithms as required. TLS has a cipher suite with NULL encryption which allows a user to use only authentication without encryption.

- Extensible cipher suites: A user can add new cipher suites to TLS.

TLS also has some disadvantages. For example, it only protects application data and cannot protect information of protocols running below TLS. This means that, for example, the TCP header isn't protected and could be forged and data could be sent to false destinations or users.

Another disadvantage occurs when two connected endpoints use many TLS connections. Every TLS connection requires an independent handshake, and this is an expensive process in terms of computation effort.

TLS is unable to use UDP because it requires a reliable transport protocol.

TLS cryptographic functions and cipher suites

TLS provides the following cryptographic functions: key exchange, digital signatures, symmetric key encryption, public key encryption and message authentication. The key exchange, digital signing and public key encryption functions are used in the handshake operations used to manage TLS connections. Symmetric key encryption and message authentication are used for the actual data transmission. The goal of the key exchange function is to create a master secret key for the negotiating endpoints by exchanging security parameters. This exchange is protected by using digital signatures. Both key exchange and digital signing functions use public key encryption. Public key encryption is too computationally expensive and too slow for real-time transmission of large data volumes. Therefore, symmetric key encryption is used for a data transmission. Message authentication can be provided in combination with encryption or without encryption. The purpose of the message authentication function is to authenticate the origin of a message and to protect the message against modifications on the way to the receiver.

TLS flexibly allows to use and to combine proven and well known cryptographic algorithms. The message authentication function, e.g., uses the SHA or MD5 one way hash algorithms. Table 3.2 gives an overview on the TLS cryptographic functions and the cryptographic algorithms used.

TLS cryptographic functions	Cryptographic algorithms used
Key exchange	RSA and DH
Digital signing	DSS and RSA
Public key encryption	DH, DSS and RSA
Symmetric encryption	RC2, RC4, DES, 3DES and AES
Message authentication	HMAC, SHA, MD5

Keywords: RSA - Ramir, Shamir [6], DH - Diffie-Hellmann key exchange algorithm [5], DSS - Digital Signature Standard [52], RC2 - RC2 encryption [63], RC4 - RC4 encryption algorithm [61], DES - Digital Encryption Standard [49], 3DES - Triple DES [50], AES - Advanced Encryption Standard [53], HMAC - Keyed-Hashing for Message Authentication [41], SHA - Secure Hash Standard [51], MD5 - The MD5 Message Digest Algorithm [62]

Table 3.2: Cryptographic algorithms used in TLS cryptographic functions

Specific combinations of key exchange, digital signature, symmetric key encryption and message authentication algorithms used in TLS are called ”cipher suites”. Figure 3.13 explains how TLS cipher suites are specified.

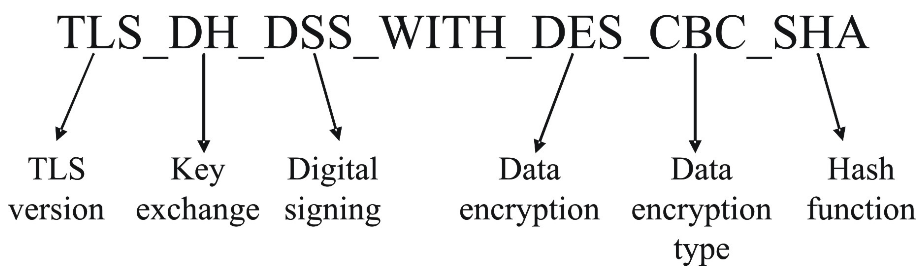


Figure 3.13: TLS cipher suite specifications

All cipher suites used for standard TLS are defined in Appendix C of [21] except for the cipher suites applying AES. Using AES in TLS was defined later and the respective cipher suites are defined in [17]. For a TLS implementation it is mandatory to implement the TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA cipher suite, support for other cipher suites is optional. TLS has been defined such that it can easily be extended by additional, new cipher suites.

SSL and TLS implementations

In this section, we give a short description of existing SSL and TLS implementations because all of them have been evaluated and one of them had to be selected as basis of our implementation. The implementations are:

1. OpenSSL: The OpenSSL Project is a collaborative effort to develop a commercial-grade, robust, full-featured, and Open Source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols as well as a full-strength general purpose cryptography library. The project is managed by a worldwide community of volunteers that use the Internet to communicate, plan, and develop the OpenSSL toolkit and its related documentation [23].

2. GNUTLS: This is a project that aims to develop a library which provides a secure layer, over a reliable transport layer. Currently the GnuTLS library implements the standards proposed by the IETF's TLS working group [26].

3. PureTLS: This is a free Java-only implementation of SSLv3 and TLSv1 (RFC2246) protocols. PureTLS was developed by Eric Rescorla for Claymore Systems, Inc. but is being distributed for free because they believe that basic network security is a public good and should be a commodity. PureTLS is licensed under a Berkeley-style license, which basically means that you can do anything you want with it, provided that you give credit to the authors and observe their copyrights [59].

4. Cryptlib: This is a security toolkit which includes TLS and SSL. At its lowest level, cryptlib provides a transparent and consistent interface to a number of widely-used encryption algorithms and systems. At an intermediate level, it provides powerful and easy-to-use digital signature and encryption key management routines. At the highest level, it provides implementations of complete security services such as S/MIME secure enveloping, SSL/TLS and ssh secure sessions, and other security operations such as secure timestamping [1].

5. TLS Lite: This is a free python library that implements SSL 3.0 and TLS 1.0. TLS Lite supports non-traditional authentication methods such as SRP, shared keys, and cryptoIDs in addition to X.509 certificates. TLS Lite is pure Python, however it can access OpenSSL, cryptlib, pycrypto, and GMPY for faster crypto operations. TLS Lite integrates with httplib, xmlrpclib, SocketServer, asyncore, and Twisted [55].

6. yaSSL: Yet another implementation of the Secure Sockets Layer (yaSSL) employs the dual licensing model, like MySQL, so it is available under GPL and commercial licenses. It provides SSL version 3 and TLS version 1 (client and server), OpenSSL

compatibility layer, MySQL integration, crypto algorithms (MD5, SHA-1, HMAC, DES, 3DES, RC4, RSA, DSS, and DH), simple API, interchangeable crypto and certificate libraries [4].

After evaluating these options, I have chosen OpenSSL as a basis for my implementation work, because OpenSSL is widely used and tested and provides rich functionality and a mature implementation.

3.2.2 IPsec

The security architecture for IP (IPsec) provides security services at the network layer. These security services are mainly data origin authentication and confidentiality. It protects the path between a pair of hosts requesting security. IPsec is a vast and complicated security protocol. The IPsec working group in the IETF is currently working on a definition of the IPsec protocol, its extensions and related protocols. At the time of writing, there are a number of IPsec related RFCs in existence [34][38][39][40][45][54][56][82].

The RFCs define various aspects of the IPSec architecture, e.g. key management, base protocols and the mandatory algorithms to implement for the base protocols. IPsec is a suite of specifications defining:

- Security Architecture for the Internet Protocol [40].
- IP Authentication header (AH) [38].
- IP Encapsulating Security Payload (ESP) [39].
- The Internet key exchange (IKE) [34].
- Internet Security Association and Key Management Protocol (ISAKMP) [45].
- The Internet IP Security Domain of Interpretation for ISAKMP (DOI) [56].
- IP payload compression (IPcomp) [70].
- IP Security Document Roadmap [82].
- The OAKLEY Key Determination Protocol [54].

The next sections will give a short explanation about the most important aspects of these protocols. For a more detailed description, the reader is directed to the relevant RFCs.

3.2.3 IPsec protocols

Because IPsec is a suite of definitions and protocols, it is important to understand how these protocols interact with each other and how these protocols are tied together to implement the capabilities described by the IPsec architecture. This is shown in figure 3.14 which has been reproduced from the IP Security Document Roadmap [82].

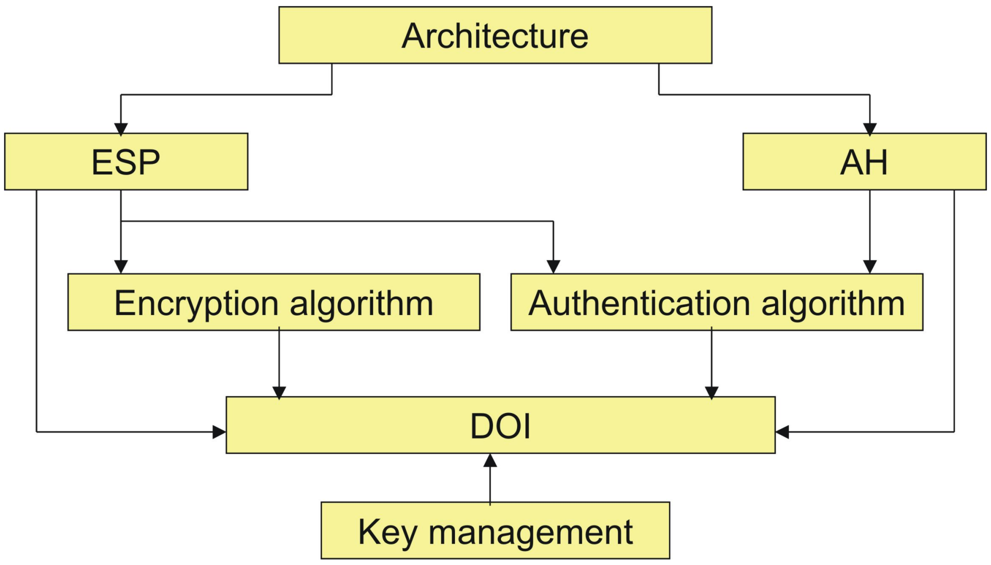


Figure 3.14: IPsec document roadmap

The IPsec architecture document defines a set of security services and the method by which these services can be employed in the IP environment. The blocks shown in figure 3.14 have following roles. ESP provides privacy and protects against modification and AH provides only protection against modification. The encryption and authentication algorithms define the transformation applied to the data to secure it. The transformation includes the algorithm, the key sizes and how they are derived, the transformation process, and any algorithm-specific information. The IKE protocol belongs to key management block and generates keys for the IPsec protocols. Any protocol can use the IKE protocol to negotiate keys. The parameters that are negotiated are documented in IPsec Domain of Interpretation (DOI) [56].

Security Architecture for the Internet Protocol

RFC 2401 [40] describes the architecture for IPsec systems [40]. The goal of the architecture is to provide various security services in both the IPv4 and IPv6 environments. This document defines important security components, e.g., the security association (SA), the security parameter index (SPI), the security association database (SAD) and the security policy database (SPD).

The term SA describes a unidirectional association characterised by the tuple <SPI, destination address, security protocol>. The SPI is an index which is needed if different SAs exist per address, the security protocol may be either AH or ESP. The set of all SAs is kept in the so-called security association database (SAD). Both inbound and outbound processing use an SAD, which may be different in each case. The SAD, therefore, can serve different security services for inbound and outbound processing.

Each IP endpoint implementing IPsec maintains another database, the security policy database (SPD), which maps inbound or outbound traffic to a SA. The SPD identifies traffic according to its characteristics such as source or destination addresses, transport protocols used, or source or destination ports. The SPD contains an ordered list of policy entries. Each policy entry is keyed by one or more selectors that define the set of IP traffic encompassed by this policy entry.

The IPsec protocol has two operation modes: transport mode and tunnel mode. In transport mode, only the payload of the message is secured. In tunnel mode, the whole IP packet is secured and encapsulated in a new IP packet.

In transport mode, IPsec may use either AH or ESP or both. In tunnel mode, IPsec uses either, but not both AH and ESP. For more details, read [40].

Authentication Header protocol

The goal of the Authentication Header (AH) is provide connectionless integrity and data origin authentication for IP datagrams and protection against replays [38]. The AH authenticates not only data packets, but also the IP header. This is an advantage over the ESP, because the ESP does not protect the IP header. In some cases, the IP header is changed during the transmission over the network. In this case, it is not possible for the receiver to know the sender, and the packet cannot be protected by the AH. The AH may be used in both tunnel and transport IPsec mode. In transport mode, the AH is inserted after the IP header and before an upper layer protocol (TCP, UDP, ICMP, SCTP, etc). In tunnel mode, the AH is inserted after the new IP header and before the original IP header. When AH and ESP are applied together, the AH is located first.

Encapsulation Security Payload protocol

The Encapsulation Security Payload (ESP) provides a security service between two entities [39]. This security service includes confidentiality (encryption), data origin authentication and an antireplay service. Unlike the AH, the ESP data origin authentication does not authenticate the IP header. In transport mode, the ESP header is inserted after the IP header and before the upper layer protocol header. In tunnel mode, the ESP header is inserted after the new IP header and before the encrypted original IP header. When the AH and ESP are applied together, the ESP header is located after the AH. The ESP trailer includes a padding field, the pad length and next header fields. The ESP encryption algorithm uses a symmetric encryption algorithm. The authentication algorithm uses a keyed message authentication code (MAC) or one way hash functions (MD5, SHA). ESP includes an authentication data field. This field contains an Integrity Check Value (ICV) computed over the ESP packet minus the authentication data. The authentication data field is optional, and is included only if the authentication service has been selected for the SA in question. The authentication algorithm specification MUST specify the length of the ICV and the comparison rules and processing steps for validation.

Internet Security Association and Key Management Protocol

The Internet Security Association and Key Management Protocol (ISAKMP) [45] combines the security concepts of authentication, key management, and security associations to establish the required security for commercial and private communications on the Internet.

ISAKMP defines procedures and packet formats to establish, negotiate, modify and delete Security Associations (SA). SAs contain all the information required for execution of various network security services, such as the IP layer services (such as header authentication and payload encapsulation), transport or application layer services, or self-protection of negotiation traffic. ISAKMP defines payloads for exchanging key generation and authentication data. These formats provide a consistent framework for transferring key and authentication data which is independent of the key generation technique, encryption algorithm and authentication mechanism.

ISAKMP is distinct from key exchange protocols in order to cleanly separate the details of security association management (and key management) from the details of key exchange. There may be many different key exchange protocols, each with different security properties. However, a common framework is required for agreeing on the format of SA attributes and for negotiating, modifying, and deleting SAs. ISAKMP serves as this common framework. ISAKMP is intended to support the negotiation of SAs for security protocols at all layers of the network stack (e.g., IPsec, TLS, OSPF, etc.). By centralising the management of the security associations, ISAKMP reduces the amount of duplicated functionality within each security protocol. ISAKMP can also reduce connection setup time, by negotiating a whole stack of services at once.

IPsec encryption and authentication algorithms

IPsec uses key exchange, digital signature, public key encryption, symmetric key encryption and message authentication cryptographic functions similar to TLS. The IPsec key exchange function uses the DH [5] key exchange algorithm and both RSA [6] and DSS [52] are used for the digital signing. The IKE protocol uses DH, RSA and DSS public key encryption algorithms. The ESP protocol uses DES [49] and 3DES [50] symmetric key encryption algorithms for data encryption as defined in [44]. The AH protocol uses HMAC [41] for message authentication and both the SHA [51] and the MD5 [62] one way functions are used for HMAC similar to TLS. The ESP protocol also uses HMAC for authentication. The key size of the DES encryption algorithm is 56 bits. This means that the maximum encryption key size in IPsec is 3*56=168 bits. Nowadays, 168 bits keys do not provide high security anymore. Therefore, in 2003 AES has been defined as new symmetric key encryption algorithm for use in IPsec in [28]. AES provides 128, 192 and 256 bits keys. The choice of cipher suites used in IPsec is not as rich as in TLS.

Internet Key Exchange

IPsec uses symmetric keys for encryption and authentication. Communicating peers negotiate security parameters before they begin to use security functions. The Internet Key Exchange (IKE) protocol provides algorithms to negotiate between communicating peers to create an SA. IKE is defined in [34]. The negotiation process defined in IKE consists of two phases:

1. Phase 1 is where the two ISAKMP peers establish a secure, authenticated channel for communication. This is called the ISAKMP Security Association (SA).

2. Phase 2 is where Security Associations are negotiated on behalf of services such as IPsec or any other service which needs key material and/or parameter negotiation. For example, an ESP or AH SA is created in this phase. The negotiation process secured by the ISAKMP SA.

Therefore, IPsec first creates a secure channel which is then used to establish a secure connection.

IKE defines the main, aggressive, base, and quick modes. Each mode defines a different handshake and handshake messages consisting of multiple payloads and headers. For example, the main mode uses a 6 way handshake, the basic mode uses a 4 way handshake. The main, aggressive and base modes are used in phase 1 and the quick mode is used in phase 2. IKE defines one more mode which is named new group mode. This mode serves to establish a new group which can be used in future negotiations. All modes are described in [34].

IKE uses the Deffie-Hellman (DH) key exchange algorithm for negotiation of secret key material or preshared secret keys. In the IKE key exchange process, the RSA and DSA algorithms are used for digital signatures, the RSA algorithm is used for public key authentication and the DES algorithm for encryption.

IPsec implementation

This section briefly describes existing IPsec implementations. The implementations are:

1. FreeS/WAN: Free Secure Wide Area Network project's primary objective is to help make IPsec widespread by providing source code which is freely available, runs on a range of machines including ubiquitous cheap PCs, and is not subject to US or other nations' export restrictions. The Linux implementation has reached its first major release and is ready for production use in manually-configured networks, using Linux kernel version 2.0.36. Later snapshots work on 2.2.x Linux kernels [2].

2. KAME: This project is a joint effort of six companies in Japan to provide a free IPv6 and IPsec (for both IPv4 and IPv6) stack for BSD variants to the world. This implementation already merged in FreeBSD, NetBSD, OpenBSD and BSD/OS. This implementation is especially dedicated to
IPv6/IPsec [3].

I analysed both implementations and chose the FreeS/Wan implementation, because the KAME implementation has some disadvantages. For example, it is dedicated to IPv6 and it works only under FreeBSD, NetBSD and OpenBSD systems. Our sctp prototype implementation was almost always tested under Linux and our socket API implementation was never tested under BSD systems.

Chapter 4

Standardised SCTP security solutions

4.1 TLS over SCTP

TLS is one security solution for SCTP. TLS over SCTP is standardised and defined in RFC3436 [37]. TLS requires a reliable, sequenced transport service from the underlying protocol, and for that reason, UDP or other such unreliable transport protocols cannot be used with TLS. A nice feature of TLS over SCTP is that many applications may use the same SCTP association, some of which require security, whilst others don't. This is an advantage over TCP, where it is necessary to create many connections.

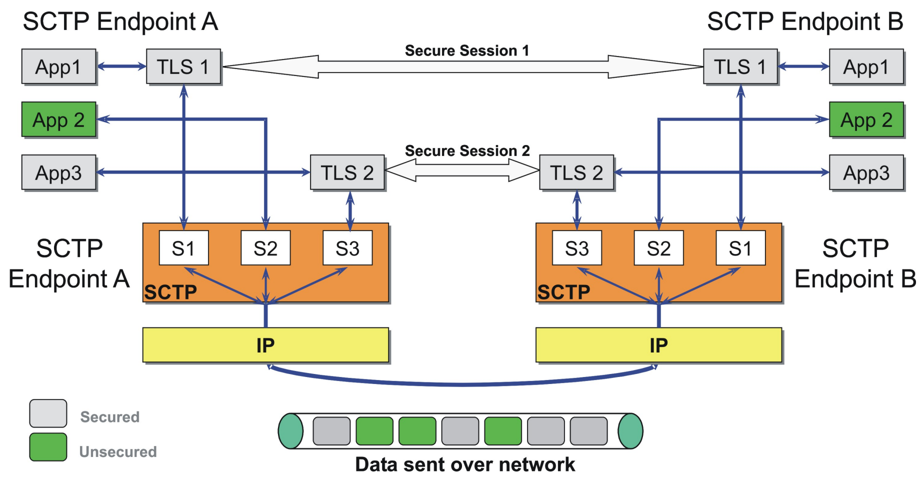


Figure 4.1: TLS over SCTP

Figure 4.1 shows a scenario of TLS over SCTP. In this scenario there are three applications, two of which require security. Every application requiring security creates a new TLS connection and secure session, but the packets sent over the network can be a mix of secure and insecure data. It is defined in RFC3436 [37] that the TLS application must use the same identified streams for input and output (SCTP streams are unidirectional). TLS connections can use two or more streams. Also defined is that the TLS handshake process can send the same messages by two different streams. If one of the streams is blocked or otherwise prevented from delivering data, then the other stream will transport the message. The TLS connection initialization and shutdown occurs any time during the lifetime of the SCTP association.

4.1.1 Data transmission

The following two mechanisms may operate during a TLS over SCTP data transmission. Which mechanism is used depends on the size of the application data. The TLS data block size has a maximum of 16384 bytes, and so application data must be not larger than this.

- Fragmentation: SCTP fragmentation occurs when the data size is bigger than the current SCTP MTU. The TLS application data is fragmented in several SCTP chunks and every SCTP chunk is transported separately. TLS only compresses, encrypts and authenticates the unfragmented (original) data. All fragmented data is concatenated at the receiver in the SCTP layer and delivered to TLS. Figure 4.2 shows how the TLS application data is inserted into an SCTP packet.

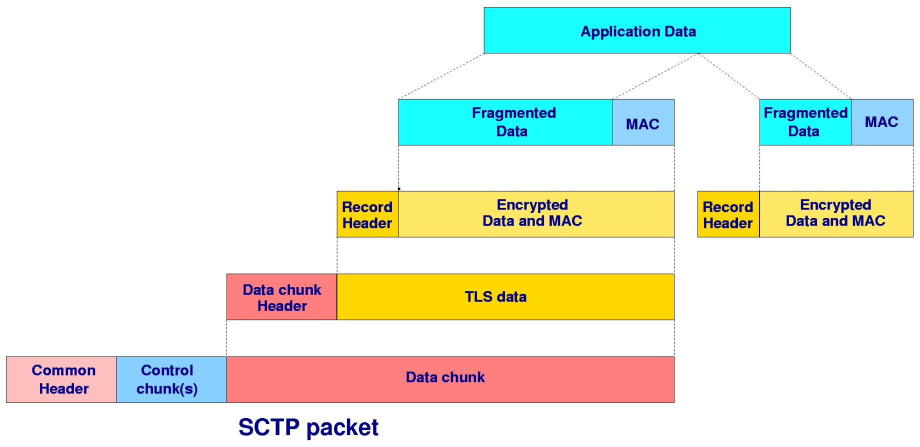


Figure 4.2: TLS data transport

- Bundling: SCTP bundles when the TLS application data is small. The bundling process is performed until the resulting IP packet size reaches the current MTU. Sending small application data messages increases overhead, because every TLS user message has its own record header and authenticated parts. For example, if the SCTP packet includes five TLS user messages, then it also includes five TLS record headers and authentication parts.

4.2 SCTP over IPsec

The second standardised SCTP security solution is a combination of SCTP with IPsec. The use of SCTP with IPsec is defined in RFC3554 [12]. There are, however, at the moment no implementations available which are supporting this RFC. In this section SCTP over IPsec is described. SCTP over IPsec is similar to TCP over IPsec, except that SCTP uses multiple addresses.

4.2.1 One-to-one IP address relationship

In this scenario both associated endpoints have only one IP address. Before transferring secure data over IP, the IPsec Security Association (SA) must be negotiated. The IPsec SA is uni-directional and every associated endpoint has one or two SAs. When the endpoint uses AH or ESP then it has only one SA, but when it uses both ESP and AH it has two SAs.

Figure 4.3 shows three applications using same SCTP association. The IPsec SA is established if one of these applications requires security. Secure and insecure data cannot be mixed in SCTP over IPsec, and so data sent from the applications not requiring security must now also be secured. This is inefficient when an application requiring security sends a small amount of data, for example, billing or control data, and the applications which don't require security send large amounts of data, for example, large files or media streams.

4.2.2 One-to-many IP address relationship

In this scenario, two endpoints have more than one IP address (multihoming) and every endpoint has multiple paths. Every path has a different IPsec SA. We know when the primary path is inaccessible then the SCTP association selects a new primary path from the available redundant paths. In this case, the new IPsec SA is used for a security mechanism, which belongs to the new selected primary path. Therefore, before beginning secure transmission, all IPsec SAs have to be negotiated.

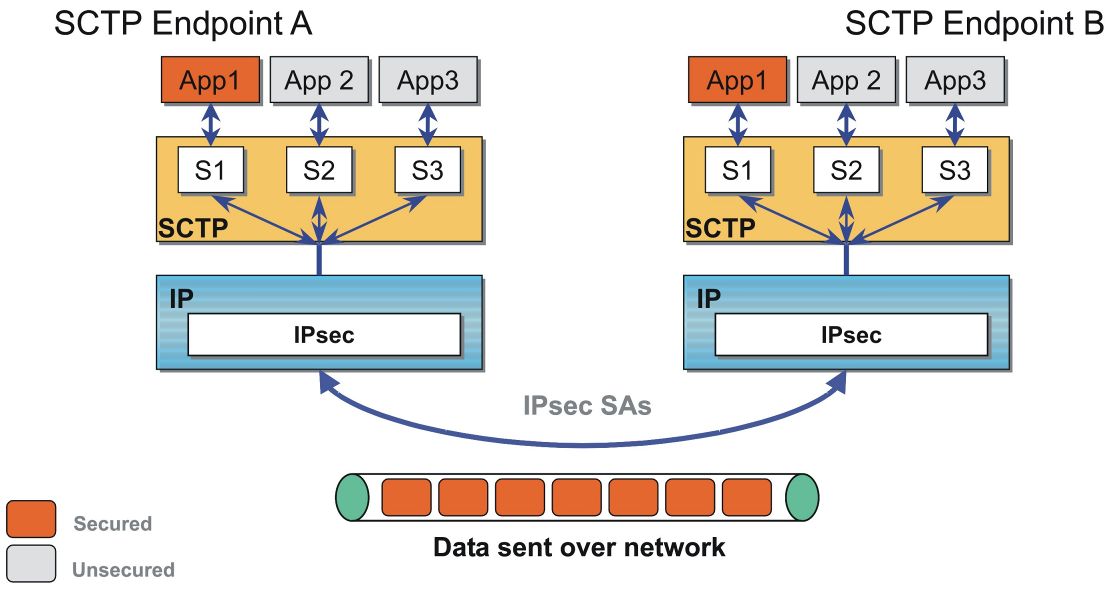


Figure 4.3: SCTP over IPsec uses a single IP address

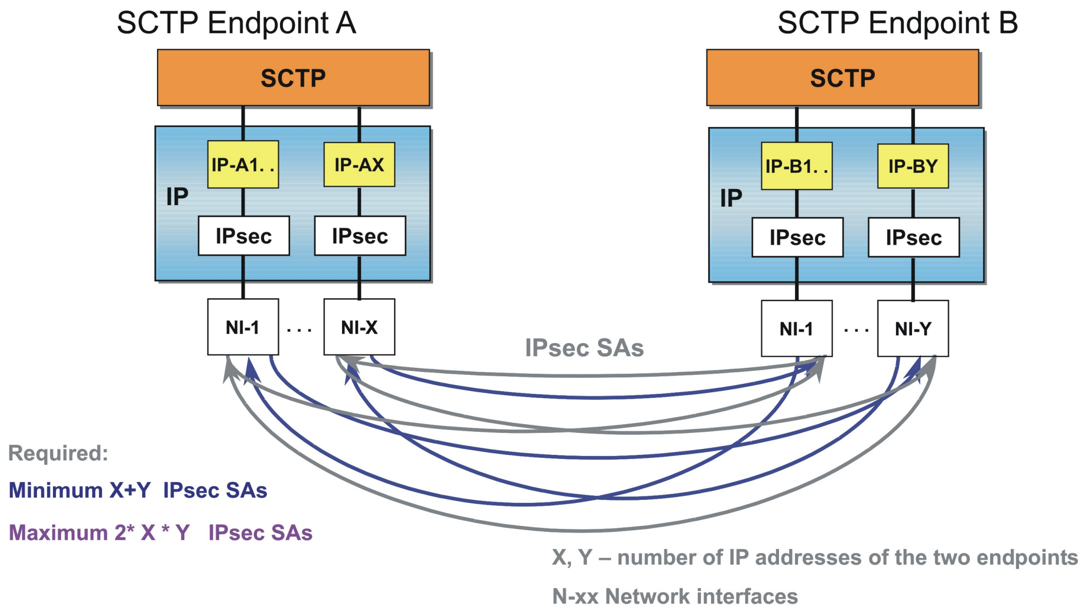


Figure 4.4: SCTP over IPsec uses multiple IP addresses

Figure 4.4 shows two endpoints with N and M IP addresses. In this scenario the user has to negotiate at least N+M IPsec SAs. When the user has an extra IP routing configuration, then a maximum of 2*N*M IPsec SAs may have to be negotiated.

IPsec is implemented in the operating system or kernel, and so it is more complicated, and requires a privileged user. For example in Linux, the user has to be ”root”.

4.2.3 Data transmission

Associated SCTP endpoints can use both IPsec transport and tunnel modes and all SCTP packets transported over IP will be secured and/or protected. The security mechanism depends on which IPsec security protocol is used. Figure 4.5(a) shows the SCTP packet transmission using AH [38]. In this case the SCTP packet and IP header are protected from any changes, but the third party can see what data is being transported. Figure 4.5(b) shows the SCTP packet transmission using ESP [39]. In this case the SCTP packet is encrypted and the third party cannot see inside the packet. The SCTP endpoints can use both AH and ESP. Figure 4.5(c)shows the SCTP packet transmission using the combination of ESP and AH.

4.3 Assessment of existing security solutions

As a basis for my further work, I have performed a qualitative assessment of the existing standardised security solutions for SCTP. In general, a secure end-to-end transport service over SCTP using bi-directional communication, in-sequence and reliable delivery, multiple streams and multi-homing can be realised by using either TLS over SCTP or SCTP over IPsec.

Both provide cryptographically secure integrity and authentication checks without encryption by using cryptographic hash functions and message authentication codes. While TLS uses a cipher with a non-null hash algorithm for this purpose for example, IPsec uses the Authentication Header protocol (AH) TLS_RSA_WITH_NULL_SHA [39].

Methods for authentication can be based on the mechanisms provided by TLS, i.e. mutual authentication with X.509 certificates. Alternatively, simpler mechanisms like server authentication with an X.509 certificate and client authentication via CHAP can be used. When using IPsec and confidentiality is required, the ESP protocol with a nun-null encryption algorithm can be used to protect the user data. Obviously, TLS also provides encryption by using one of the available cipher suites.

However, some scalability problems and limitations, with respect to the usability of the full SCTP feature set, exist for both solutions as summarised below.

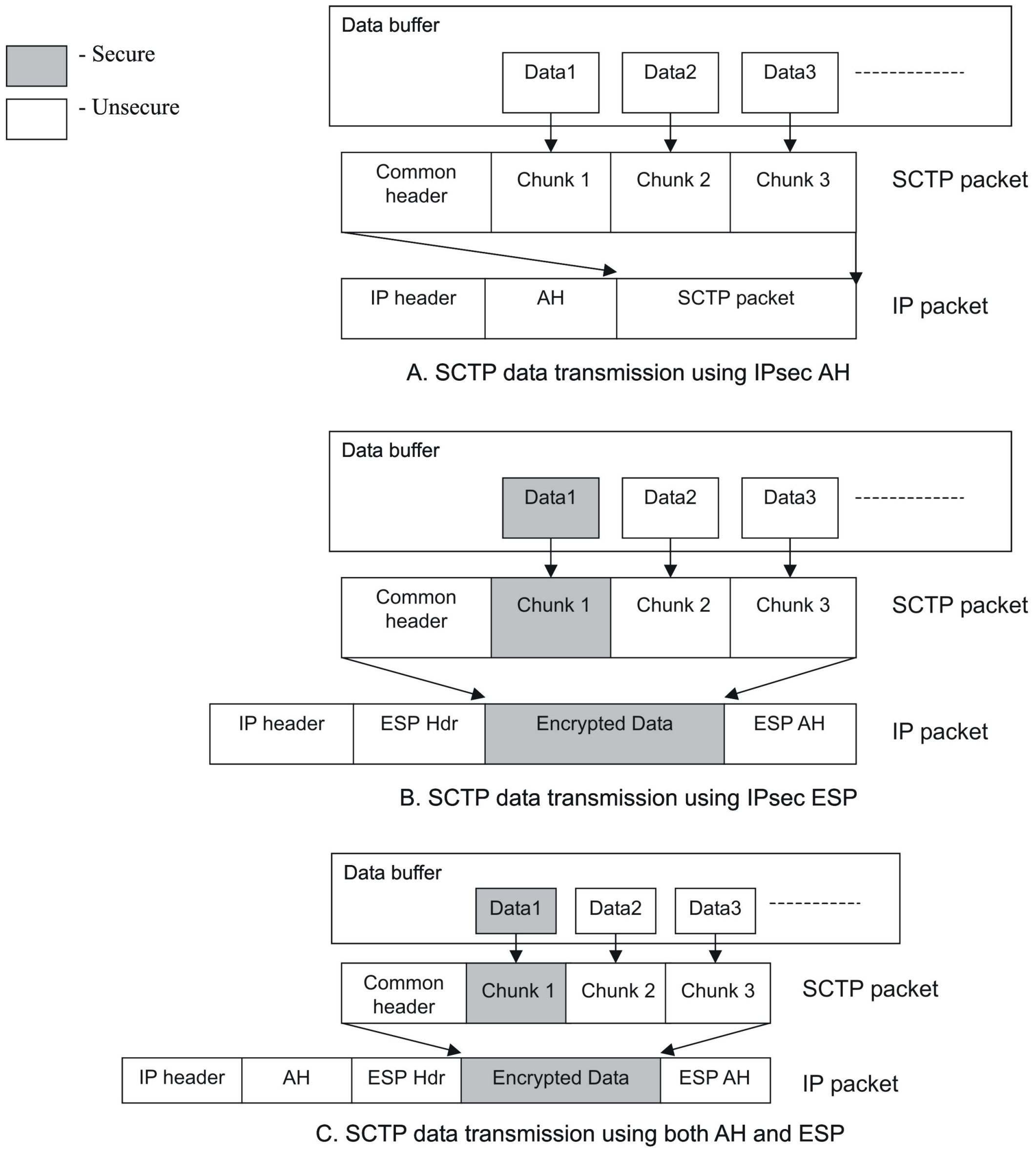


Figure 4.5: SCTP data transmission over IPsec

4.3.1 Assessment of TLS over SCTP

The deployment of TLS over SCTP may lead to potential scaling problems in scenarios with a large number of streams and strict bounds on association setup delays, as well as in the case of frequent re-keying due to the large number of separate TLS connections required. Moreover, only SCTP user data is protected, whereas control information of the underlying SCTP associations remains vulnerable.

Since TLS expects reliable and in-sequence transport, the TLS over SCTP solution does not support the unordered delivery service and the partial reliability extension of SCTP. Appropriate extensions would have to be defined, standardised and implemented for TLS to support these features. However, these features, as well as uni-directional communication, still can be used for unsecured communication within protected associations.

TLS protects only the SCTP user data, neither the control chunks exchanged within the corresponding SCTP association, nor any IP layer information can be protected (as TLS is only a SCTP user). This is a general problem with security solutions operating above the transport layer, which possibly allows some forms of denial of service attacks against the SCTP association itself. IPsec, on the other hand, secures the complete SCTP packets.

In TLS over SCTP secure and insecure applications can use the same association. Moreover, every TLS application uses different cryptographic parameters such as key material, master secret key, authentication key, etc. So, when one of the TLS connection's security parameters are compromised, the other TLS applications should not be effected. The third part would see only the compromised TLS application's data but not any other TLS applications' data.

Multihoming is transparent to TLS and TLS applications can use redundant paths. TLS over SCTP is also compatible with the dynamic address reconfiguration extension of SCTP. Security considerations of TLS over SCTP are described in section 9 in RFC3436 [37].

4.3.2 Assessment of SCTP over IPsec

One obvious disadvantage of the SCTP over IPsec solution is the fact that – in principle – it requires the configuration of multiple, possibly many, IPsec Security Associations (SA) to support a multi-homed SCTP association. To solve this problem a solution specified in RFC3554 [12] has been proposed how the SA definition of IPsec and IKE can be extended to support multi-homing.

This implementation-specific feature can only be used in an inter-operable fashion, if the key management protocols also allow a simple way to configure SAs for sets of endpoint addresses.

For this purpose, RFC3554 presents an enhancement to ISAKMP and IKE, allowing SAs to be configured for lists of IP addresses rather than single addresses. This would mean that only two IPsec SAs would be required to protect any SCTP association. However, this RFC is only available since 2003 and nowadays there are no IPsec implementations available to provide this extended functionality.

Another disadvantage of SCTP over IPsec is that, unlike TLS, IPsec cannot mix secure and insecure data. When a large portion of the transmitted data does not require security, IPsec is inefficient.

The advantage of SCTP over IPsec is that it supports unordered delivery and has no problem with multiple streams due to the fact that IPsec establishes a SA per path but not per stream. All applications connected to the same SCTP association use the same IPsec SA. Moreover IPsec does not increase overhead like TLS if small user messages are bundled and it protects SCTP control chunks and common header. The security considerations of SCTP over IPsec are defined in section 4 of RFC3554 [12].

4.3.3 Comparison of standardised security solutions

In summary, both solutions have limitations with respect in full support of SCTP features and extensions. In addition, there are inefficiencies, which can result in poor performance in some application scenarios, in both solutions. The table 4.1 shows a comparison between the standard SCTP security solutions.

In table 4.1 ”advantage” and ”disadvantage” provide a qualitative comparison of the security solutions with respect to usability, computation and management cost or performance. ”Not possible” means a SCTP security solution can not support this specific functionality at all. Based on the comparison summarised in table 4.1, the S-SCTP concept has been defined to avoid the drawbacks of the existing solutions as discussed in detail in chapter 5.

Configuration of TLS over SCTP is simple for TLS-aware applications which can be used without major modifications. Also, the adaptation of TLS to SCTP does not require a great deal of modifications as our prototype implementation shows (see section 5.4) and can be done without touching the operating system. If IPsec is used, the management of security associations – including key exchange – must be taken care of by the application and operating system. IPsec has a rather complex API which means considerable effort.

Criteria	TLS over SCTP	SCTP over IPsec
Scalability for multiple streams	-	+
Support for SCTP multi-homing	+	(-)
Overhead for small messages (bundling)	-	+
Overhead for long messages (fragmentation)	+	-
Protection for unordered delivery service	no	+
Protection for SCTP control chunks	no	+
Flexible multiplexing of secure/insecure streams	+	no
Management of security sessions (handling, automation)	+	-
Partially Reliable Transport (SCTP extension)	no	+
Dynamic Address Reconfiguration (SCTP extension)	+	-

Keywords: ”+” - Advantage, ”-” - Disadvantage, ”no” - Not possible

Table 4.1: Comparison of TLS over SCTP and SCTP over IPsec

Chapter 5

Secure SCTP

SCTP requires strong security solutions to protect sensitive data as outlined in section 2.2. In the previous chapter, I discussed the significant limitations and inefficiencies when using SCTP together with standard security protocols (TLS and IPsec). The main result of the assessment of the existing security solutions in section 4.3 is:

- that both standardised security solutions fail to support all SCTP features and extensions
- that due to multi-streaming at the upper service access point and multi-homing at the lower service access point, non-integrated solutions are potentially inefficient in some scenarios.

Appropriate combinations of TLS and IPSec to secure SCTP transport could be used - in principle - to solve the first issue in most cases, but the complexity and overhead of such a combination would be prohibitive. Incorporation of the features required to fully support SCTP on the other hand would require a major redesign of the TLS and IPSec concepts and it was, therefore, not considered a valid option.

Based on this analysis, I propose a security extension to SCTP – Secure-SCTP (S-SCTP) – to solve these issues by integrating crypto functions into SCTP itself, in an efficient and user-friendly way. The proposed extension is designed to avoid the drawbacks of the non-integrated solutions, whilst still providing full compatibility with the original SCTP protocol when no protection is being used.

5.1 Basic design goals for S-SCTP

Following the analysis of the standardised SCTP security solutions, I considered the following criteria for designing S-SCTP:

- **Downward compatibility:** S-SCTP must be compatible to standard SCTP and support all its features and extensions.
- **Security:** This is the main point of S-SCTP. S-SCTP has to provide authentication data integrity and confidentiality which are also required to protect SCTP control information.
- **Performance:** The potential performance limitations due to the combination of multistreaming and multihoming should be avoided.
- **User friendliness and flexibility:** S-SCTP should be easy to configure for a user by providing a simple API. It should also provide maximum flexibility, e.g. with respect to cipher suites and mixing of secure and unsecure data.

These four criteria define the main requirements for defining the S-SCTP concept. Each of these criteria results in specific design choices and properties. S-SCTP has to fulfill all of these requirements in order to avoid the drawbacks of the existing solutions.
S-SCTP has to be compatible to standard SCTP, i.e. it has to support standard SCTP association management (initialisation, termination, ...) and data transport. S-SCTP is not standardised and, thus, not widely deployed. Therefore, in many application cases one of the associating endpoints will support S-SCTP while the other endpoint does not and compatibility is absolutely crucial. To keep normal SCTP association initialisation, S-SCTP secure session initialisation begins after the normal SCTP association initialisation has been performed. Fortunately, in SCTP the chunk types are encoded such that the two highest-order bits specify the action that must be taken if the processing endpoint does not recognise a chunk type. An endpoint that receives unrecognised chunks looks at the first two bits. If they are equal to ”11” then processing of this chunk is skipped and processing is continued with the next chunk. Therefore, all new S-SCTP chunk types have been chosen from this range to keep compatibility.
The main focus of S-SCTP is security that has to be provided for major SCTP application scenarios as already discussed in chapter 2. As a result, there exist several security requirements for S-SCTP. S-SCTP has to secure SCTP control chunks to protect association management and the security mechanism must be flexible enough to efficiently support all application scenarios while remaining simple and easy to use. Moreover, secure session

management (secure session initialisation, rekeying, termination, ...) must be fast and efficient. To provide such fast and efficient secure session management, it is crucial to use only one single secure session for the complete SCTP association instead of multiple – possibly many - as for TLS over SCTP in multi-streaming mode and for SCTP over IPsec in multi-homing mode. However, it is practical to use similar handshake mechanism as for TLS and IPsec because these mechanisms are proven already. Also, the idea to have a rich choice of cipher suites to provide flexible security mechanisms and the possibility to easily extend them has been taken over from the existing mechanisms.
S-SCTP has to provide an efficient data path to be suitable for high throughput. Therefore, S-SCTP supports a flexible mix of secured and unsecured data – not only on a per-stream basis as TLS over SCTP but even on a per-chunk basis. In addition, care has been taken to minimise protocol overhead. To achieve this, S-SCTP calculates the HMAC per packet and not per chunk. To be able to handle this flexibility, S-SCTP provides a set of predefined security levels which are easy to select and, furthermore, can be changed during a secure session lifetime. To make the rich functionality of S-SCTP easily accessible for the user application, an S-SCTP socket API has been defined providing:

- simple initialisation, rekeying and termination of secure sessions
- flexible choice of standard cipher suites
- easy access to newly defined cipher suites if required
- simple selection and modification of security levels

The main challenge when defining S-SCTP was to find a solution fulfilling all the - sometimes conflicting – design goals at the same time. To achieve this, every design decision had to be evaluated with respect to its impact on

- security (avoid new attack possibilities, reuse of proven mechanisms),
- performance and scalability (both control and data path) and
- complexity (with respect to specification, implementation and usability)

A more detailed discussion of the specific design decisions taken and the reasons for making these choices is provided in section 5.2.

5.2 Basic concept of S-SCTP

The basic concept of the S-SCTP solution is that an association should have only one secure session for all data and streams and it uses the same secure session and secure session key for all addresses in a multihoming scenario. This distinguishes S-SCTP from TLS over SCTP and SCTP over IPsec. In order to achieve this, I integrated the security mechanism between the upper functional block which performs grouping of SCTP data chunks to SCTP packets (also named bundling) and the lower functional block which performs the selection of network paths by choosing a destination address to send the SCTP packet as shown in figure 5.1.

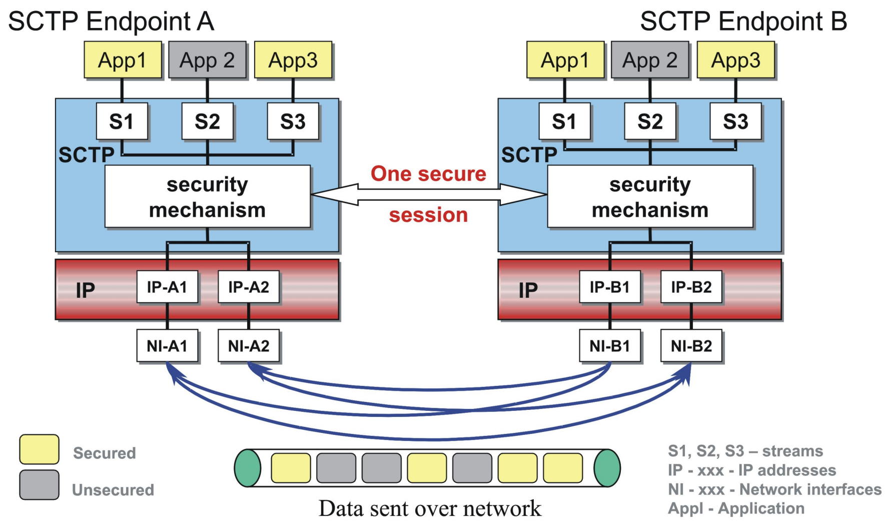


Figure 5.1: S-SCTP concept

However, S-SCTP can still mix secure and insecure messages, even if all applications use a common secure session.

Before choosing this solution for S-SCTP, I evaluated SCTP and found three possible design alternatives where to integrate security mechanisms. Figure 5.2 shows the alternative solutions.

In figure 5.2A, the security mechanism is integrated before bundling different streams, so that every stream has its own security mechanism. We know that SCTP streams are independent. Therefore, such a design would result in multiple stream specific security mechanisms which have to be individually organised. If an application uses many secure streams then the SCTP association requires many security sessions similar to TLS over

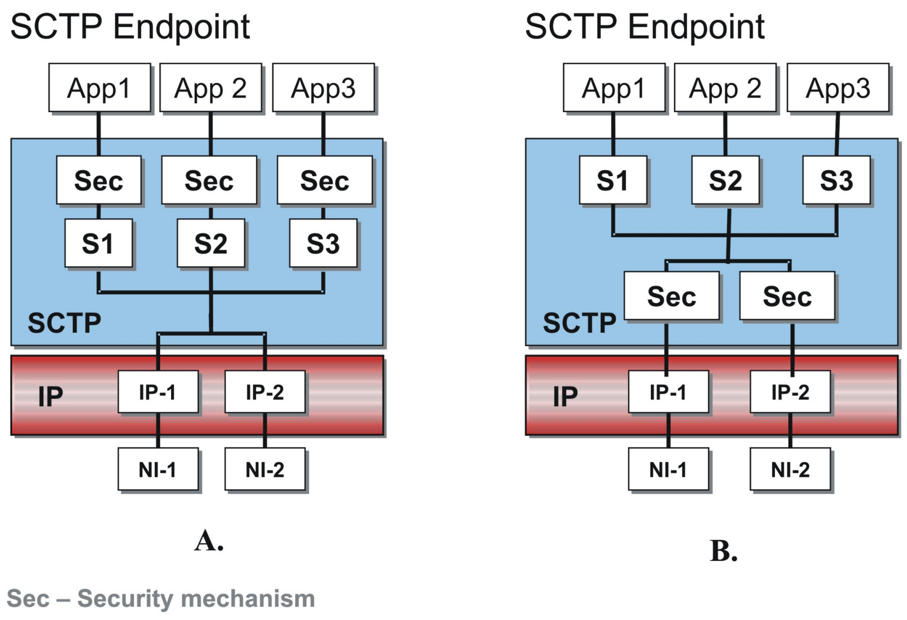


Figure 5.2: Possibilities to integrate a security mechanism into SCTP

SCTP. The distinct advantage compared to TLS over SCTP would be, however, that such a solution supports the unordered delivery service and the partial reliability extension which are not supported in TLS over SCTP. In addition, also protection of SCTP control chunks could be provided such that all the major functional drawbacks of TLS over SCTP could be avoided. Yet, this solution still has the issue with multiple streams which is also occurs in TLS over SCTP (see section 4.3.1) and potentially limits performance and scalability.

In figure 5.2B, the security mechanism is integrated after splitting up the different destination addresses. In this design the security mechanism is not affected by the number of secured streams. Also, the security mechanism could be defined such that mixing of secured and unsecured traffic becomes feasible. However, the number of security associations scales with the number of IP addresses in a multi-homing scenario. Therefore, this solution would still suffer from the scalability and management issues already described for SCTP over IPsec (see section 4.3.2).

While these alternatives would still have some problems of the standardised security solutions, the selected S-SCTP solution can avoid these drawbacks.

In the following sections, I describe in more detail how the design goals have been reached and what the design decisions were I have taken.

5.2.1 Downward compatibility

S-SCTP is an extension like the other two existing SCTP extensions described in section 3.1.4 and it is not standardised. That's why it cannot be expected that all implementations would use it. There are scenarios with two endpoints, where one supports the security extension and the other doesn't. In this case it is mandatory that both endpoints are able to connect and to exchange information in clear text. There are no security functions in this scenario. Therefore, S-SCTP has to provide a mode which is compatible to standard SCTP. During a secure session initialisation of an endpoint can recognise whether the other supports the secure extension or not. During secure session initialisation, the first (S-SCTP) endpoint sends a secure session open request to the second endpoint. If the second endpoint supports the secure extension it accepts the request and sends back a response. This is the normal secure session initialisation procedure described in section 5.3.3. If the second endpoint doesn't support the secure extension, it sends back an error chunk with ”unrecognised chunk type”. After receiving the error chunk with ”unrecognised chunk type” the first endpoint notifies to the upper layer that the second endpoint doesn't support the secure extension. Nevertheless, the standard SCTP association still is operational. From then on for the association lifetime, the first endpoint ignores all upper layer secure session open requests. The notifications from S-SCTP to the upper layer are described in Appendix B.

5.2.2 Security

The security functions of S-SCTP are similar to TLS and IPsec. S-SCTP provides the following two security functions for both data and control chunks:

- **Data integrity and origin authentication:** These have to be provided by SCTP for the transported user data and for the peer-to-peer control information of SCTP itself in order to avoid vulnerabilities, e.g. connection hijacking by using the Dynamic Address Reconfiguration extension [74]. To minimise transmission overhead, an HMAC [41] is computed over the whole SCTP packet including all chunks and the common header (figure 3.1). If SCTP has to do message fragmentation, the SCTP fragment for every segment carries its own HMAC to simplify and speed up the retransmission process.

- **Data confidentiality:** Flexible and optional encryption of SCTP user data and control information has to be provided. The decision to encrypt a chunk depends on an appropriate indication flag which is set by the application. In order to do an encryption with minimum overhead, all chunks within one packet which have to be encrypted are grouped and encrypted together.

5.2.3 Performance

As previously stated, there are several factors which affect performance in the standardised SCTP security solutions. Therefore, S-SCTP's efficiency and performance over a broad range of application scenarios and SCTP parameter settings were all together the main objectives of my design.

One major factor is that S-SCTP uses only one secure session per association. This decreases secure session initialisation cost. The TLS handshake uses 13 messages (see figure 3.12). Therefore with TLS over SCTP in a multistreaming scenario (N secure streams), N*13 messages are exchanged between two associated endpoints to establish the secure sessions completely. In the same scenario S-SCTP uses 8 messages to establish a secure session (see figure 5.11). This decreases setup time and bandwidth consumption. The same is in principle true when compared to IPsec with multihoming, but normally the number of IP addresses is not so high, so the effect is not that dramatic.

When a secure session is long-lived, it is necessary to update the secret keys periodically. For S-SCTP, only one key has to be updated, for TLS over SCTP with multistreaming, or SCTP over IPsec with multihoming, this is much more complex. Because every secure session independently performs the update mechanism, the cost in terms of computations and bandwidth consumption obviously increases when a large number of streams or IP addresses are used in an SCTP association for the latter two solutions.

Moreover, TLS implementations define a data structure, which stores security parameters related to a TLS connection. Every TLS connection has its own structure. This means when N TLS connection are used, then N structures have to be replaced in memory. Also certificates that are used in TLS handshakes have to be replaced in memory. Therefore, TLS over SCTP memory usage obviously depends on the number of secure streams which is a drawback compared to S-SCTP and SCTP over IPsec.

Overhead is one factor that decreases the throughput. With respect to protocol overhead, the SCTP over IPsec and the S-SCTP security solutions are rather similar. The overhead (in bytes) can be calculated as follows:

SCTP over IPsec: 20(IP header)+ 8(ESP header)+ 28(AH header)+12(SCTP Common header)= 68 bytes

S-SCTP: 20(IP header)+ 12(SCTP Common header)+ 16(EncData chunk header)+ 24(AUTH chunk header)= 72 bytes

From above calculations the difference in overhead size between SCTP over IPsec and S-SCTP is just 4 bytes. This result is not influenced by the message size, because fragmentation on one side and message bundling on the other has the same consequences for the overhead in both solutions.

However, this is not true when comparing the overhead for S-SCTP and TLS over SCTP.

In SCTP, small messages can be bundled together in one packet, but they are increasing the overhead. Figure 5.3 shows the relationship between overhead and message size (MTU=1500 bytes Ethernet) for S-SCTP and TLS over SCTP which can be derived by a simple calculation.

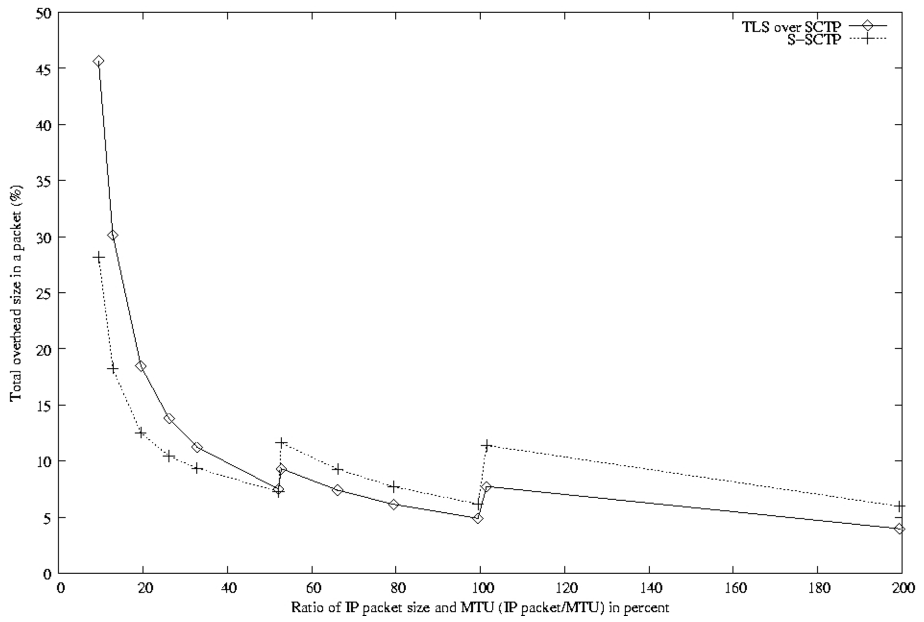


Figure 5.3: Relationship between packet overhead and message size

The y axis indicates the total overhead percentage, which includes IP header, SCTP common header, authentication header and data chunk header. For example, 40 percent means 2/5 of a packet is just overhead and 3/5 is a user data. The x axis indicates the ratio of IP packet size [1] to MTU.

We can distinguish three zones:

[1]MTU size(1500 byte) - IP header(20 byte) - S-SCTP overhead(72 byte) leaves 1408 bytes maximum data length per packet if a packet includes only one DATA chunk. If IP packet includes two DATA chunks then maximum data length per chunk is (1408 - Second DATA chunk header(16 byte))/2 = 696 byte.

- The first zone is between 0% and 52.5% [2]. This is the bundling zone of SCTP. The bundling mechnism will continue until the IP packet size reaches 52.5 percent of the MTU.
- The second zone is between 52.5% and 100%. This is the zone where every packet inlcudes only one message but no fragmentation is necessary.
- The third zone is over 100%. This is the fragmenting zone where messages bigger than the MTU are fragmented.

TLS over SCTP has a high overhead in zone 1 and S-SCTP has obviously advantages. In S-SCTP, the authentication field is calculated for the whole packet not each separately for each message as in TLS over SCTP. Therefore, every SCTP packet includes only one authentication code and several messages. Because of this, S-SCTP is clearly more effective than TLS over SCTP for small messages. As you see in figure 5.3, the overheads of TLS over SCTP and S-SCTP are increasing on the boundary of zone 1 and zone 2, because here a fragmentation occurs and the data message is fragmented into two packets thus increasing overheads.

Because there is only one message per SCTP packet, S-SCTP has a total overhead of 72 bytes (12 bytes common header, 16 bytes EncData chunk header, 16 bytes DATA chunk header and 28 bytes AUTH chunk), whereas TLS over SCTP has only an overhead of 52 bytes (12 bytes SCTP common header, 16 bytes DATA chunk header, 4 bytes TLS record header and 20 bytes authentication code). Therefore, TLS over SCTP is slightly more efficient than S-SCTP in zone 2.

In zone 3, S-SCTP requires one authentication field per SCTP packet, whereas TLS over SCTP authenticates a whole message which is transported in two or more SCTP packets. Therefore, TLS over SCTP has less overhead for fragmented messages.

To clarify the jumps in overhead, consider e.g. a message of length 1410 bytes (in the S-SCTP scenario). For a 1500 byte MTU resulting in maximum chunk size of 1408 bytes, the data message will be divided into two parts of size are 1408 and 2 bytes, respectively. The first packet size is 1408+72+20=1500 bytes and the second packet size is 2+72+20=94 bytes. The total overhead size of the two packets is (72+20)*2=184 bytes and total length of two packet is 1500+94=1594. The percentage of the overhead is (184/1594)*100=11.54%. If we calculate the overhead percentage before fragmentation occurs, we have only one packet which has 1500 bytes and overhead size is 72+20=92 bytes. The percentage of overhead is therefore 92/1500=6.13%. Therefore, the overhead

[2]Maximum data length per chunk(696 byte) + IP header(20 byte) + S-SCTP overhead(72 byte) = 788 byte maximum IP packet size in bundling and this is 52.5% of MTU (1500 byte)

will be increased in every time when an additional fragment is needed i.e. at 200%, 300% etc.

A major performance advantage of S-SCTP compared to IPsec is that S-SCTP can mix secure and insecure messages. We know the computation time of encryption mechanisms is relatively high. In SCTP over IPSec, all data will be encrypted which will unnecessarily increase computation time and decrease throughput. In S-SCTP, just the data requiring security will be encrypted. Of course, the effect of computation time depends on the ratio of secure and insecure messages. When the percentage of secure messages is high (e.g. 90 percent) then both security solution's computation times are similar. When the percentage of secure messages is low (e.g. 10 percent) then S-SCTP has lower computation time and higher throughput than SCTP over IPsec. In section 6.4 results of experiments with a mix of secure and insecure messages are discussed.

Another factor that influences the performance is the implementation of the security protocols and their cryptographic algorithms. IPsec is completely implemented in the kernel space, whereas TLS and S-SCTP are implemented in user space. That's why IPsec is highly optimised and its performance is in general superior to TLS and S-SCTP. Therefore, the comparison experiments for the SCTP security solutions provided in section 6.4 are not ideal. However, I tried to use as identical conditions as possible in my experiments by using the same encryption and authentication algorithms in every experiment.

5.2.4 Flexibility and user friendliness

For users to easily take advantage of the full functionality of S-SCTP without excessive configuration effort, I defined several security levels:

- **Security Level 0:** If this level is set S-SCTP does not use any of the security functions. Therefore this level is fully compatible to standard SCTP and it provides downward compatibility to implementations not supporting the S-SCTP extension.
- **Security Level 1:** This security level applies the HMAC algorithm to all SCTP packets of the association, i.e. every user data chunk and all peer-to-peer control information of SCTP are authenticated and their integrity is checked.
- **Security Level 2:** In addition to the HMAC operation, *selected user data chunks* are encrypted based on an indication flag provided by a S-SCTP aware application according to its security requirements. In order to minimise computational effort, this security level avoids e.g. encryption of high volume and insensitive user data while providing encryption for sensitive control information (e.g. for billing) which has a relatively low volume of information.

- **Security Level 3:** If this level is selected for an association, HMAC is applied to all SCTP packets and *all data and control chunks* are encrypted.

Within a S-SCTP session, two endpoints may well use different security levels: for instance, one endpoint may only require authentication whereas the other endpoint also requires privacy. The first would choose security level 1 in this case, and the second would choose security level 3. The security levels can be selected by the application and can be modified dynamically during the lifetime of an SCTP association. Every time when the endpoint selects a new security level it informs the associated endpoint.

Defining security levels, S-SCTP can provide flexible security functionalities. From table 5.1 showing the security functional comparison of all security solutions, it becomes clear, that the various S-SCTP security levels allow to mimic basically all operation modes of the other solutions. For example, S-SCTP security level 1 provides the same level of protection as IPsec ESP with NULL encryption in transport mode.

Protocols		IP	SCTP	Data
SCTP		–	–	–
TLS	with encryption	–	–	E+A
	NULL encryption	–	–	A
S-SCTP	Security level 0	–	–	–
	Security level 1	–	A	A
	Security level 2	–	A	E+A
	Security level 3	–	E+A	E+A
IPsec transport mode	AH	A	A	A
	ESP with encryption	–	E+A	E+A
	ESP with NULL encryption	–	A	A
IPsec tunnel mode	AH	A	A	A
	ESP with encryption	E+A	E+A	E+A
	ESP with NULL encryption	A	A	A

Keywords: IP - IP header, SCTP - SCTP common header + control chunks
A - authenticated, E- encrypted

Table 5.1: Security functional comparison of all security protocols

We can see from table 5.1 that TLS cannot protect the IP header, the SCTP common header and the SCTP control chunks, S-SCTP cannot protect the IP header. In transport mode of IPsec cannot provide an encryption mechanism for the IP header but in tunnel mode IPsec provides security functions for all headers and data. In general, a security protocol located at a specific layer cannot protect information belonging to the underlying layers, therefore, IPsec can provide the most comprehensive security.

However, it is relatively complex to handle IPsec. An IPsec security association (SA) must be established before starting a secure transmission. It is not possible to initialise or delete a SA during the transmission because there is normally no management interface between an application and the IPsec SA. In other words, there are no IPsec API functions except PF_KEY. PF_KEY is a new socket protocol family defined in [46] and used in IPsec key management. But without IPsec policy definitions, PF_KEY cannot provide secure functions. PF_KEY's purpose is to create a SAD and connect to the policy entry. The IPsec working group of IETF currently works on IPsec policy management and they want to create a new socket family, named PF_POLICY, in the future. For the time being IPsec is not dynamically manageable. A user has to configure IPsec manually and has to have a high privilege (e.g. in Linux ”root”). Therefore in IPsec it is much more difficult particularly to remove and add IP addresses. TLS and S-SCTP on the other hand have API functions which provide flexible management, i.e. a user can create, delete or modify a secure session at any time.

5.3 Architecture of S-SCTP

To realise the design goals of S-SCTP, the SCTP protocol architecture has to be extended as shown in figure 5.4. The main control entity for SCTP is the SCTP controller module consisting of a complex state machine which controls the status of an SCTP association (see e.g. [75]). In addition to this controller, I introduce a new state machine which controls the activation and deactivation of cryptographic processing including initial key exchange and re-keying. The path management module to monitor and control the different network paths to the peer can remain unchanged. The stream layer responsible for in-sequence or unordered delivery per stream and for fragmentation/defragmentation of user messages has to take into account the additional overhead for the cryptographic functions when doing fragmentation. Therefore, a simple modification of the calculation algorithm is required. The reliable transport and flow control modules which are responsible for buffering, acknowledgements and retransmissions, are not affected by the security extensions.

The main part of the new functionality of S-SCTP – apart from key and security session management – is integrated into the packet assembly and disassembly module which is responsible for mapping user data and control chunks to SCTP packets together with the common SCTP header.

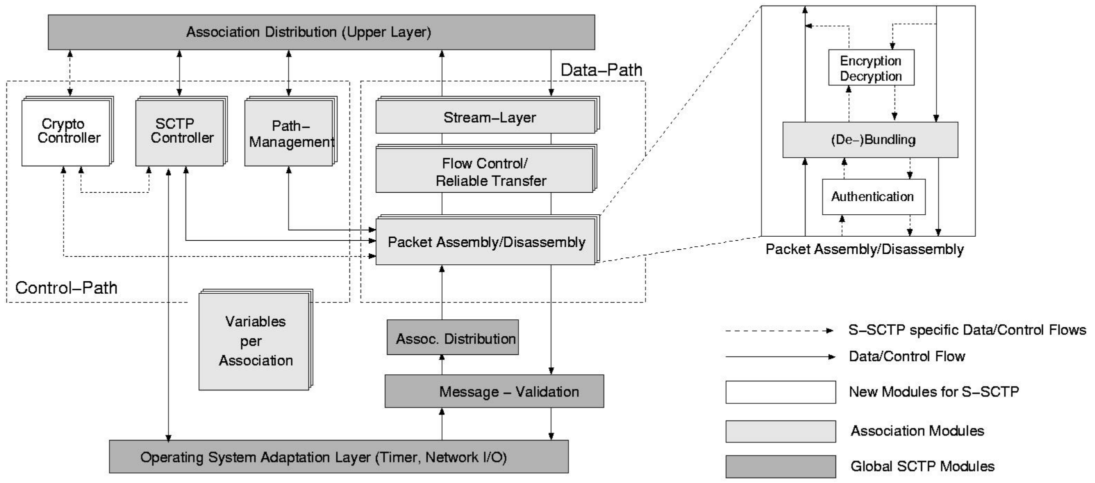


Figure 5.4: S-SCTP protocol architecture

5.3.1 New chunk types for S-SCTP

The standard SCTP packet format is described in section 3.1.1. SCTP chunks have common headers which include an 8 bit type. In SCTP, the highest-order two bits of chunk types described below specify the action that must be taken if the processing endpoint does not recognise the chunk type.

- 00 - Stop processing this SCTP packet and discard it, do not process any further chunks within it.
- 01 - Stop processing this SCTP packet and discard it, do not process any further chunks within it, and report the unrecognized parameter in an ’Unrecognized Parameter Type’ (in either an ERROR or in the INIT ACK).
- 10 - Skip this chunk and continue processing.
- 11 - Skip this chunk and continue processing, but report this in an ERROR Chunk using the ’Unrecognized Chunk Type’ cause of error.

The range ”11” was chosen when defining new chunk types for SCTP to provide compatibility to standard SCTP. If an S-SCTP endpoint sends a chunk specific to S-SCTP (e.g. for secure session establishment), a receiving endpoint not supporting the S-SCTP extension just ignores the chunk and reports the unrecognised chunk. Thus, the S-SCTP endpoint knows that the extended functionality cannot be used and continues with standard SCTP procedures and a normal SCTP association without security mechanisms is maintained. To support the S-SCTP extensions, I defined a number of new chunk types (see also table 5.2) which are used for the initial establishment of the secure session, intermediate re-keying handshakes, and the deletion of secure sessions. The detailed specification of relevant chunk parameters can be found in Appendix B. In addition, new chunks have been defined for the transport of encrypted data and packet authentication. They will be described in detail in the next section.

Chunk Type	Chunk Name
0xD0	Secure Session Open Request (SSOpReq) Chunk *
0xD1	Secure Session Certificate (SSCert) Chunk *
0xD2	Secure Session Acknowledge (SSOpReq_Ack) Chunk *
0xD3	Secure Session Server Key (SSSerKey) Chunk *
0xD4	Secure Session Client Key (SSCliKey) Chunk *
0xD5	Secure Session Open Complete (SSOpCom) Chunk *
0xD6	Secure Session Close (SSClose) Chunk *
0xD7	Secure Session Close Acknowledge (SSClose_Ack) Chunk *
0xD8	Security Level Change Chunk *
0xD9	Security Level Change Acknowledge Chunk *
0x10	Encrypted Data (EncData) Chunk **
0x11	Authentication (AUTH) Chunk **
0x12	Padding (PADDING) Chunk **

* – control path ** – data path

Table 5.2: List of new chunk types for S-SCTP

When a S-SCTP session encounters an SCTP implementation not supporting the S-SCTP extension, it sends a error report message but the session will continue normally as insecure SCTP association. In this case, the user can still decide to use one of the standardised security solutions (see chapter 4) because S-SCTP behaves exactly like standard SCTP. [1]

[1] In general, TLS, IPSec and S-SCTP can be combined freely because they do not interfere with each other.

5.3.2 Secure session data path

On the right side of figure 5.4 the new functionalities for data path handling are shown. S-SCTP defines two new blocks in the data path handling: the encryption/decryption block provides message confidentiality and the authentication block provides message integrity and origin authentication. The encryption/decryption block is integrated before bundling to allow to encrypt selected data chunks only. This provides S-SCTP with the ability to mix secure and insecure messages. The authentication block is integrated after bundling to minimise transmission overhead, and an HMAC [41] is computed over the whole SCTP packet including all chunks and the common header (figure 3.1).

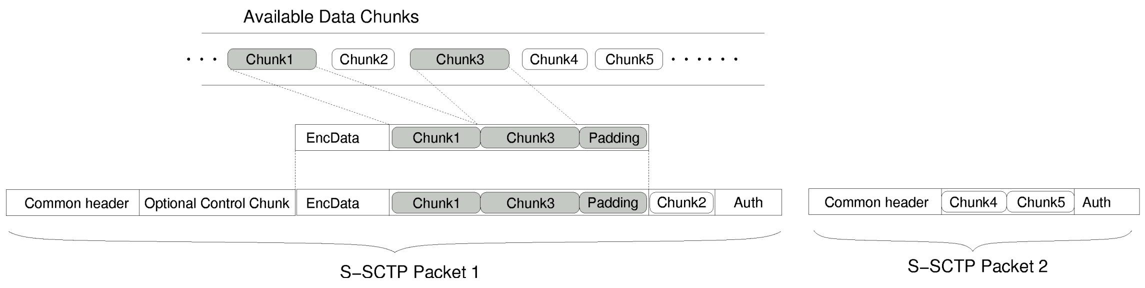


Figure 5.5: S-SCTP packet assembly

Secure data transport

Figure 5.5 shows the packet assembly process in more detail for security level 2 where data chunks are encrypted selectively, because this level requires the most complex processing. Chunks that are available for sending, and which are to be encrypted (see darker chunks in figure 5.5), are grouped together with the newly defined padding chunk to form a newly defined chunk type called Encrypted Data chunk (EncData). Insertion of a PADDING chunk is done depending on data length and encryption block size. The EncData chunk is placed after (unencrypted) control chunks. If space is left in the packet, any data chunks that need not be encrypted can be added (see for example DATA chunk 2 in figure 5.5). On the receiver side, after decryption of the EncData chunk, chunks could be out of sequence. But the SCTP packet disassembly part orders all DATA chunks anyway and delivers them to the upper layer in the right sequence.

After adding the SCTP common header, the HMAC is computed over the whole packet and appended in the newly defined, fixed size Authentication chunk (AUTH chunk) at the end of the packet. This is done before computing the packet checksum and inserting it into the appropriate field in the SCTP common header. Therefore, the checksum is the only field not protected by the HMAC and has to be disregarded when checking the HMAC at the receiver.

The integration of encryption and HMAC calculation clearly optimises the transmission overhead and the computational effort. Moreover, S-SCTP packets are self-contained in the sense that all information required to decrypt and verify them is included in the same packet. The allocation of the cryptographic functions to the packet assembly and disassembly module also fully avoids any problems related to multiple streams and order of delivery, as well as to multiple IP addresses and dynamic address reconfiguration (see section 4.3).

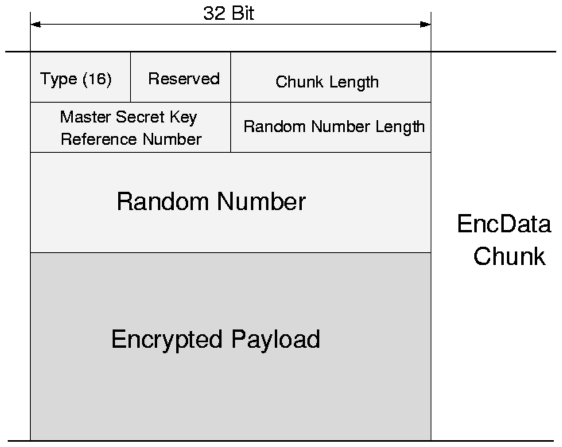


Figure 5.6: Format of the S-SCTP EncData chunk

The format of an EncData chunk[2] is shown in figure 5.6. It contains (in cleartext) the chunk length, a random number and its length in bytes, a reference to a master secret key, and the payload field for the encrypted data. With the reference to the master secret key, the receiver of an EncData chunk is able to decide which set of keys was used for encryption and authentication, since during the lifetime of a secure session the keying material between client and server may be updated several times, each time leading to a different set of master secrets. The initialisation vector needed for Cipher Block Chaining (CBC) encryption and decryption of the payload is derived in a cryptographically secure way from:

[2]Each row in this representation indicates 32 Bit, the rows are transmitted in sequential order with the upper row first

1. a shared master secret which is computed after the secure session setup, and
2. the random number transported in cleartext by the EncData chunk.

A similar approach is used for IPsec's ESP. High quality random numbers of a sufficient length must be used, e.g. 64 bits or longer (see also [22] for necessary properties of random numbers).

The format of PADDING chunks is very simple, it only contains a common chunk header and a random number.

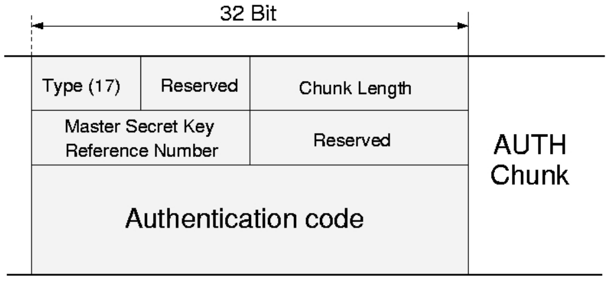


Figure 5.7: Format of the S-SCTP Authentication chunk

The format of an AUTH chunk is shown in figure 5.7. It contains (in cleartext) the chunk length, a reference to a master secret key and the authentication code. The reference to a master secret key is the same as in an EncData chunk.

S-SCTP uses user data compression to transport data before building a data chunk. When user data is compressed the data chunks include a compression flag. In standard SCTP, a data chunk has 5 flag bits reserved and the fourth bit is used for compression. Compression is performed before fragmentation (if necessary).

When the receiver detects a data decryption or a decompression failure then it discards the EncData chunk. When the receiver detects an authentication failure then the whole packet is discarded.

S-SCTP packet format and security levels

The S-SCTP packet format varies slightly at every security level. Figure 5.8 shows all packet formats in the different security levels.

In security level 0, the packet format is same as the standard SCTP packet format.

In security level 1, every outgoing S-SCTP packet must include an AUTH chunk, which is added at the end of the packet. The advantage compared to TLS over SCTP is

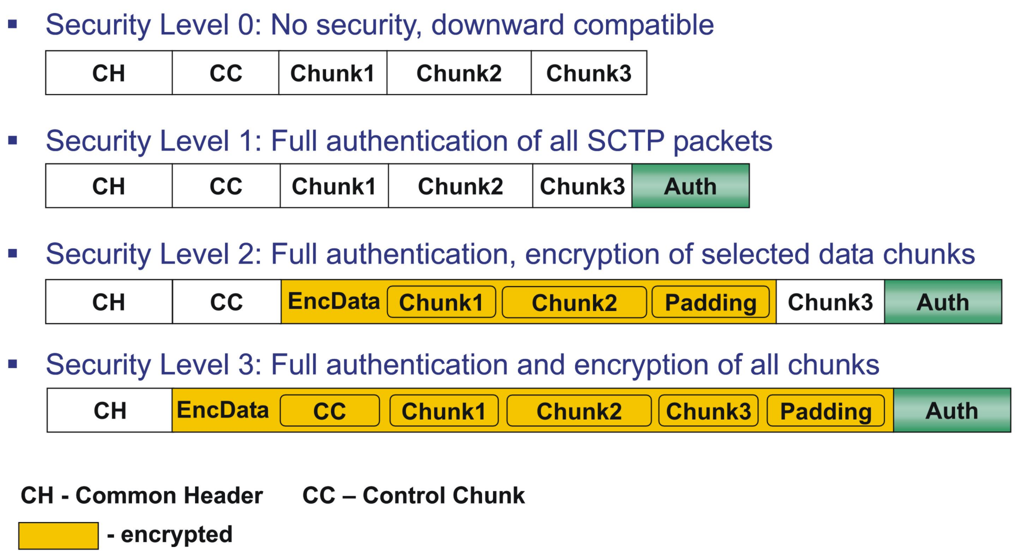

Security Level 0: No security, downward compatible
CH
CC
Chunk1
Chunk2
Chunk3
Security Level 1: Full authentication of all SCTP packets
CH
CC
Chunk1
Chunk2
Chunk3
Auth
Security Level 2: Full authentication, encryption of selected data chunks
CH
CC
EncData
Chunk1
Chunk2
Padding
Chunk3
Auth
Security Level 3: Full authentication and encryption of all chunks
CH
EncData
CC
Chunk1
Chunk2
Chunk3
Padding
Auth
CH - Common Header
CC – Control Chunk
- encrypted

Figure 5.8: Packet format in different security levels

that it also authenticates SCTP control chunks and common header whereas they remain unprotected in TLS.

In security level 2, S-SCTP has two packet formats. One is shown in figure 5.8, and describes the packet if an encrypted data chunk exists. If a packet doesn't include an EncData chunk, then the packet format is the same as in security level 1.

In security level 3, every S-SCTP packet includes both EncData and AUTH chunks. This is the strongest security level and the original information is only visible at the two associated endpoints. For example, when a new IP address is added to an existing association, a third party cannot know this.

5.3.3 Secure session control path

The S-SCTP crypto controller block in the control path includes functions to initialise, close, rekey a secure session and set to the security level.

S-SCTP model and SDL

Specification and Description Language (SDL) diagrams are used to specify S-SCTP procedures. SDL is a standard language for specifying and describing systems. It has been developed and standardised by ITU-T in the recommendation Z.100 [36]. SDL has both a graphical and a textual representation. Both representations are equivalent and can be used at the respective users preference[3]. SDL allows for hierarchical modelling and is, therefore, well suited for a top-down specification approach. Since SDL is based on the extended finite state machine approach, the factor ”time” is difficult to model in SDL. From SDL specifications, the corresponding state transition diagrams as well as message sequence charts representing specific paths through a protocol can be derived. SDL specifications can be used as basis for a formal verification (e.g. reachability analysis), for performance simulation (with temporal extensions) and for non-exhaustive, semi-automatic testing and validation.

In this project, the SDL suite from Telelogic Tau [81] was used to design the S-SCTP model. This SDL tool offers a rich set of options for simulation, for verification and for validation of the SDL specification.

Figure 5.9 shows an SDL overview of the S-SCTP model. The S-SCTP model defines message groups for the communication between the function blocks. The groups include messages for requests from (A/B_Request) and indications (A/B_Indication) to the S-SCTP user as well messages exchanged (A/B_Signal) between endpoints.

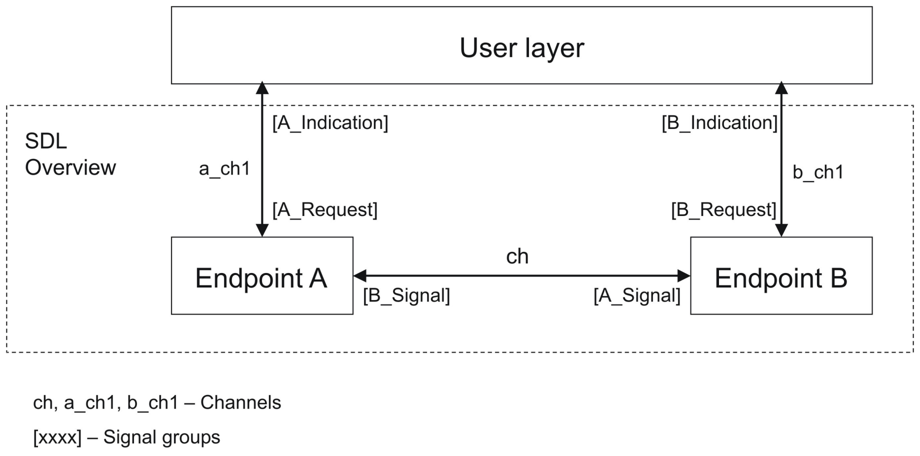


Figure 5.9: SDL overview of S-SCTP model

Since the protocol is symmetric, both endpoints have the same SDL specifications, which are given in detail in Appendix A. Sixteen pages of S-SCTP SDL diagram define all S-SCTP procedures (e.g. secure session intialisation, deletion, update ...), which are described in next sections. Using Telelogic software, the S-SCTP functionality has been

[3]The textual representation allows for a dense representation of large protocols, whereas the graphical representation is useful, e.g. for visual inspections of all possible transitions originating in one specific state.

specified to the level of detail required to simulate the complete protocol. The S-SCTP SDL specification generates the state-machine shown in figure 5.10. By simulation it was possible to test every state of the model according to A/B_Request messages from a user.

Using Telelogic software, the S-SCTP functionality has been specified in SDL to the level of detail required to simulate the complete protocol. I defined all states and all allowed state transitions. I also specified all variables and timers which are required for the transactions and states. Tasks, however, have not been fully specified. By using the S-SCTP SDL specification, the state machine shown in figure 5.10 can be generated. I used the SDL model for a basic validation of the protocol by visually inspecting the state machine to identify missing and unnecessary transitions. Furthermore, I simulated the protocol behaviour by manually generating input events and checking the resulting state transitions for correctness.

Secure session initialisation

The first phase of the setup of an S-SCTP association is basically the same as for a normal SCTP association (see e.g. section 5 of [75] for a description), because the secure session negotiation only starts after the basic association has been established. Any SCTP user can start a secure session negotiation at any time, allowing the dynamic creation and deletion of secure sessions.

To negotiate the secure session and its parameters, an initiating SCTP client (called Endpoint A) issues a request primitive indicating, among other parameters, the security level and the cryptographic algorithms to be used. The information relevant to the secure session negotiation is exchanged by means of the appropriate, newly defined SCTP control chunks (see table 5.2), which can be bundled together with regular chunks. The handshake is supervised by a timer. If this timer expires or if there are other errors in the handshake, the secure session negotiation is aborted and Endpoint A is notified. The negotiation is done in a four-way handshake as shown in figure 5.11.

A secure session intialisation begins when one of the associated endpoints sends the secure session Open Request (SSOpReq) message to the server (called Endpoint B). The chunk type is chosen such that a peer implementation not supporting S-SCTP will report an error and otherwise ignore this chunk. In that case, a failure is reported to the user (or application) and the – insecure – SCTP association is continued. The chunk contains a list with crypto-algorithms and compression methods supported by the client (called Endpoint A). Endpoint A should present its certificate to Endpoint B, which is an X.509v3 [35] certificate as is also used for TLS. This allows for re-using an already existing public key infrastructure. The Secure Session Certificate (SSCert) control message includes the certificate and it is optional.

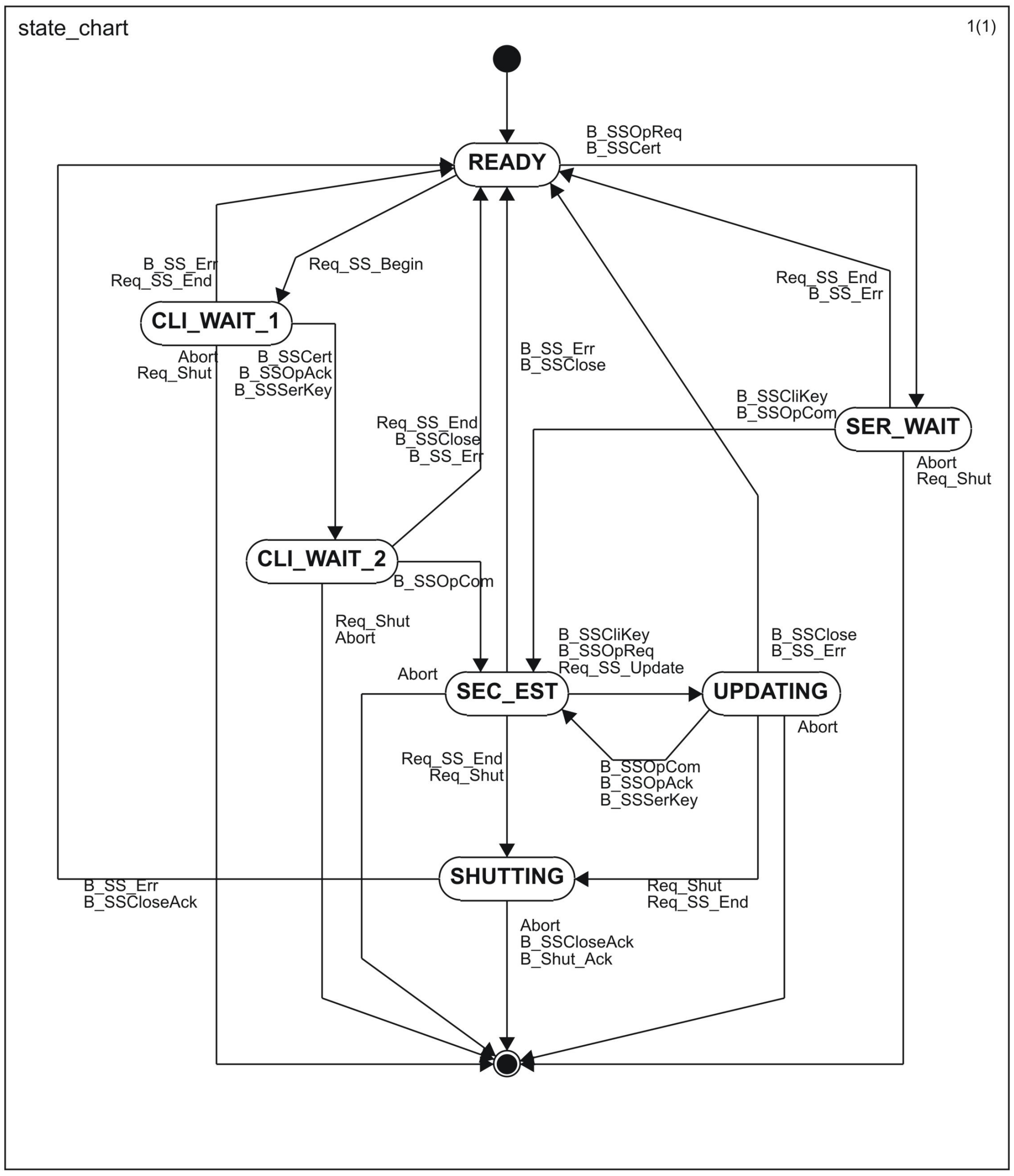


Figure 5.10: S-SCTP state machine

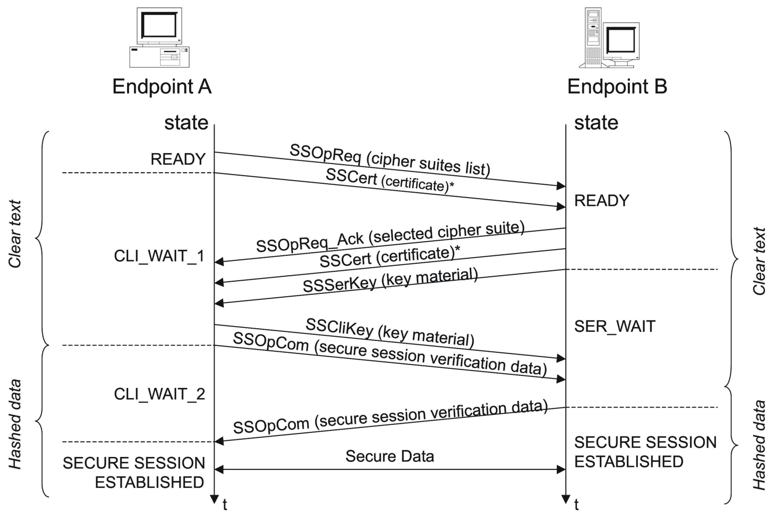


Figure 5.11: S-SCTP secure session negotiation

When both associated endpoints request the initialisation of the secure session simultaneously by sending SSOpReq messages and changing the state from READY to CLI_WAIT_1, then both ignore the received SSOpReq message and wait for a random time before resending the SSOpReq message. Each endpoint generates the random time independently.

After having received these messages, Endpoint B sends a Secure Session Open Request Acknowledged (SSOpReq_Ack) message to Endpoint A. SSOpReq_Ack acknowledges the secure session initiation by Endpoint A and indicates the crypto algorithm that Endpoint B has chosen. This identifies the algorithm used for asymmetric key exchange (e.g. Diffie-Hellman, DH), symmetric data encryption (e.g. 3DES with cipher block chaining) and authentication (e.g. secure hash algorithm, SHA). S-SCTP implementations must at least support DH_with_3DES_CBC_SHA at each endpoint. The encryption algorithm may be NULL, indicating that only authentication is used to ensure data integrity. Endpoint B also presents its certificate if it has one.

Then Endpoint B generates key material depending on the negotiated key exchange algorithm. This key material is used to compute the master secret keys. Endpoint B sends the Secure Session Server Key (SSSerKey) message including the newly generated key material and changes its state from READY to SER_WAIT.

After receiving these last 3 messages Endpoint A sends the Secure Session Client Key (SSCliKey) message which is similar to SSSerKey. Both Endpoint A and B key materials are used to compute the master secret keys which are described in Appendix B.

Then Endpoint A generates a Secure Session Open Complete (SSOpCom) message which contains a HMAC of all messages received before, and a HMAC computation on the newly generated master secret key. Finally, Endpoint A after sending the SSOpCom message moves to CLI_WAIT_2.

Endpoint B computes the master secret keys after receiving the SSCliKey message. Then Endpoint B verifies the SSOpCom message from Endpoint A and checks whether all messages from the server to the client have been exchanged correctly. If the verification is successful, Endpoint B generates the SSOpCom message, sends the message to Endpoint A and establishes the secure session. Otherwise Endpoint B sends an error message to the peer and changes its state to READY.

S-SCTP has an internal parameter which indicates whether an endpoint is a server or a client. This parameter is used to solve conflicts if both endpoints subsequently initialise the same procedure at the same time. After moving to the SEC_EST state, Endpoint B sets this parameter to 1 (ser=1).

When Endpoint A receives the SSOpCom message from Endpoint B it establishes the secure session after using the same verification procedure as Endpoint B for the message. After moving to the SEC_EST state, Endpoint A sets the parameter ”ser” to 0.

Secure session close

The closing of a secure session begins when the secure session has been established and one of the associated endpoints sends a Secure Session Close (SSClose) message (Table 5.2) to its peer, as is shown in figure 5.12.

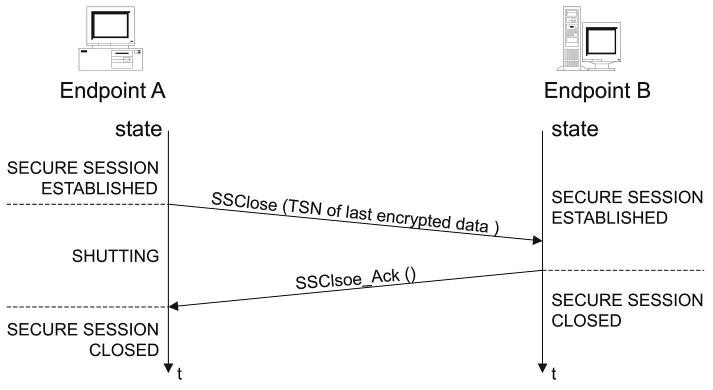


Figure 5.12: S-SCTP secure session close

This message includes a cumulative transmission sequence number (TSN) of encrypted data messages with the highest encrypted TSN that it is still outstanding (or a flag, if no more encrypted data is outstanding).

The peer, after having received the SSClose message, returns a Secure Session Close Acknowledge (SSClose_Ack) message as soon as all outstanding encrypted data (as indicated by the TSN of the secure session close message) has been acknowledged.

The encryption or authentication of new outgoing data is not allowed after sending or receiving SSClose or SSClose_Ack messages. If the current security level of endpoints is changed to 0 after the establishment of a secure session the peer sends the SSClose message with authentication.

A S-SCTP secure session could also be terminated by other reasons than explained before. The endpoint closes the secure session immediately after receiving a secure session error. S-SCTP defines several new error codes in more detail, as described in Appendix B.

Secure session rekeying

The secure session has a parameter controlling the lifetime of the keying material, which can be set by an application or is set to a default value defined by the implementation. Upon expiration of this lifetime, re-keying is automatically triggered. Since there already exists a secure session, the re-keying mechanism uses an abbreviated handshake (figure 5.13).

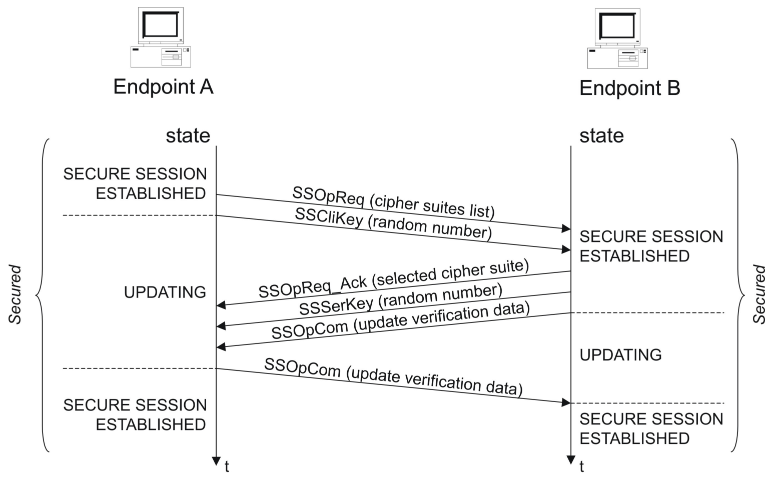

Endpoint A
Endpoint B
state
state
SECURE SESSION ESTABLISHED
SSOpReq (cipher suites list)
SSCliKey (random number)
SECURE SESSION ESTABLISHED
UPDATING
SSOpReq_Ack (selected cipher suite)
SSSerKey (random number)
SSOpCom (update verification data)
UPDATING
SSOpCom (update verification data)
SECURE SESSION ESTABLISHED
SECURE SESSION ESTABLISHED
Secured
Secured
t
t

Figure 5.13: S-SCTP secure session rekeying

The abbreviated handshake messages are all encrypted and authenticated. In other words, security level 3 is used in the re-keying mechanism. The control messages used in the abbreviated handshake are similar to those used in a secure session initialisation. In the abbreviated handshake, S-SCTP uses the RSA key exchange algorithm.

The secure session re-keying begins when one of the associated endpoints sends a SSOpReq control message (Table 5.2) to its peer. After the re-keying mechanism, a secure session can use a new encryption and authentication algorithm. When both Endpoint A and Endpoint B send an SSOpReq message simultaneously, Endpoint A ignores Endpoint B's SSopReq message based on the status of the internal "ser" parameter (ser=0). Endpoint B accepts the SSOpReq message from Endpoint A knowing it is the server (ser=1). While the meaning of the following messages is the same as in the secure session initialisation, the exchange procedure of these messages is slightly different.

5.4 Prototype implementation

The realisation of S-SCTP requires – in addition to the definition of the new chunk types and procedures – modifications and extensions of the various modules of the SCTP implementation (see figure 5.4). The detailed draft specification of S-SCTP defining types and encoding of new chunks can be found in Appendix B. The S-SCTP implementation is based on our prototype socket API, sctplib implementation [19] and the OpenSSL crypto library [23].

5.4.1 Starting point

In our group, a fully featured SCTP implementation has already been developed in co-operation with an industrial partner and is available under the GNU public license [27].

The SCTP implementation

Our sctplib implementation runs under the operating systems Linux, FreeBSD, and MAC OS. It is implemented in userland rather than in the kernel. The advantage of the userland implementation is that it is easy to observe and modify it. However users of this implementation should have the root privileges.

Figure 5.14 shows the functional structure of the sctplib implementation. SCTP defines Upper Layer Protocol to SCTP (ULP-to-SCTP) and SCTP-to-ULP interfaces, which are briefly described in the following.

ULP-to-SCTP. ULP-to-SCTP primitives deliver upper layer requests to SCTP. The primitives used in this section are defined in section 10.1 of RFC2960 [75]. The text written in

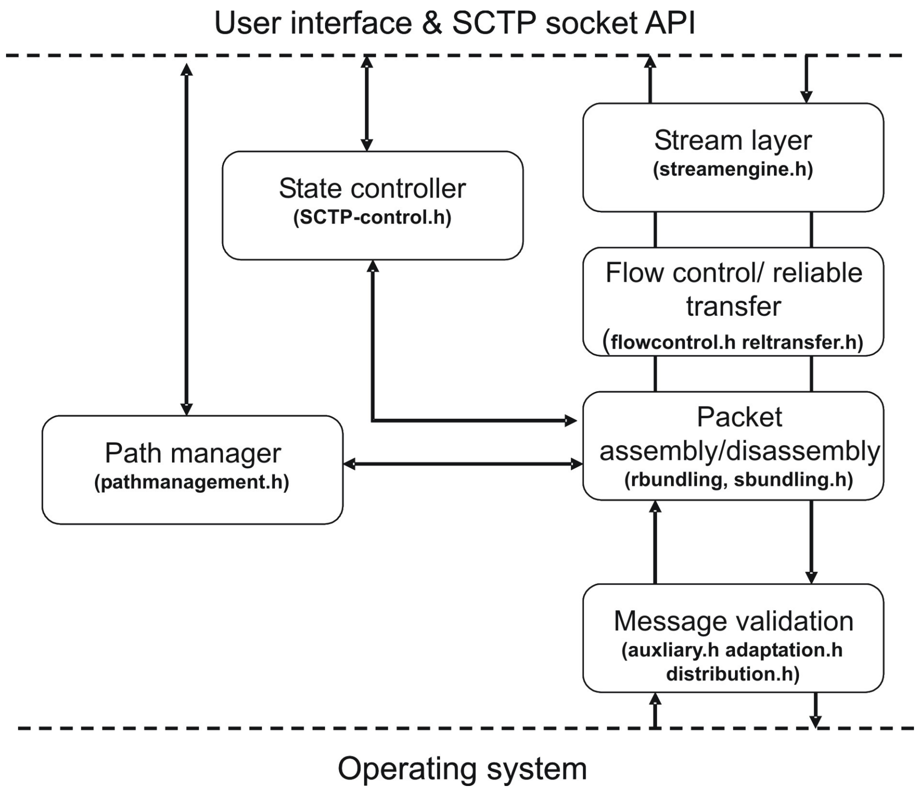

User interface & SCTP socket API
Stream layer
(streamengine.h)
State controller
(SCTP-control.h)
Flow control/ reliable transfer
(flowcontrol.h reltransfer.h)
Packet assembly/disassembly
(rbundling, sbundling.h)
Path manager
(pathmanagement.h)
Message validation
(auxliary.h adaptation.h distribution.h)
Operating system

Figure 5.14: Functional structure of the SCTP implementation

italic indicates the respective functions that have been developed in our implementation. All ULP-to-SCTP primitives described below are defined in the sctp.h header file.

INITIALIZE: This primitive initialises SCTP internal data structures and allocates necessary resources (*sctp_initLibrary()*).

ASSOCIATE: This primitive allows the upper layer to initiate a new association (*sctp_associate()*).

SEND: This primitive sends user data via SCTP (*sctp_send()*).

RECEIVE: This primitive reads user data from a buffer which is specified by the Upper layer protocol (*sctp_receive()*).

SETPRIMARY: This primitive allows to set a specified destination transport address as the primary path (*sctp_setPrimary()*).

SHUTDOWN: This primitive closes down an association gracefully (*sctp_shutdown()*).

ABORT: This primitive closes an association immediately (*sctp_abort()*).

STATUS: This primitive requests information about an association (*sctp_getAssocStatus()*).

SCTP-to-ULP. SCTP defines notifications to deliver information to the upper layer. The notifications described in this section are defined in section 10.2 of RFC2960 [75]. The text written in italic indicates the respective functions that have been developed in our implementation. All notifications are defined in the sctp.h header file.

DATA ARRIVE: This notification indicates user data successfully received and ready for retrieval (*dataArriveNotif()*).

COMMUNICATION UP: This notification indicates that SCTP is ready to send or receive user data (*communicationUpNotif()*).

COMMUNICATION LOST: This notification indicates that an endpoint has lost a communication (*communicationLostNotif()*).

SEND FAILURE: This notification indicates that user data cannot be delivered (*sendFailureNotif()*).

NETWORK STATUS CHANGE: This notification indicates whether the destination transport address is marked active or inactive (*networkStatusChangeNotif()*).

COMMUNICATION ERROR: This notification indicates when SCTP receives error chunks from its peer (*communicationErrorNotif()*).

RESTART This notification indicates when SCTP detects that the peer has restarted (*restartNotif()*).

SHUTDOWN COMPLETE: This notification indicates that SCTP completes the shutdown procedure (*shutdownCompleteNotif()*).

Besides our implementation, there are several other SCTP implementations. The FreeBSD team implemented SCTP in the FreeBSD kernel. Nokia, Motorola and IBM developed LKSCTP, which is a Linux kernel SCTP implementation. Cisco developed its own IOS SCTP implementation, which is integrated in the Cisco OS. AIX was developed by IBM, who implemented SCTP in their own AIX-Unix system. Our sctplib has been successfully tested for interoperability with these other SCTP implementations in six official interoperability workshops.

SCTP socket API implementation

Our socket API implementation has been defined according to the SCTP socket API draft defined in [78]. It is written in C++ and works under Linux. SCTP socket API functions' purposes and formats are similar to TCP Socket API functions. All functions are defined in the ext_socket.h header file, and the new API functions begin with ext_... . For example, in TCP or UDP implementations, calls are made to the socket() function to create a new TCP or UDP socket; in our implementation, calls are made to the ext_socket() function to create a new SCTP socket. The socket API [78] supports two kinds of socket styles:

- One-to-one style interface: a relationship of one socket descriptor to one association.
- One-to-many style interface: a relationship of one socket descriptor to many associations.

Clearly, the one-to-one style interface is a TCP-like interface, and the one-to-many style interface is a UDP-like interface.

The only difference between SCTP and TCP socket calls is that SCTP uses IPPROTO_SCTP instead of IPPROTO_TCP which is used in the TCP socket call. The functions used on both server and client side are defined in table 5.3. Several socket calls have the same functions as shown in table 5.3, but they use different parameters, e.g.

ext_send(socket descriptor, buffer, length, flags);

ext_sendto(socket descriptor, buffer, length, flags, address, address length);

The differences between SCTP and UDP socket calls are that while SCTP uses the IPPROTO_SCTP protocol type and SOCK_SEQPACKET communication type, UDP uses the IPPROTO_UDP protocol type and SOCK_DGRAM communication type. The functions used on both server and client side are also defined in table 5.3. More details about the above socket styles and new options of socket functions are defined in [78].

SCTP uses the following new options in ext_setsockopt() and ext_getsockopt() functions:

SCTP_RTOINFO: This option is used to initialise and bound retransmission timeout (RTO).

SCTP_ASSOCINFO: This option is used to both examine and set various association and endpoint parameters.

SCTP_INITMSG: This option is used to modify protocol parameters. For example, the user can define which stream numbers to use.

SCTP_AUTOCLOSE: This option is only applicable to the UDP-like socket. When this option is set, an association is automatically closed after a certain idle time.

SCTP_SET_PEER_PRIMARY_PATH: This option is used in the dynamic address reconfiguration extension to request that the peer marks the enclosed address as the association primary.

SCTP_PRIMARY_ADDR: This option is used in the dynamic address reconfiguration extension to request that the local SCTP stack uses the enclosed peer address as the association primary.

SCTP_ADAPTATION_LAYER: This option is used to request the local endpoint to set the specified adaptation layer indication parameter for all future INIT and INIT-ACK exchanges.

SCTP_DISABLE_FRAGMENTS: This option switches the fragmentation on and off. Data bigger than the current MTU is not transported and an error is indicated if fragmentation is off.

SCTP_PEER_ADDR_PARAMS: This option is used to enable or disable heartbeats for any peer address of an association, modify an address's heartbeat interval and force a heartbeat to be sent immediately.

SCTP_DEFAULT_SEND_PARAM: This option allows an application to set the default sctp_sndrcvinfo structure. The sctp_sndrcvinfo structure is defined in section 5.2.2 of [78].

SCTP_EVENTS: This option is used to specify various notifications and ancillary data the user wishes to receive. The SCTP notifications are defined in section 10.2 of RFC2960 [75].

SCTP_STATUS: This option is read-only. Applications can retrieve current status information about an association, including association states, peer receiver window size, number of unacknowledged chunks and number of data chunks pending receipt.

SCTP_GET_PEER_ADDR_INFO: This option is also read-only. Applications can retrieve information about a specific peer address of an association, including its reachability state, congestion window and retransmission timer values.

SCTP_I_WANT_MAPPED_V4_ADDR: This option is used to turn on or off mapped IPv4 addresses. This flag is used to map IPv4 address to IPv6 address.

Calls	Description	TCP like		UDP like	
		server side	client side	server side	client side
ext_socket()	This function is used to create a new socket descriptor	yes	yes	yes	yes
ext_bind()	This function is used to assign a local address to a socket	yes	no	yes	no
ext_listen()	This function is used for accepting inbound association	yes	no	yes	no
ext_accept()	This function accepts a newly established SCTP association	yes	no	no	no
ext_connect()	This function is used to initiate an association to peer	no	yes	no	no
ext_close()	This function is used to gracefully close down an association	yes	yes	yes	yes
ext_write()	This function used to transmit data	yes	yes	yes	yes
ext_read()	This function is used to receive data	yes	yes	yes	yes
ext_send()	This function is used to transmit data	yes	yes	yes	yes
ext_recv()	This function is used to receive data	yes	yes	yes	yes
ext_sendto()	This function is used to transmit data	yes	yes	no	no
ext_recvfrom()	This function is used to receive data	yes	yes	no	no
ext_sendmsg()	This function is used to send data	yes	yes	no	no
ext_recvmsg()	This function is used to receive data	yes	yes	no	no

Table 5.3: Socket calls

New interfaces. According to the document [78] for the socket API, the following interfaces that can be implemented as system calls or functions.

sctp_bindx(): This function is an extension of the ext_bind() call. An application can use this function to associate additional IP addresses with an endpoint, after calling ext_bind().

sctp_peelof(): This function is used to branch off an association into a separate socket.

sctp_getpaddrs(): This function returns all peer addresses in an association. After calling this function, resources are reserved for addresses.

sctp_sctp_freepaddrs(): This function releases all resources allocated by sctp_getpaddrs.

sctp_getladdrs(): This function returns all local bound addresses on a socket. After calling this function, resources are reserved for addresses.

sctp_freeladdrs(): This function releases all resources allocated by sctp_getladdrs.

sctp_connectx(): This function allows an endpoint to be associated with multiple addresses.

OpenSSL implementation

I analysed the existing TLS open source implementations which are described in section 3.2.1 and as a result I have chosen OpenSSL to implement TLS over SCTP. All TLS implementations have different purposes. To be suitable for my project, a TLS implementation has to be flexible, easy to use, focused on SSL and TLS development and functionalities (e.g. support most ciphers), well documented and widely used. Based on these criteria, the various implementations can be characterised as follows:

- The Cryptolib implementation is a multi-purpose, powerful secure toolkit, and hence is not exclusively focused on TLS development. There are already existing applications using this implementation. For example, in SSH sessions, PGP or S/MIME.
- The yaSSL implementation is dedicated to SSL but the implementation's documentation is poor, and not many applications use it. The advantage is that it is very fast.
- The PureTLS implementation is a Java-only implementation which supports few cipher suites from the cipher suite list defined in [21].

- The GnuTLS implementation is better than the above implementations. It uses the gnu crypto library. However, this library is not as well developed as, e.g., the OpenSSL crypto library.
- OpenSSL is only focused on SSL and TLS development. It has rich functional possibilities, which allow much more flexible programming. Moreover, it includes its own crypto libraries, which provide strong crypto functionalities. OpenSSL is the most widely-used implementation and most Linux kernels support it. The OpenSSL crypto library has been developed separately and the crypto functions are not integrated in SSL. Therefore users can use just the crypto library of OpenSSL without the SSL library.

For the reasons mentioned above, OpenSSL was chosen for my TLS over SCTP implementation.

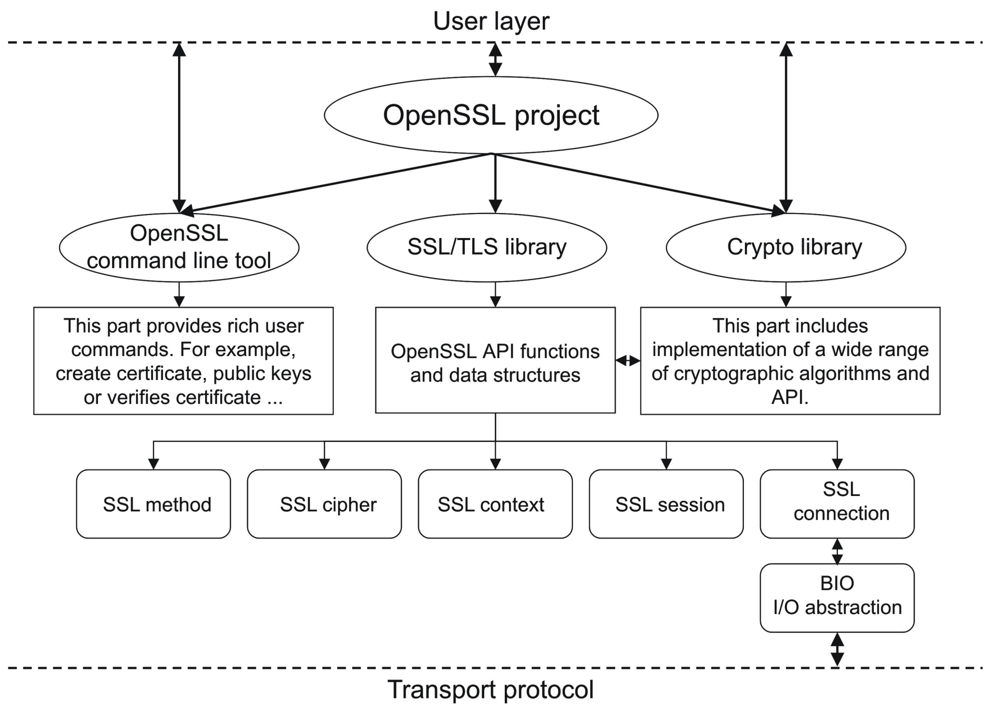

User layer
OpenSSL project
OpenSSL command line tool
SSL/TLS library
Crypto library
This part provides rich user commands. For example, create certificate, public keys or verifies certificate ...
OpenSSL API functions and data structures
This part includes implementation of a wide range of cryptographic algorithms and API.
SSL method
SSL cipher
SSL context
SSL session
SSL connection
BIO I/O abstraction
Transport protocol

Figure 5.15: Architecture of the OpenSSL implementation

Figure 5.15 shows the architecture of the OpenSSL implementation. The SSL/TLS library includes five data structures. SSL method contains information about SSL versions (e.g. SSL version 2, 3 and TLS version 1) and their structures. SSL cipher contains information about crypto algorithms. SSL context contains default values of the currently established SSL structure. SSL session contains detailed information of the current SSL connection. The SSL connection is a SSL structure that is created by a server or client per established connection. This actually is the core structure in the SSL API. At run-time the application usually deals with this structure which has links to nearly all other structures. A BIO is an input/output (I/O) abstraction, that can hide many of the underlying I/O details from an application. If an application uses a BIO for its I/O it can transparently handle SSL connections, unencrypted network connections and file I/O.

5.4.2 S-SCTP implementation

The S-SCTP implementation developed in this project is based on our prototype sctplib and socket API implementations. For the cryptographic functions, I used the cryptographic part of the OpenSSL library to avoid recoding of the crypto algorithms. As a default, I used the DH_with_3DES_CBC_SHA cipher suite, but similarly to TLS and IPsec, our concept allows flexible addition and selection of various cipher suites. All cipher suites are defined in Appendix B. The S-SCTP prototype implementation is written in C, as the sctplib. Several new header files such as ssctp.h, cryptocrtl.h and ssctp_messages.h have been defined in the sctplib implementation. The header file ssctp.h contains all the new functions of initialisation, deletion, and rekeying of a secure session. While the header file cryptocrtl.h provides encryption, decryption, compression, decompression and authentication of messages, the header file ssctp_messages.h contains all data structures used in a secure session.

ULP to S-SCTP

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 RFC2960. The text written in italic indicates the respective functions that have been developed in my implementation. 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)* $\rightarrow$ *result*

- secure session ID: local handle to the secure session
- security level: This parameter indicates the new security level

GETSECLEVEL: This pritimive gets the current security level of a secure session.
Format: *getSecLevel(secure session ID)* $\rightarrow$ *security level*

- secure session ID: local handle to the secure session

SENDSEC: This primitve 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)* $\rightarrow$ *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 RFC2960 part 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)* $\rightarrow$ *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)* $\rightarrow$ *result*

- secure session ID: local handle to the secure session.
- time: The new lifetime in seconds.

SHUTSECSESS: This primivite deletes a secure session.
Format: *shutSecSess(secure session ID)* $\rightarrow$ *result*

- secure session ID: local handle to the secure session.
- security level: This parameter indicates the new security level.

S-SCTP to ULP

S-SCTP defines new notifications to deliver information to the upper layer. The notifications extend the section 10.2 of RFC2960 [75]. The text written in italic indicates the respective functions that have been developed in my implementation. 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 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 to compress user data. On the receiver side there are no changes.

Extension of Transmission Control Block (TCB)

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 RFC2960.

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 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.

Extension of the socket API implementation

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 doesn't exist.
- sec_status: This parameter contains a security status. This parameter MUST be zero when a secure session doesn't 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.

Status of the S-SCTP implementation

Due to the limited project duration, it was not possible to completely finalise the implementation and priorities had to be defined. Therefore, I focused on the implementation of the data path in order to be able to perform the performance studies described in chapter 5.2.3. As a consequence, the data path of S-SCTP is fully implemented whereas the dynamic secure session management has only partially been realised and the S-SCTP sessions have to be configured manually.

It should be mentioned that the SDL specification is complete and, therefore, the implementation of the control path would be straightforward. Due to the incomplete control path implementation, no performance measurement could be performed for secure session management.

Chapter 6

Quantitative evaluation of the data path of S-SCTP

In this chapter I will evaluate the standard security solutions and S-SCTP in an experimental setup. Since IPsec secure session management cannot be controlled dynamically, it is not possible to perform a meaningful comparison of the mechanisms with respect to the control path performance.
Furthermore, most of the application scenarios use rather static scenarios with long lived associations.
Therefore, the experiments in this chapter focus on the data path performance.

6.1 Testbed description

The testbed used a simple local network in a single-homing configuration, which contained two Linux based computers and a switch (see figure 6.1). There are no significant differences between single-homing and multi-homing when measuring the data path performance of the different solutions. In all test scenarios the same testbed was used.

The two computers had 100mbit/s Ethernet cards and they are connected through a 100mbit/s switch. The first computer, with IP address 132.252.150.149 had an Athlon AMD 1333MHz processor and the second computer, with IP address 132.252.150.157, had a Pentium 3 600MHz processor.

I developed a measurement setup for the experiments which is shown in figure 6.2. The traffic generator is a simple application which is running on the computer 1. This application continuously generates random data and sends this data to the computer 2. The size of the messages depends on the experiment scenarios and is constant during a measurement.

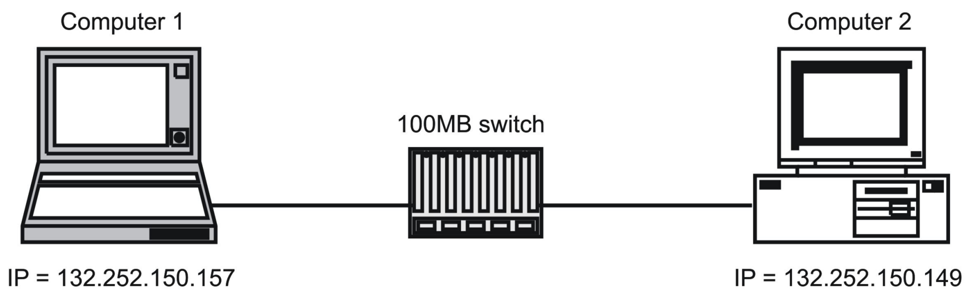


Figure 6.1: Configuration of testbed

The traffic analyser (figure 6.2) is also a simple application which is running on the computer 2. The role of the traffic analyser is receive the random data sent from computer 1 and calculate the message throughput in different scenarios for each security solution. The message throughput is calculated based on the received data messages without any overheads (e.g. SCTP packet header, encryption header, authentication header). The traffic analyser calculates a throughput value for every second and the measurement continues 5 minutes. This means the traffic analyser produces 5*60=300 data points for measurement. I used the average of all measurements in the evaluation. For the calculation of the average, the first 20 values were not used because SCTP uses a slow start algorithm (see section 7.2 of [75]) for data transmission after establishment an association. The throughput in the slow start phase is much less than the sustained average throughput.

6.2 Monitoring SCTP security solutions

I used the Ethereal open source program to monitor the SCTP security solutions. Ethereal can be used for troubleshooting, analysis, software and protocol development. It has all of the standard features one would expect in a protocol analyser, and several features not seen in any other product. Its open source license allows to add enhancements. It runs on all popular computing platforms, including Unix, Linux, and Windows (more details in www.ethereal.com).

Ethereal uses a port number or protocol payload ID (PPID) to recognise a payload of SCTP. The Internet Assigned Number Authority (IANA) standarisation organisation defines these port numbers and PPIDs (e.g. ASAP PPID is 11). Currently, IANA has not yet defined a TLS PPID and port number for SCTP. Without this, TLS over SCTP is invisible – we can decode only SCTP but not the TLS payload. For my experiment, I therefore defined a TLS payload protocol ID for SCTP and integrated it into Ethereal.

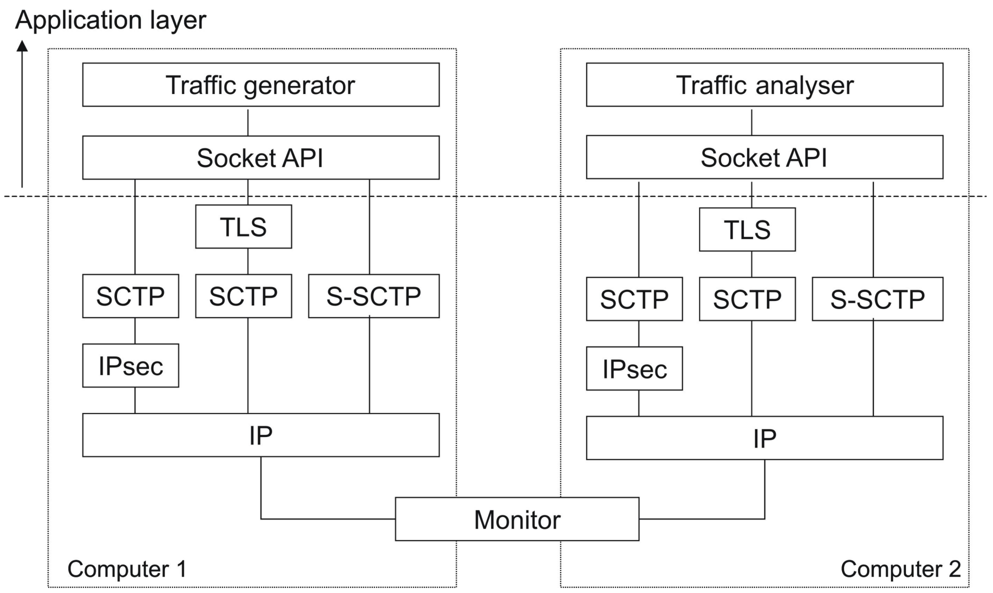


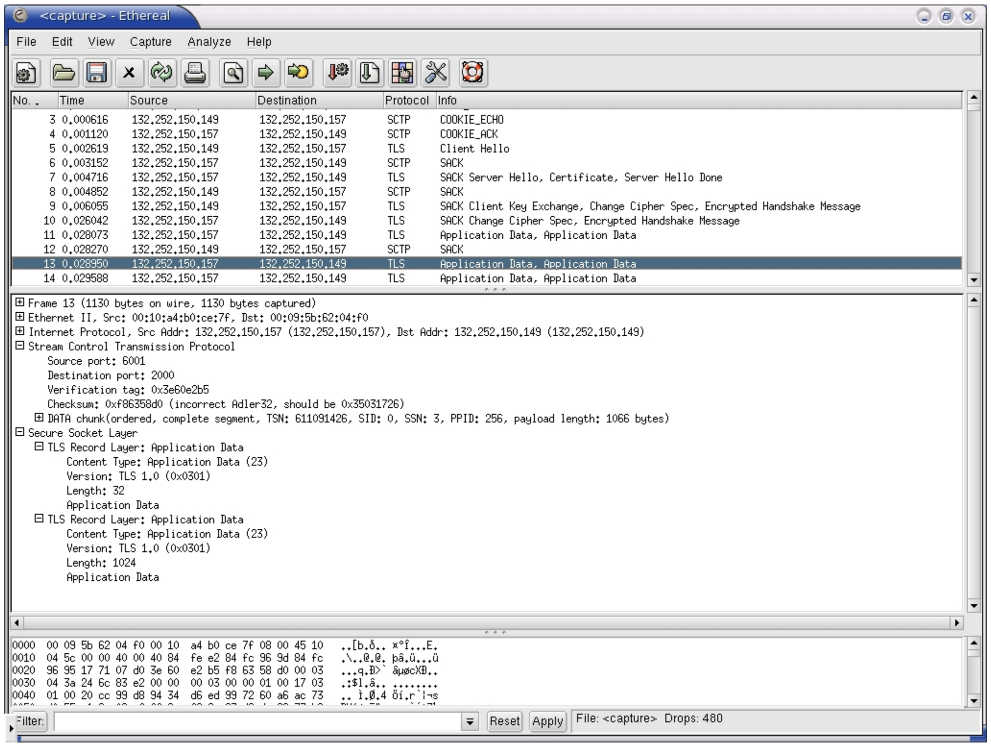
Figure 6.2: Measurement setup

The TLS PPID is defined in sctpppids.h, which contains all PPIDs of protocols which are using SCTP. Figure 6.3 shows a screenshot of a fully decoded packet for TLS over SCTP. For SCTP over IPsec Ethereal doesn't need any additional changes.

To monitor S-SCTP transport I integrated the AUTH and EncData chunk codes of S-SCTP into Ethereal. Figure 6.4 shows decoded AUTH and EncData chunks. Because the implementation of the S-SCTP control path was not yet fully functional, the other S-SCTP control chunks have not been integrated into Ethereal. However, their integration is straight-forward.

6.3 Setup of reference scenarios

This section describes the experimental setup of the standardised SCTP security solutions. While the SCTP over IPsec scenario required only configuration of the standard protocols, adaptations and extensions of the OpenSSL implementation were necessary for the TLS over SCTP scenario, extensions provide transport of TLS data over SCTP. The following two paragraphs describe the configuration of both security solutions.

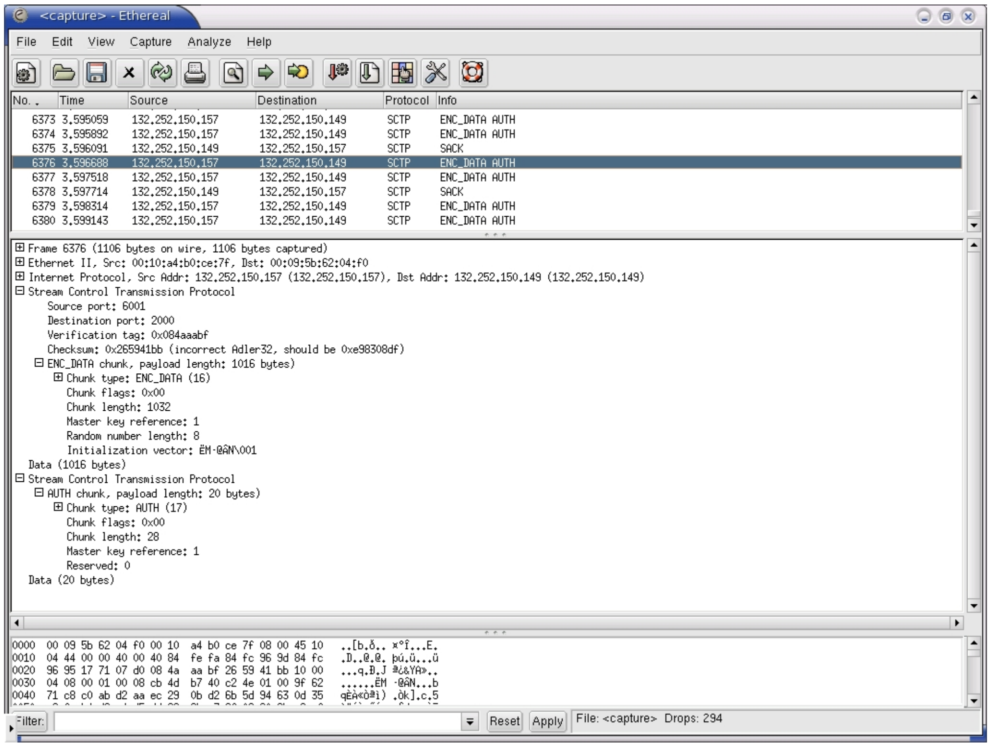


```
No.  Time      Source           Destination      Protocol Info
   3 0.000616  132.252.150.149  132.252.150.157  SCTP     COOKIE_ECHO
   4 0.001120  132.252.150.157  132.252.150.149  SCTP     COOKIE_ACK
   5 0.002619  132.252.150.149  132.252.150.157  TLS      Client Hello
   6 0.003152  132.252.150.157  132.252.150.149  SCTP     SACK
   7 0.004716  132.252.150.157  132.252.150.149  TLS      SACK Server Hello, Certificate, Server Hello Done
   8 0.004852  132.252.150.149  132.252.150.157  SCTP     SACK
   9 0.006055  132.252.150.149  132.252.150.157  TLS      SACK Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
  10 0.026042  132.252.150.157  132.252.150.149  TLS      SACK Change Cipher Spec, Encrypted Handshake Message
  11 0.028073  132.252.150.157  132.252.150.149  TLS      Application Data, Application Data
  12 0.028270  132.252.150.149  132.252.150.157  SCTP     SACK
  13 0.028950  132.252.150.157  132.252.150.149  TLS      Application Data, Application Data
  14 0.029588  132.252.150.157  132.252.150.149  TLS      Application Data, Application Data

⊞ Frame 13 (1130 bytes on wire, 1130 bytes captured)
⊞ Ethernet II, Src: 00:10:a4:b0:ce:7f, Dst: 00:09:5b:62:04:f0
⊞ Internet Protocol, Src Addr: 132.252.150.157 (132.252.150.157), Dst Addr: 132.252.150.149 (132.252.150.149)
⊟ Stream Control Transmission Protocol
    Source port: 6001
    Destination port: 2000
    Verification tag: 0x3e60e2b5
    Checksum: 0xf86358d0 (incorrect Adler32, should be 0x35031726)
  ⊞ DATA chunk(ordered, complete segment, TSN: 611091426, SID: 0, SSN: 3, PPID: 256, payload length: 1066 bytes)
⊟ Secure Socket Layer
  ⊟ TLS Record Layer: Application Data
      Content Type: Application Data (23)
      Version: TLS 1.0 (0x0301)
      Length: 32
      Application Data
  ⊟ TLS Record Layer: Application Data
      Content Type: Application Data (23)
      Version: TLS 1.0 (0x0301)
      Length: 1024
      Application Data

0000  00 09 5b 62 04 f0 00 10  a4 b0 ce 7f 08 00 45 10   ..[b.ð.. ¤°Î...E.
0010  04 5c 00 00 40 00 40 84  fe e2 84 fc 96 9d 84 fc   .\..@.@. þâ.ü...ü
0020  96 95 17 71 07 d0 3e 60  e2 b5 f8 63 58 d0 00 03   ...q.Ð>` âµøcXÐ..
0030  04 3a 24 6c 83 e2 00 00  00 03 00 00 01 00 17 03   .:$l.â.. ........
0040  01 00 20 cc 99 d8 94 34  d6 ed 99 72 60 a6 ac 73   .. Ì.Ø.4 Öí.r`¦¬s
```

Figure 6.3: Network capture of TLS over SCTP



```
No.  Time      Source           Destination      Protocol Info
6373 3.595059  132.252.150.157  132.252.150.149  SCTP     ENC_DATA AUTH
6374 3.595892  132.252.150.157  132.252.150.149  SCTP     ENC_DATA AUTH
6375 3.596091  132.252.150.149  132.252.150.157  SCTP     SACK
6376 3.596688  132.252.150.157  132.252.150.149  SCTP     ENC_DATA AUTH
6377 3.597518  132.252.150.157  132.252.150.149  SCTP     ENC_DATA AUTH
6378 3.597714  132.252.150.149  132.252.150.157  SCTP     SACK
6379 3.598314  132.252.150.157  132.252.150.149  SCTP     ENC_DATA AUTH
6380 3.599143  132.252.150.157  132.252.150.149  SCTP     ENC_DATA AUTH

⊞ Frame 6376 (1106 bytes on wire, 1106 bytes captured)
⊞ Ethernet II, Src: 00:10:a4:b0:ce:7f, Dst: 00:09:5b:62:04:f0
⊞ Internet Protocol, Src Addr: 132.252.150.157 (132.252.150.157), Dst Addr: 132.252.150.149 (132.252.150.149)
⊟ Stream Control Transmission Protocol
    Source port: 6001
    Destination port: 2000
    Verification tag: 0x084aaabf
    Checksum: 0x265941bb (incorrect Adler32, should be 0xe98308df)
  ⊟ ENC_DATA chunk, payload length: 1016 bytes)
    ⊞ Chunk type: ENC_DATA (16)
      Chunk flags: 0x00
      Chunk length: 1032
      Master key reference: 1
      Random number length: 8
      Initialization vector: ËM·@ÂN\001
  Data (1016 bytes)
⊟ Stream Control Transmission Protocol
  ⊟ AUTH chunk, payload length: 20 bytes)
    ⊞ Chunk type: AUTH (17)
      Chunk flags: 0x00
      Chunk length: 28
      Master key reference: 1
      Reserved: 0
  Data (20 bytes)

0000  00 09 5b 62 04 f0 00 10  a4 b0 ce 7f 08 00 45 10   ..[b.ð.. ¤°Î...E.
0010  04 44 00 00 40 00 40 84  fe fa 84 fc 96 9d 84 fc   .D..@.@. þú.ü...ü
0020  96 95 17 71 07 d0 08 4a  aa bf 26 59 41 bb 10 00   ...q.Ð.J ª¿&YA»..
0030  04 08 00 01 00 08 cb 4d  b7 40 c2 4e 01 00 9f 62   ......ËM ·@ÂN...b
0040  71 c8 c0 ab d2 aa ec 29  0b d2 6b 5d 94 63 0d 35   qÈÀ«Òªì) .Òk].c.5
```

Figure 6.4: Network capture of S-SCTP

6.3.1 Setup of SCTP over IPsec

I used the Freeswan IPsec implementation in my experiments. The Freeswan implementation uses several configuration files such as ipsec.conf and ipsec.secret. The ipsec.conf file includes the main configuration of the IPsec connection and the ipsec.secret configuration file includes a public or shared secret key. These configuration files must be configured before beginning transport of secure data. In my experiment I used the following configuration.

```
conn sample             This row indicates a connection name.
type=transport          This row indicates IP connection type.
left=132.252.150.157    This row indicates IP address of left
                        entity.
right=132.252.150.149   This row indicates IP address of right
                        entity.
ike=aes128-md5          This row defines a crypto algorithm
                        suite for key exchange.
esp=aes128-sha1         This row defines a crypto algorithm
                        for encryption.
compress=no             This row enables or disables
                        a compression.
auth=ah                 This row enables or disables
                        authentication.
authby=secret           This row defines that key mechanism uses
                        shared secret or public key mechanism.
```

Both associated endpoints use the same configuration, only the left and right IP addresses have to change positions.

6.3.2 Setup of TLS over SCTP

I adapted some functions from our sctplib implementation to OpenSSL and created additional OpenSSL functions. All additional functions are related to SCTP streams. The following new functions were added in OpenSSL and they are used only for SCTP:

SSL_set_sid(SSL *s, int fd, int stream_num): Connect the SSL object(s) to a file descriptor (fd) and stream (stream_num), which is used to transport TLS messages.

SSL_get_sid(SSL *s, int fd, int stream_num): Get file descriptor (fd) and stream number (stream_num) linked to an SSL object.

BIO_set_sid(b,stream_num,c): Sets the stream number (stream_num) belonging to the file descriptor (fd) of BIO (b). Before using the BIO_set_sid() function, the same BIO_set_fd() function as in TLS over TCP has to be called.

BIO_get_sid(b,stream_num): Gets the stream number (stream_num) belonging to the file descriptor of BIO (b).

The functions SSL_set_sid() and SSL_get_sid() are integrated in the SSL connection part and the functions BIO_set_sid() and BIO_get_sid() are integrated in to the BIO I/O abstraction part (see figure 5.15).

The setup fashion of TLS over SCTP is similar to TLS over TCP. TLS over SCTP uses the new functions defined above instead of TCP functions. The correspondence of TCP functions and new functions for SCTP is shown in the table 6.1.

TLS over SCTP functions	TLS over TCP functions
SSL_set_sid()	SSL_set_fd()
SSL_get_sid()	SSL_get_fd()
BIO_set_fd()	BIO_set_fd()
BIO_get_fd()	BIO_get_fd()
BIO_set_sid()	–
BIO_get_sid()	–

Table 6.1: Comparison of OpenSSL functions for TCP and SCTP

The setup of TLS over SCTP is dynamically configured by the user. The following example shows the order of function calls for a simple configuration of TLS over SCTP.

```
. . . . . .
. . . . . .
ext_socket();   Create a socket.
ext_bind();     Assign a local address to a socket.
ext_accept();   Accepts a newly establish association.
SSL_new();      Create a new SSL structure.
SSL_set_sid();  Connects a socket descriptor to SSL.
SSL_accept();   Establish a SSL connection.
. . . . . .     Ready to transport a secure data.
```

6.4 Experimental measurements and results

In this section, I will describe some experiments using the following three measurement scenarios to compare the data path performance of all three security solutions:

1. In a first scenario, I measured the end-to-end throughput varying the user message size that needed to be transmitted securely. The bundling option of SCTP was disabled.

2. The second scenario is similar to the first scenario. The only difference is that this scenario uses the bundling mechanism of SCTP.

3. In a third scenario, I measured the end-to-end throughput varying the percentage of data that needed to be transmitted securely. Bundling was not used, but measurements were performed with and without fragmentation.

In a fourth measurement scenario, I compared the main memory used by the different security solutions depending on the number of streams that needed to be secured.

All security protocols support different crypto algorithms and for my experiments I chose the AES, 3DES and SHA crypto algorithms which are supported by all security protocols. In the first three scenarios two different encryption algorithms, 3DES and AES, were used for every security solution. For authentication, I used the SHA algorithm. It doesn't affect the throughput how many applications or how many streams are used in an association. That's why I used only one stream in the first three scenarios. For example, in the secure mix scenario both secured and unsecured data used the same stream.

The following two factors have to be taken into account when interpreting the measurement results because they affect the measurement of throughput:

- For a completely fair comparison, identical implementations of the crypto algorithms have to be used in measurements. OpenSSL and S-SCTP use the same crypto implementation whereas IPsec uses a different crypto implementation for AES. However, all three implementations use the 3DES and SHA implementations from OpenSSL.

- IPsec is implemented in the operating system or kernel whereas OpenSSL and S-SCTP are implemented in user space. Therefore, IPsec has a performance advantage over the other two security solutions. Also, OpenSSL is a well optimised implementation whereas S-SCTP is a prototype implementation which is not fully optimised for performance.

6.4.1 Sensitivity of throughput to message size

In this scenario, messages with different length were transported between two connected computers and the throughput was measured. All transported data was fully encrypted and authenticated. The results of these measurements are shown in figure 6.5 for the 3DES-SHA cipher and in figure 6.6 for the AES-SHA cipher. The throughput of standard SCTP is also shown. The maximum throughput of standard SCTP is about 2000 kb/sec and the throughputs of TLS over SCTP are 1364 kb/sec for the 3DES-SHA cipher and 1672 kb/sec for the AES-SHA cipher. The differences between SCTP's throughput and e.g. TLS over SCTP's throughput (2000-1364=636 kb/sec for 3DES-SHA and 2000-1672=228 kb/sec for AES-SHA) indicate the cost of the crypto algorithms.

Figure 6.5 shows that the throughput increases with the message size. It also shows that all graphs exhibit a drop in throughput at an IP packet size of 1500 bytes. This is due to the fact that Ethernet has a 1500 byte MTU and fragmentation occurs when the packet size reaches 1500 bytes. When fragmentation has to be done, additional overhead, e.g., for the headers of the second packet has to be transferred, the fragmentation algorithm has to be performed and – depending on the security solution – additional encryption and authentication operations have to be performed. This in combination leads to the drop in throughput shown in figures 6.5 and 6.6.

As an example, figure 6.7 shows the drop in throughput for SCTP over IPsec and TLS over SCTP (3DES-SHA cipher) in more detail. In this figure, point A (1231 kb/sec for SCTP over IPsec and 1230 kb/sec for TLS over SCTP) indicates the throughput without fragmentation and point B (1123 kb/sec for SCTP over IPsec and 1173 kb/sec for TLS over SCTP) indicates the throughput with fragmentation. The differences 1231-1123=108 kb/sec and 1230-1173=57 kb/sec reflect the cost of security for the second packet and for the fragmentation mechanism. We know TLS performs security per message and IPsec performs the security per packet (overhead + message). Therefore, the difference of the two drop values (108-57=51 kb/sec) reflects the cost of security for the second packet. The throughput drop will occur every time when a message needs a new fragment, e.g. when the IP packet size reaches 3000 bytes, 4500 bytes etc. It should be mentioned that the curves for large message sizes are interpolated between the few measurement points explicitly shown in the graphs and, therefore, this effect is not visible.

In figure 6.5, left of the fragmentation point the throughput for all security solutions is practically identical, because in this scenario bundling is not allowed. After fragmentation, however, the throughput for TLS over SCTP is clearly higher than for the other two security solutions. This can be explained by the fact that in TLS over SCTP security operations (encryption and authentication) are performed per message (see section 5.2.3), whereas in S-SCTP and SCTP over IPsec they are performed per message fragment. This

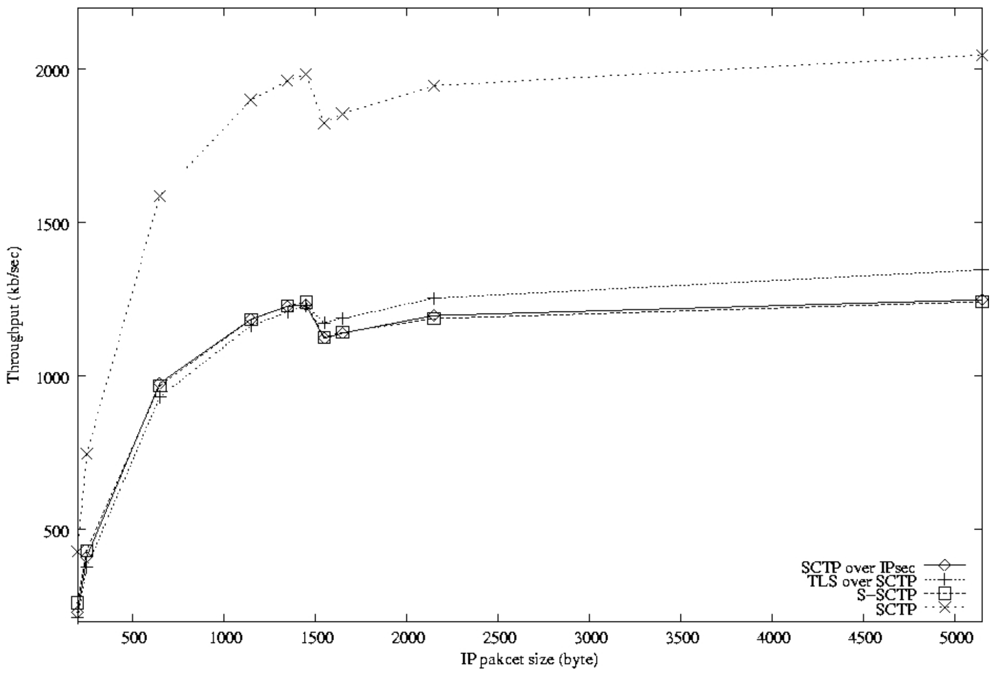


Figure 6.5: Throughput of secured data (3DES-SHA cipher)

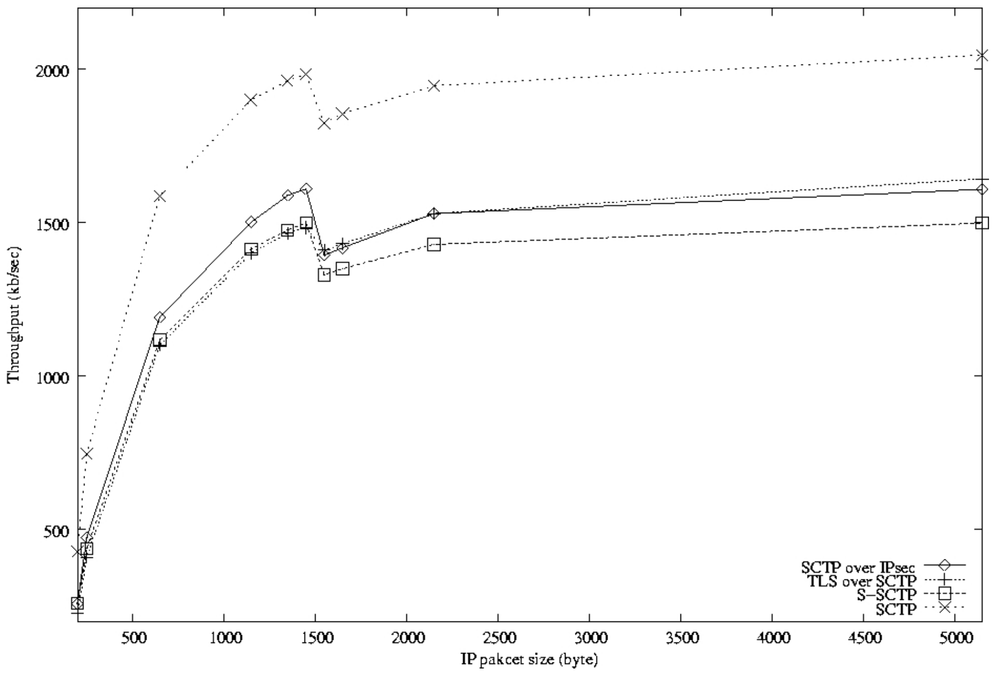


Figure 6.6: Throughput of secured data (AES-SHA cipher)

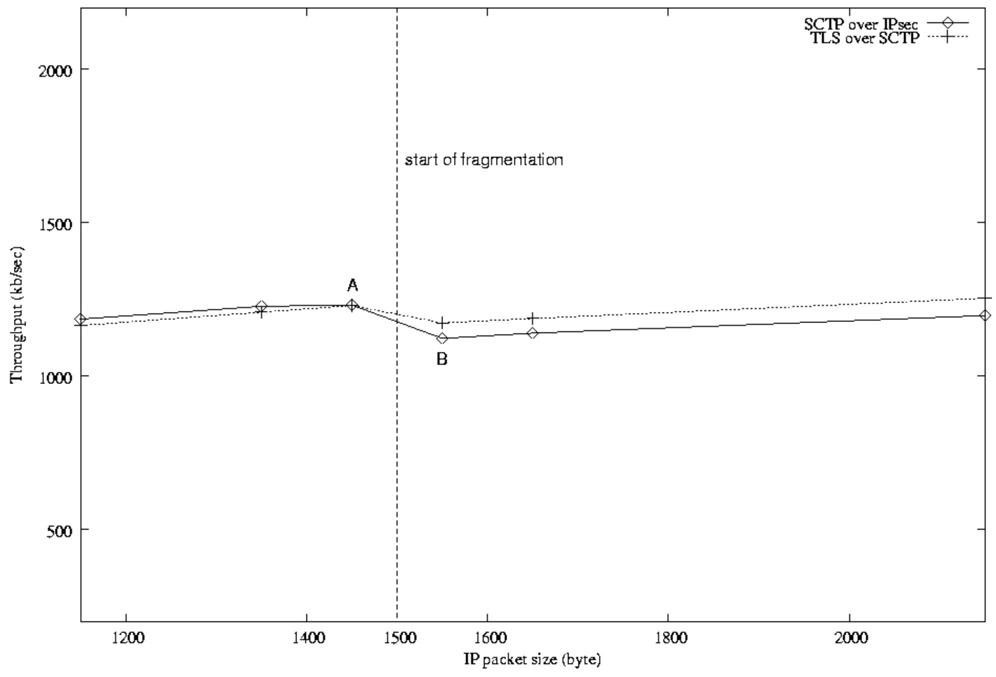


Figure 6.7: Throughput of secured data without and with fragmentation (3DES-SHA cipher)

means, for example, that when the user sends a 2500 byte message, security operations are performed once in TLS and twice in S-SCTP and IPsec.

Figure 6.6 for the AES-SHA cipher shows a fairly similar result. The difference is that the throughput of all security solutions is higher. This is due to the fact that AES computation is much faster than 3DES computation. Because IPsec uses a different, obviously more efficient, AES implementation than TLS and S-SCTP, the performance of IPsec reaches that of TLS even with the additional authentication effort while S-SCTP has lower throughput.

6.4.2 Sensitivity of throughput to message size (bundling)

In this scenario, small messages with different length were bundled and transported between two connected computers and the throughput was measured. All transported data was fully encrypted and authenticated. In this scenario I exchanged the two computers and used the faster one as sender to achieve the maximum bundling gain. SCTP waits a certain time (time depends on the implementation) before bundling messages. Messages are only bundled if they can be delivered from the user during this time interval. Delivery time depends on the processing time of socket API and some sctplib procedures. With the faster computer, the messages can be delivered fast enough to bundle messages up to the MTU size.

However, because decryption and authentication verification require more computation time than encryption and authentication, exchanging the computers also has an impact on the achieved throughput. Therefore, the figures are not directly comparable to those in section 6.4.1.

The result of these measurements are shown in figure 6.8 for the 3DES-SHA cipher and in figure 6.9 for the AES-SHA cipher.

Again, TLS performs authentication per message and both S-SCTP and IPsec perform authentication per packet. However, in these measurements a packet includes more than one message. This means this scenario is not favorable for TLS. For example, if the message size is 200 bytes then a SCTP packet includes approximately seven messages, which means TLS performs authentication seven times.

SCTP over IPsec and S-SCTP have similar throughput for 3DES-SHA, but for AES-SHA IPsec performs better than S-SCTP due to the more efficient AES implementation.

In figure 6.9 it is shown that if the number of bundled messages decreases then the throughput difference between SCTP over IPsec and S-SCTP increases. In this bundling scenario, the throughput depends on the cost of security and of the bundling mechanism. If a SCTP packet includes many chunks then the time consumed for bundling in the sctplib is the dominating factor. With increasing message size (less bundling), the advantage of the superior AES implementation of IPsec becomes obvious. Also, the performance penalty of TLS over SCTP diminishes.

6.4.3 Sensitivity of throughput to percentage of secure traffic

In this scenario the percentage of data that needed to be transmitted securely between two connected computers was varied. In this experiment the message size was constant. I measured the throughput with two different message sizes, i.e. 1000 bytes, which doesn't require fragmentation and 10000 bytes, which requires fragmentation.

Figure 6.10 shows the results for the 3DES-SHA cipher if sending 1000 bytes of data (not fragmented) in a message, figure 6.11 shows the results for the AES-SHA cipher for the same scenario.

Because IPsec secures all data transported over a SA (security association), the varying percentage of traffic requiring encryption is irrelevant for IPsec and the throughput of IPsec is constant. The advantage of the capability to mix secure and unsecure data in TLS over SCTP and S-SCTP is obvious. TLS over SCTP and S-SCTP achieve up to 56% higher

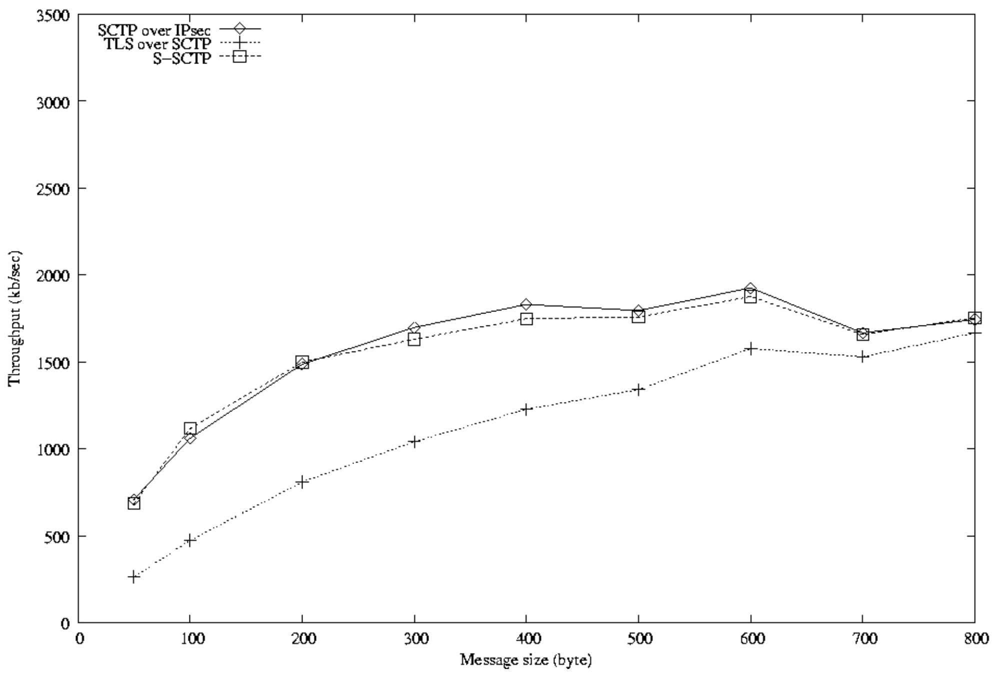


Figure 6.8: Throughput of secured data with bundling (3DES-SHA cipher)

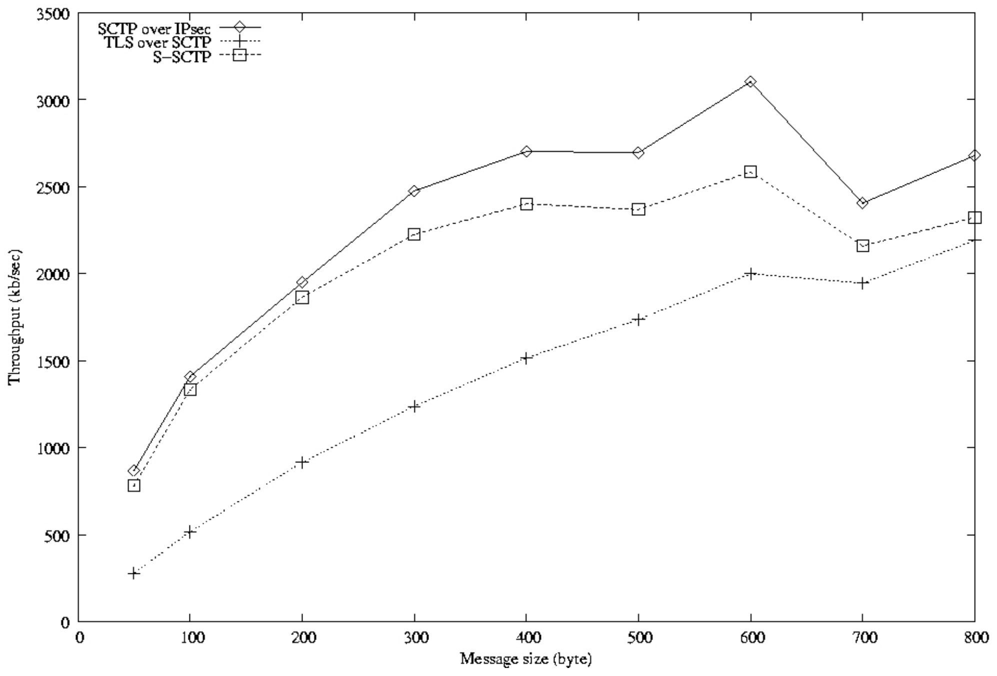


Figure 6.9: Throughput of secured data with bundling (AES-SHA cipher)

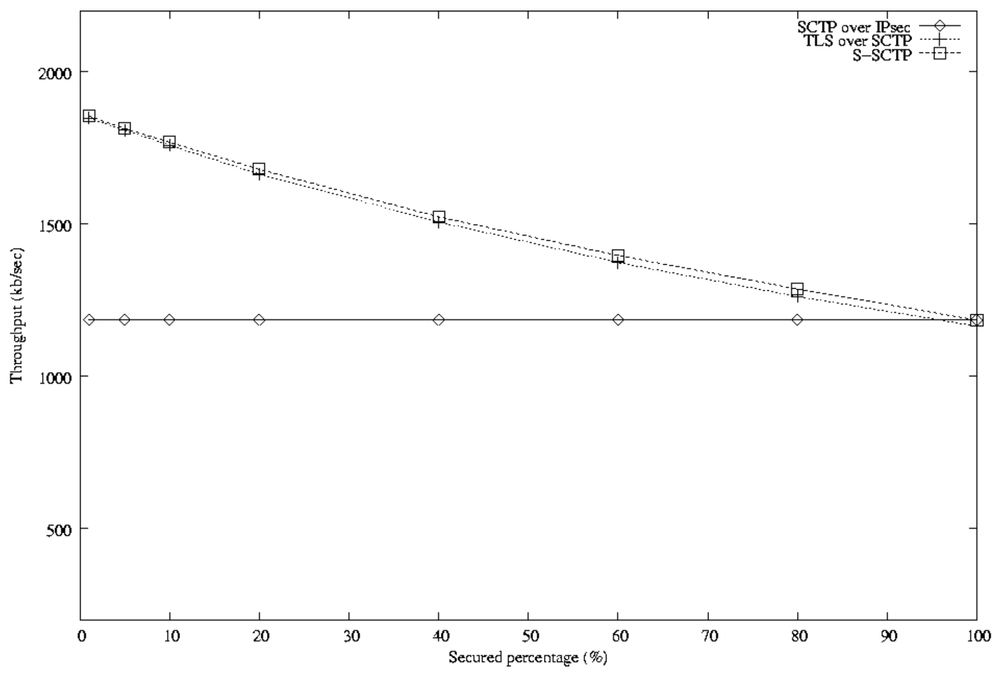


Figure 6.10: Secure mix without fragmentation (3DES-SHA cipher, message size 1000 bytes)

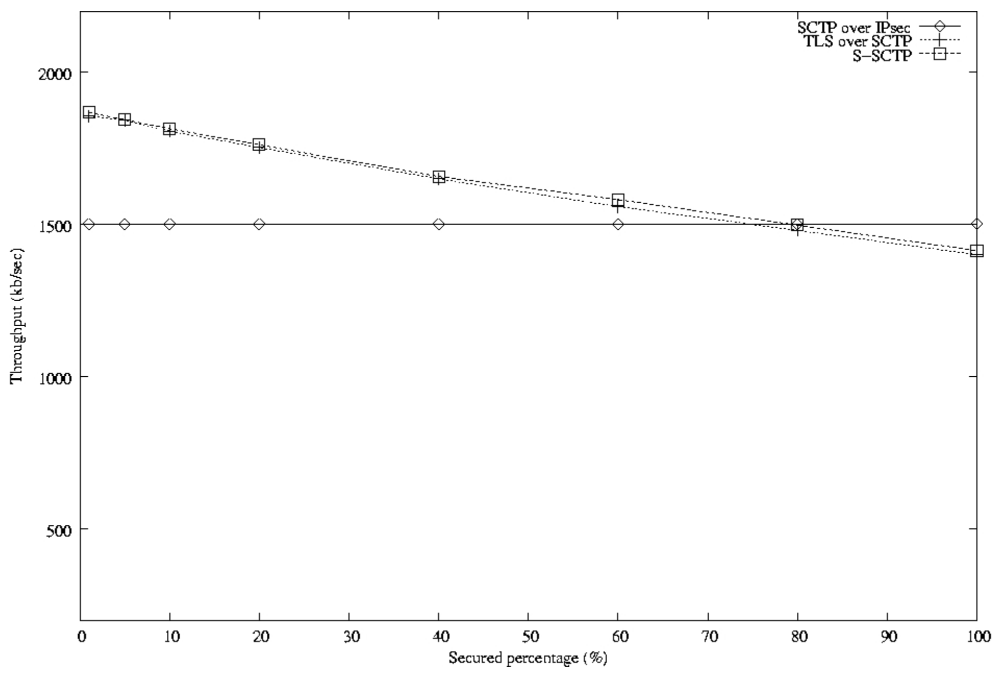


Figure 6.11: Secure mix without fragmentation (AES-SHA cipher, message size 1000 bytes)

throughput than IPsec if only a small part of the traffic is secured. The performance of TLS over SCTP and S-SCTP is practically identical.

Figure 6.11 shows that both S-SCTP and TLS over SCTP have lower throughput than IPsec if 80 percent or more of the traffic is secured. The reason for this result is again that IPsec uses a more efficient AES implementation.

Figures 6.12 and 6.13 show the same measurement if 10000 byte messages are sent (fragmented). Both graphs show that TLS over SCTP has a higher throughput than the other two security solutions as expected. TLS performs authentication per message whereas S-SCTP and IPsec perform authentication per fragmented message and a 10000 byte message will be fragmented into seven packets. Therefore, TLS performs only one authentication and S-SCTP and IPsec perform seven.

6.4.4 Relationship between memory and number of streams

In this scenario, I tried to find out the characteristics of the security solutions with respect to computer hardware resource usage (e.g memory and CPU). Figure 6.14 shows the memory consumption of the security solutions depending on the number of SCTP message streams that need to be secured.

Since TLS over SCTP establishes a separate TLS connection for each secured message stream, the memory usage for this solution increases approximately linear with the number of secured streams. Furthermore, my measurements showed that this trend continues for even higher numbers of streams until the physical memory is completely filled. The other two solutions show a nearly constant use of main memory (approx. 1.5 MByte for SCTP over IPSec, and 1.8 Mbyte for Secure SCTP). This difference is due to the fact that for IPSec the standard SCTP implementation could be used, and security mechanisms are implemented in the operating system kernel. Secure SCTP, on the other hand, is implemented as a user process and thus shows a slightly larger memory footprint. The memory usage of standard SCTP is similar as for SCTP over IPsec (approx. 1.5 MByte).

6.4.5 Summary of the measurement results

The S-SCTP data path implementation is fully functional and first measurements have been performed in several scenarios for all three security solutions. The results of these experiments can be summarised as follows:

1. S-SCTP and SCTP over IPsec perform cryptographic operations on packet level rather than on message level as TLS over SCTP does. Therefore, their performance is superior for small messages where several messages are bundled into one SCTP

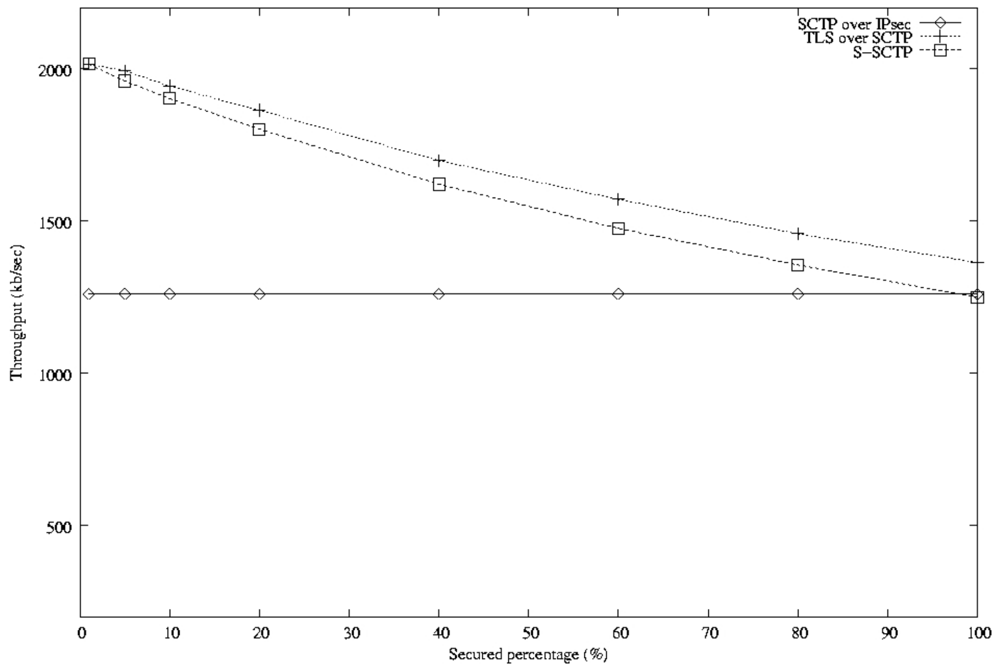


Figure 6.12: Secure mix with fragmentation (3DES-SHA cipher, message size 10000 bytes)

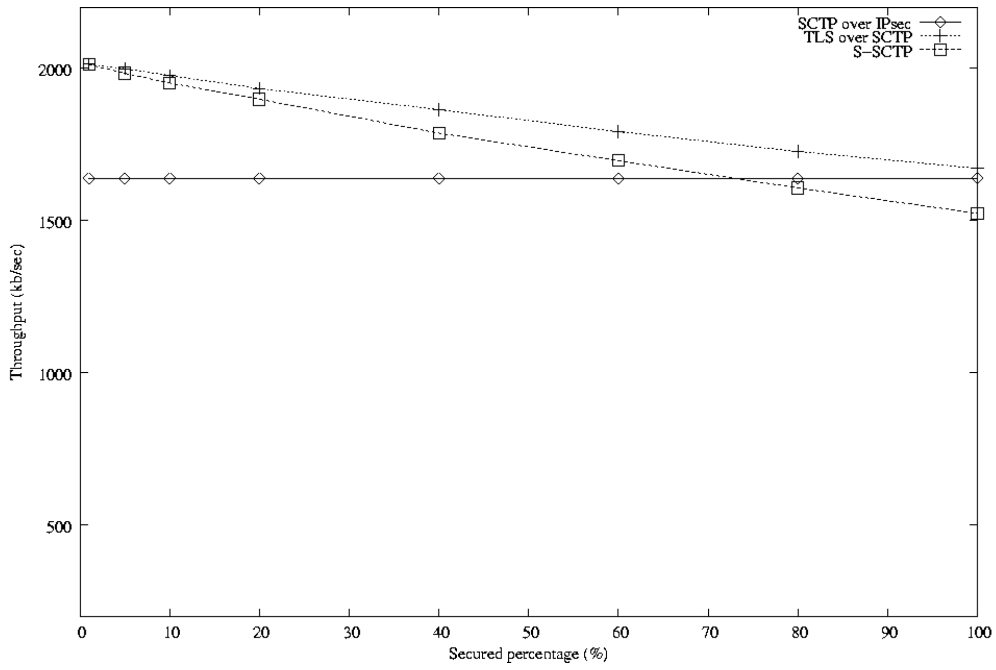


Figure 6.13: Secure mix with fragmentation (AES-SHA cipher, message size 10000 bytes)

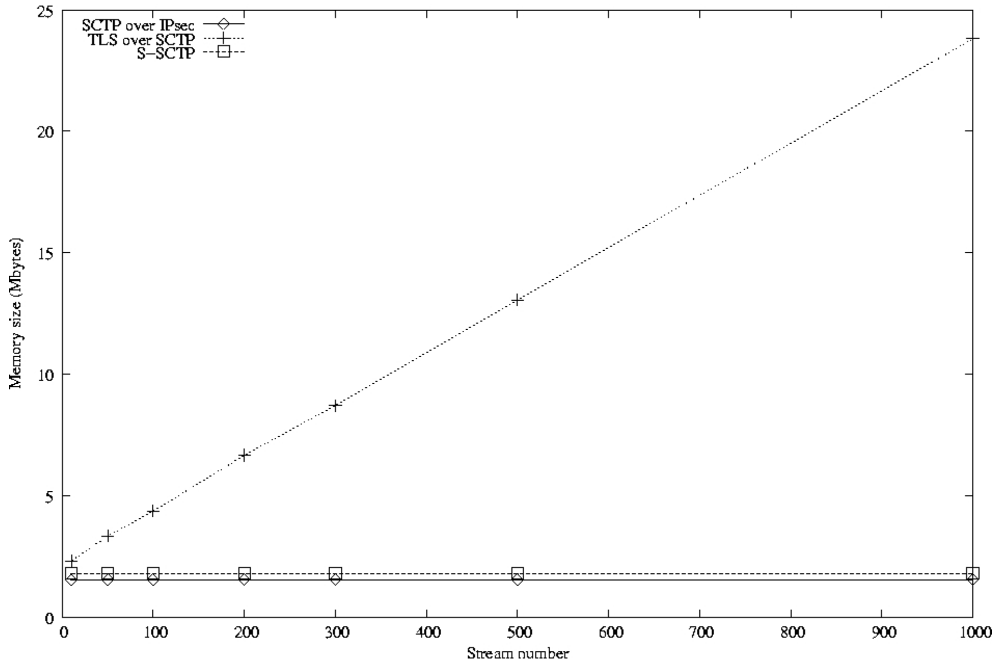


Figure 6.14: Memory usage

packet. On the other hand, TLS over SCTP performs better for large messages which have to be fragmented. For messages with size between 50% and 100% of the current MTU, all security solutions show only slight differences in performance.

2. One factor in data path performance is overhead. TLS provides one HMAC per message, S-SCTP provides one HMAC per SCTP packet and IPSec provides one HMAC per IP packet. One HMAC per message gives TLS a significant advantage when a message has to be fragmented into several packets. For the same reason, S-SCTP and IPSec have a significant advantage when bundling is done, e.g. a packet includes several messages. Since signalling messages are typically small, this is the expected operating point for the main application scenario.

3. S-SCTP and TLS over SCTP can mix secured and unsecured messages. This results in a major performance benefit compared to SCTP over IPsec if the percentage of secured traffic is low to moderate. Here TLS and S-SCTP achieve up to 56% performance advantage compared to IPsec for a small percentage of secured traffic. The measurements have been done for message sizes where bundling is not applied. For small messages allowing bundling, S-SCTP would even perform significantly better

than TLS. The mix scenario with a low percentage of secured traffic is a typical operating point for mixes, e.g. assuming a high volume multimedia stream with a low volume control stream.

4. TLS over SCTP requires a lot of memory for a large number of streams whereas for the other two security solutions, the number of streams (applications) doesn't affect the usage of memory. The linearly increasing memory consumption of TLS can lead to performance penalties for a large number of streams, especially for embedded devices with small main memory.

The measurements show, that the efficiency of the cryptographic algorithms and their implementations has a significant influence on data path performance. As the measurements for the AES algorithm show, there is still some potential to optimise S-SCTP performance by integrating better crypto implementations. In addition, also some areas of the S-SCTP prototype implementation have been identified which could be optimised for performance during a redesign process.

From a performance point of view, the decision whether to use TLS or IPSec depends on characteristics defined by the applications at runtime (message size, percentage of secured traffic, number of SCTP streams) which cannot be predicted beforehand. However, S-SCTP performs always similar to the better solution in all measured scenarios. Even for scenarios with large messages requiring fragmentation, the performance advantage of TLS compared to S-SCTP is limited. Therefore, S-SCTP is more universally applicable over a wide range of protocol parameters. In addition to the performance benefits, S-SCTP avoids the major functional and management drawbacks (see table 4.1) of the other two solutions which makes S-SCTP clearly superior to the other solutions.

Chapter 7

Conclusion

This thesis deals with security solutions for the new multi-purpose transport protocol called SCTP. SCTP was defined by IETF and due to its various features and easy extendability will be a valid option in many application scenarios – not only in those already standardised. There are a few already existing SCTP applications which are all requiring a security mechanism to protect sensitive data. Up to now, there exist two already standardised security solutions for SCTP, namely TLS over SCTP and SCTP over IPsec. The standardised security solutions are using the standard protocols TLS and IPsec without or with only minimum modification, which was a major requirement when defining these solutions.

I evaluated both standardised security solutions and my evaluation showed drawbacks and limitations of both security solutions with respect to functionality, performance and secure session management. Some of the drawbacks are simply due to the nature of these protocols. For example, TLS was defined with only TCP in mind and, therefore, requires ordered delivery. This means that TLS cannot support the SCTP unordered delivery feature. SCTP has several features which are not provided by TCP and UDP and most drawbacks and limitations of the standardised security solutions are related to these features. SCTP has two extensions which are called partially reliable delivery and dynamic address configuration. Both standard security protocols have problems with these extensions.

Based on this evaluation I proposed an integrated solution (S-SCTP) which supports all SCTP features and options in an efficient way. I examined standard SCTP and found three possible ways for integration of cryptographic functions. The first solution – which I have chosen – requires only one common secure session per association. The other two solutions, while avoiding the functional drawbacks of the standardised solutions, were similar to SCTP over TLS and SCTP over IPsec in that they would require several secure sessions per association. The key advantage of one secure session is that it decreases the secure session initialisation cost (in terms of delay and processing) and also decreases rekeying cost. Of course, a disadvantage is that if the secure key is compromised then all data will be unsafe. If in the other solutions many secure sessions exist and if the secure key of one secure session is compromised, then it doesn't affect the other secure sessions.

The new security extension I propose is fully compatible to standard SCTP and it avoids all functional drawbacks of the standardised security solutions. S-SCTP defines four security levels which provide flexible security functionalities such as mixing of secured and unsecured messages. Security level 2, in particular, does selective encryption on a per chunks basis which provides fine granular adaptation of S-SCTP to the security requirement of the application. For efficiency, S-SCTP provides authentication per SCTP packet and not per chunk or per message.

After defining the basic concept for S-SCTP, I made a formal specification of S-SCTP (see Appendix A) in SDL using the SDT tool to validate the protocol engine by visual inspection of the resulting state machine. I also simulated the main use cases by manually generating input events and checking the resulting state transitions for correctness.

I have fully specified the syntax extensions and the necessary procedures for the SCTP extension to the level of detail necessary for a contribution to IETF standardisation (the resulting document is included as Appendix B). S-SCTP defines thirteen new chunks and additional parameters used in these chunks. It also defines new error types.

In addition to the conceptual work, I created a prototype implementation of S-SCTP based on the our existing SCTP implementation and the crypto library of the OpenSSL open source implementation. The prototype implementation of S-SCTP was used to prove that the potential advantages of the integrated S-SCTP solution can actually be exploited. The simple S-SCTP socket API also defined in my work allows for easy control of all S-SCTP features and parameters directly from an application using it. This is a major advantage compared to the SCTP over IPsec solution in the area of security session management.

These aspects are summarised in table 7.1 and indicate clearly that the S-SCTP concept has reached its major design goals. It provides significant improvements over the standardised solutions in a sense that all functional and management drawbacks have been avoided without introducing performance penalties or significant implementation complexity.

Possible extensions to my work include the complete implementation of the already fully specified control path and performance optimisations of the prototype implementation. With the status reached, it would also be possible to propose S-SCTP to IETF standardisation.

Criteria	TLS over SCTP	SCTP over IPsec	S-SCTP
Scalability for multiple streams	-	+	+
Support for SCTP multi-homing	+	(-)	+
Overhead for small messages (bundling)	-	+	+
Overhead for long messages (fragmentation)	+	-	-
Protection for unordered delivery service	no	+	+
Protection for SCTP control chunks	no	+	+
Flexible multiplexing of secure/ insecure streams	+	no	+
Management of security sessions (handling, automation)	+	-	+
Partially Reliable Transport (SCTP extension)	no	+	+
Dynamic Address Reconfiguration (SCTP extension)	+	-	+

Keywords: ”+” - Advantage, ”-” - Disadvantage, ”no” - Not possible

Table 7.1: Comparison of all SCTP security solutions

Appendix A

SDL diagrams of S-SCTP

The following key terms used in SDL diagram.

Messages from S-SCTP to ULP (upper layer):

- Ind_Sec_Est: Indicates when a secure Session established.
- Ind_Err: Indicates when an error occurs.
- Ind_Rec_Data: Indicates when a data received.

Messages from ULP to S-SCTP:

- Req_SS_Begin: Request to establish a secure session.
- Req_SS_End: Request to terminate a secure session.
- Req_SS_Update: Request to update a secure session.
- Send_Data: Request to send a unsecured data.
- Send_SS_Data: Request to send a secured data.

Messages between two S-SCTP endpoints:

- SSOpReq: Secure Session Open Request.
- SSCert: Secure Session Certificate.
- SSOpReq_Ack: Secure Session Open Request Acknowledged.

- SSSerKey: Secure Session Server Key.
- SSCliKey: Secure Session Client Key.
- SSOpCom: Secure Session Open Complete.
- SS_Err: Secure Session error.
- SSClose: Secure Session Close Request.
- SSClose_Ack: Secure Session Close Request Acknowledged.
- Packet: A packet includes a unsecured data.
- SS_Packet: A packet includes secured data.

Messages from an standard SCTP association:

- Req_Shut: Request to terminate an association.
- Shut_Ack: An association shutdown acknowledged.
- Abort: Error occurs in an association.

Explanation of the variables which are used in the SDL diagram. In SDL diagram, I defined the several variables, the three constant values, named RET, ITER, REG, and the timer. I also defined some boxes which are indicate operations. The operations of box writen directly on the boxes. Inside the bosex are not defined but they are not play main roles of my SDL diagram. For example in the third diagram exist the box named a discard message. This box just discards messages.
The following paragraphs will describe every variables used in SDL diagram.

T, ct, RET: This is a timer to retransmission of secure session (SS) messages. The timer unit is must be same as T3 timer which is defined in RFC2960. I defined the ”ct” variable to store the timer value. Every time when a secure session changes the current state the variable ”ct” initiated to zero. The varibale ”ct” also initiated to zero when the value of ”ct” reaches to ”RET”. The ”RET” is the retransmission value which is depends on an implementation.

ITER, r: The ”ITER” is a value that defines a retransmission number of SS messages. The variable ”r” defines the current retransmitted number. IF the value of ”r” reaches to ITER the current SS will be terminated immediately.

m1, m2, m3, x: These three temporary variables indicate SS messages, belong to same SS handshake way, are received or not. In different SS way they indicates different SS messages. The variables initiated to zero when the current state is changed. Some of the SS messages have a parameter. The variable ”x” indicates this parameter.

cert: Two negotiated endpoints may be have a certificate and want to exchange that. This case the variable ”cert” indicates that an endpoint has a certificate.

RAND: When two endpoints want to make a secure session at the same time. This case will occur a collision. This case both endpoints wait a random time before resend a secure session request message. The value ”RAND” indicates this random time.

REG, reg, con: The secure session negatiation could be unsuccessful from any problem (e.g. timer expired, certificate invalid...). This situation an endpoint try to negatiate again. The value ”RET” defines the maximum attempts to a secure session negotiation and the variable ”reg” indicates the current attemtp number. The variable ”con” indicates that hte value of ”reg” is not yet reached to ”RET” and a secure session negotiation can be to continue.

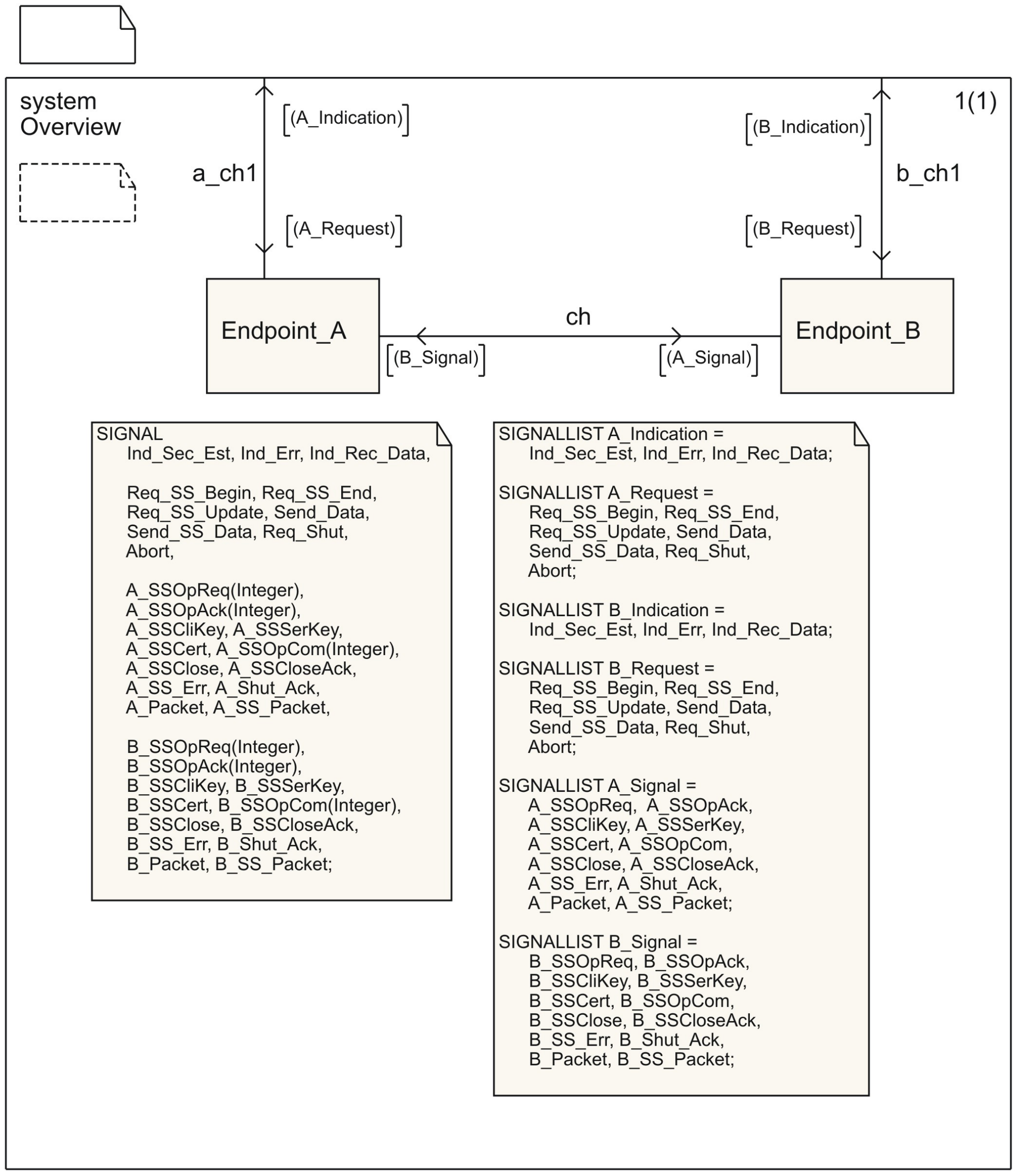
ser: After a secure session negotiation one endpoint must be a server and other endpoint must be a client. The variable ”ser” indicates that an endpoint is a server (1) or client (0).

ss: The variable ”ss” used in re-keyeing mechanism and it indicates that the endpoint in updating state.

cl: The variable ”cl” used in termination of the current secure session. There are two kind of termination. A user want to close the current secure session and want to use a normal SCTP association or a user want to close the both SCTP association and secure session. This situation the variable ”cl” indicates that a secure session closing or a SCTP association closing.

h: The varibale ”h” indicate hash value of the handshake or re-keying mechanisms.

This appendix contains a complete technical specification of S-SCTP in a format that is suitable to submit it as a draft to the IETF.



```
SIGNAL
    Ind_Sec_Est, Ind_Err, Ind_Rec_Data,

    Req_SS_Begin, Req_SS_End,
    Req_SS_Update, Send_Data,
    Send_SS_Data, Req_Shut,
    Abort,

    A_SSOpReq(Integer),
    A_SSOpAck(Integer),
    A_SSCliKey, A_SSSerKey,
    A_SSCert, A_SSOpCom(Integer),
    A_SSClose, A_SSCloseAck,
    A_SS_Err, A_Shut_Ack,
    A_Packet, A_SS_Packet,

    B_SSOpReq(Integer),
    B_SSOpAck(Integer),
    B_SSCliKey, B_SSSerKey,
    B_SSCert, B_SSOpCom(Integer),
    B_SSClose, B_SSCloseAck,
    B_SS_Err, B_Shut_Ack,
    B_Packet, B_SS_Packet;
```

```
SIGNALLIST A_Indication =
    Ind_Sec_Est, Ind_Err, Ind_Rec_Data;

SIGNALLIST A_Request =
    Req_SS_Begin, Req_SS_End,
    Req_SS_Update, Send_Data,
    Send_SS_Data, Req_Shut,
    Abort;

SIGNALLIST B_Indication =
    Ind_Sec_Est, Ind_Err, Ind_Rec_Data;

SIGNALLIST B_Request =
    Req_SS_Begin, Req_SS_End,
    Req_SS_Update, Send_Data,
    Send_SS_Data, Req_Shut,
    Abort;

SIGNALLIST A_Signal =
    A_SSOpReq,  A_SSOpAck,
    A_SSCliKey, A_SSSerKey,
    A_SSCert, A_SSOpCom,
    A_SSClose, A_SSCloseAck,
    A_SS_Err, A_Shut_Ack,
    A_Packet, A_SS_Packet;

SIGNALLIST B_Signal =
    B_SSOpReq, B_SSOpAck,
    B_SSCliKey, B_SSSerKey,
    B_SSCert, B_SSOpCom,
    B_SSClose, B_SSCloseAck,
    B_SS_Err, B_Shut_Ack,
    B_Packet, B_SS_Packet;
```

Figure A.1: S-SCTP SDL overview

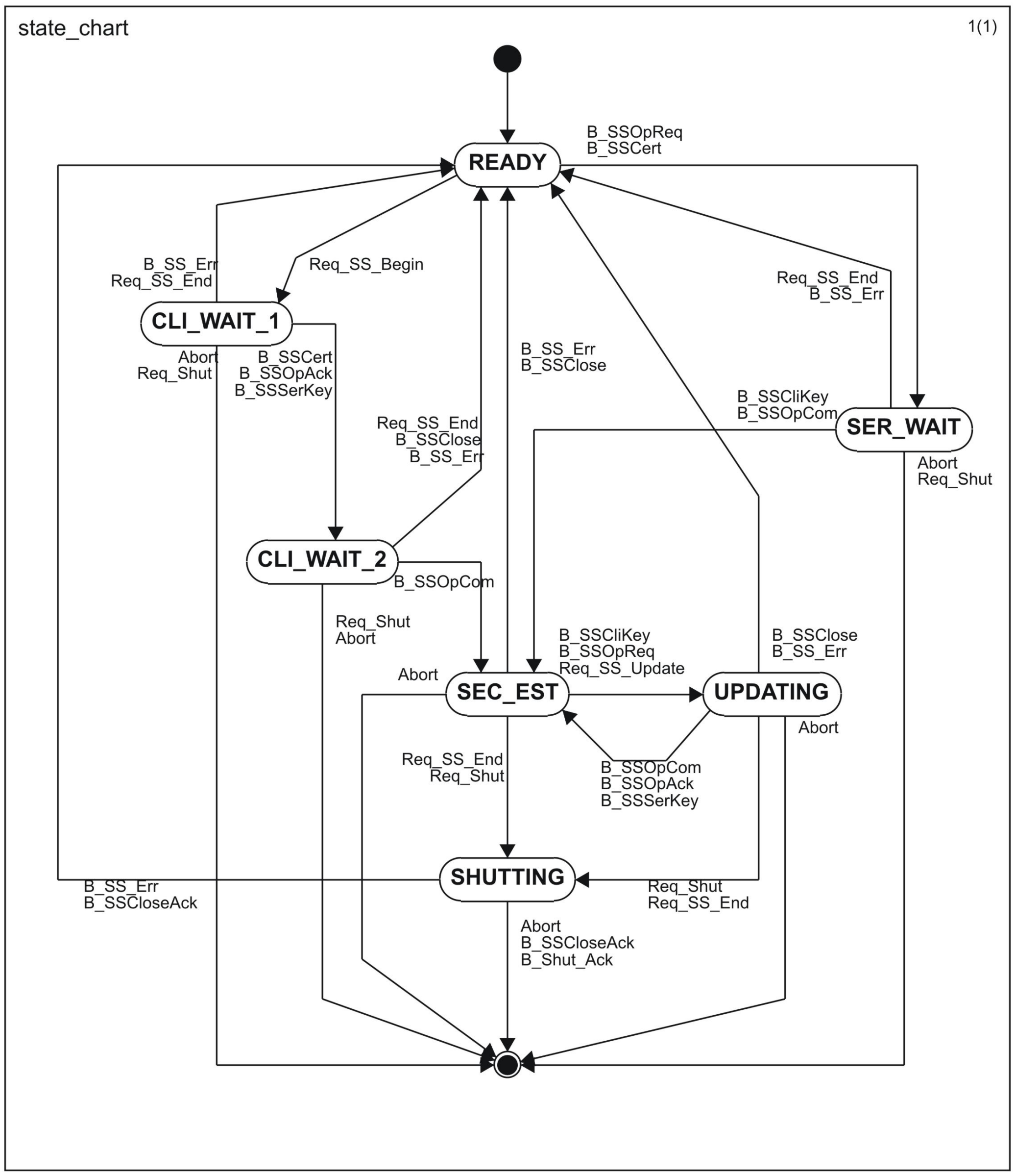
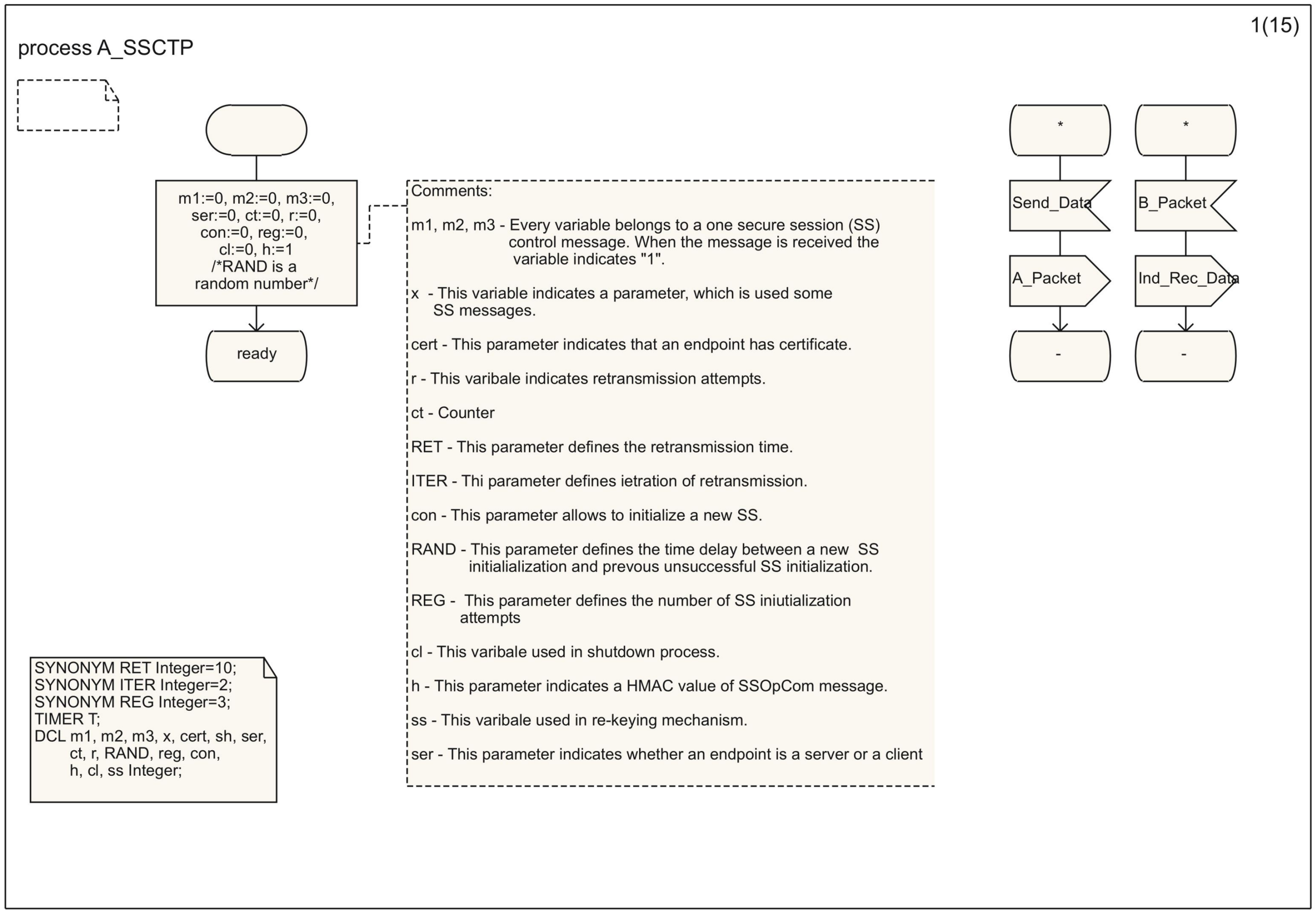


Figure A.2: S-SCTP state machine



```
m1:=0, m2:=0, m3:=0,
ser:=0, ct:=0, r:=0,
con:=0, reg:=0,
cl:=0, h:=1
/*RAND is a
random number*/
```

Comments:

m1, m2, m3 - Every variable belongs to a one secure session (SS) control message. When the message is received the variable indicates "1".

x - This variable indicates a parameter, which is used some SS messages.

cert - This parameter indicates that an endpoint has certificate.

r - This varibale indicates retransmission attempts.

ct - Counter

RET - This parameter defines the retransmission time.

ITER - Thi parameter defines ietration of retransmission.

con - This parameter allows to initialize a new SS.

RAND - This parameter defines the time delay between a new SS initialialization and prevous unsuccessful SS initialization.

REG - This parameter defines the number of SS iniutialization attempts

cl - This varibale used in shutdown process.

h - This parameter indicates a HMAC value of SSOpCom message.

ss - This varibale used in re-keying mechanism.

ser - This parameter indicates whether an endpoint is a server or a client

```
SYNONYM RET Integer=10;
SYNONYM ITER Integer=2;
SYNONYM REG Integer=3;
TIMER T;
DCL m1, m2, m3, x, cert, sh, ser,
      ct, r, RAND, reg, con,
      h, cl, ss Integer;
```

Figure A.3: S-SCTP SDL diagram 1

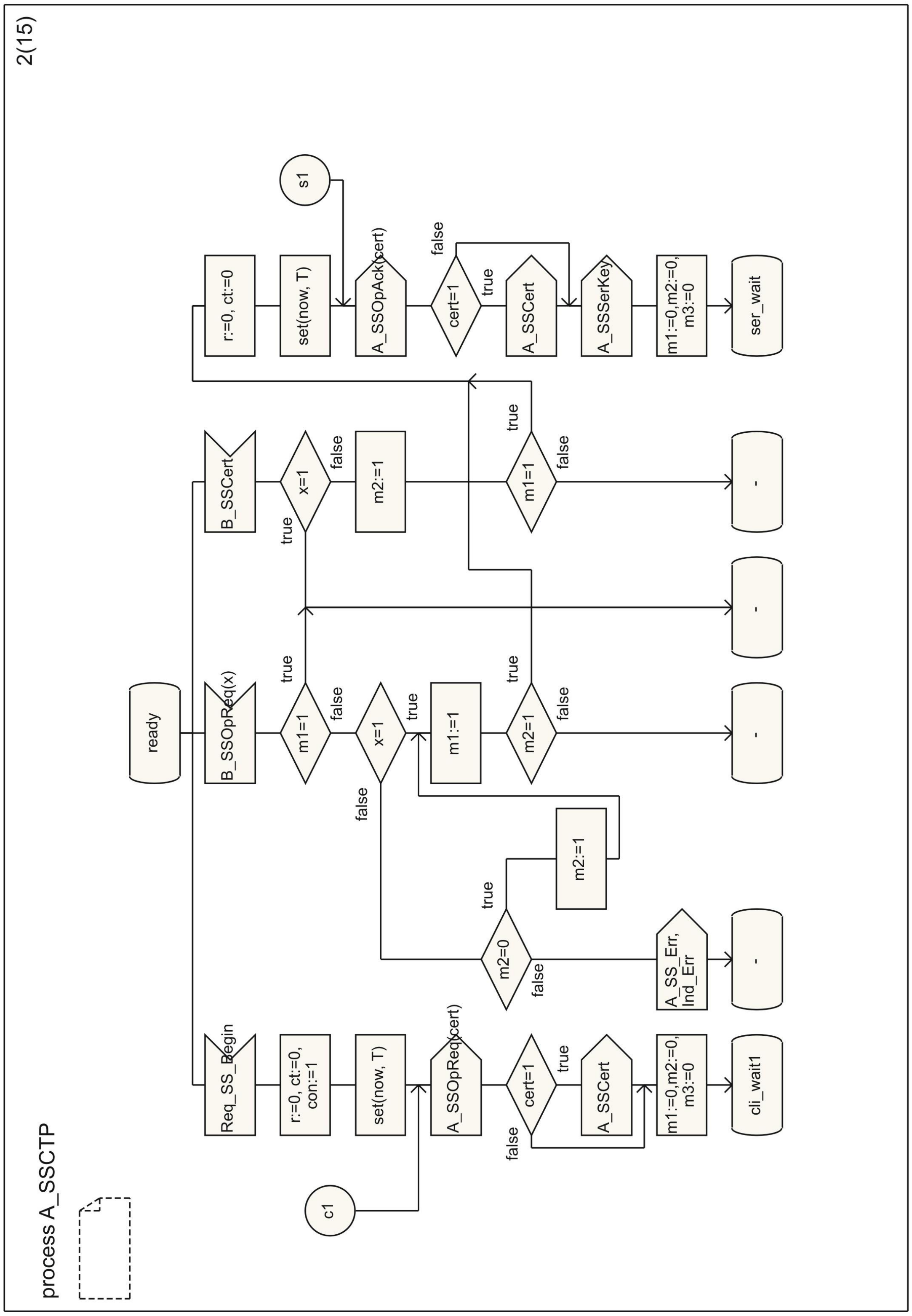


Figure A.4: S-SCTP SDL diagram 2

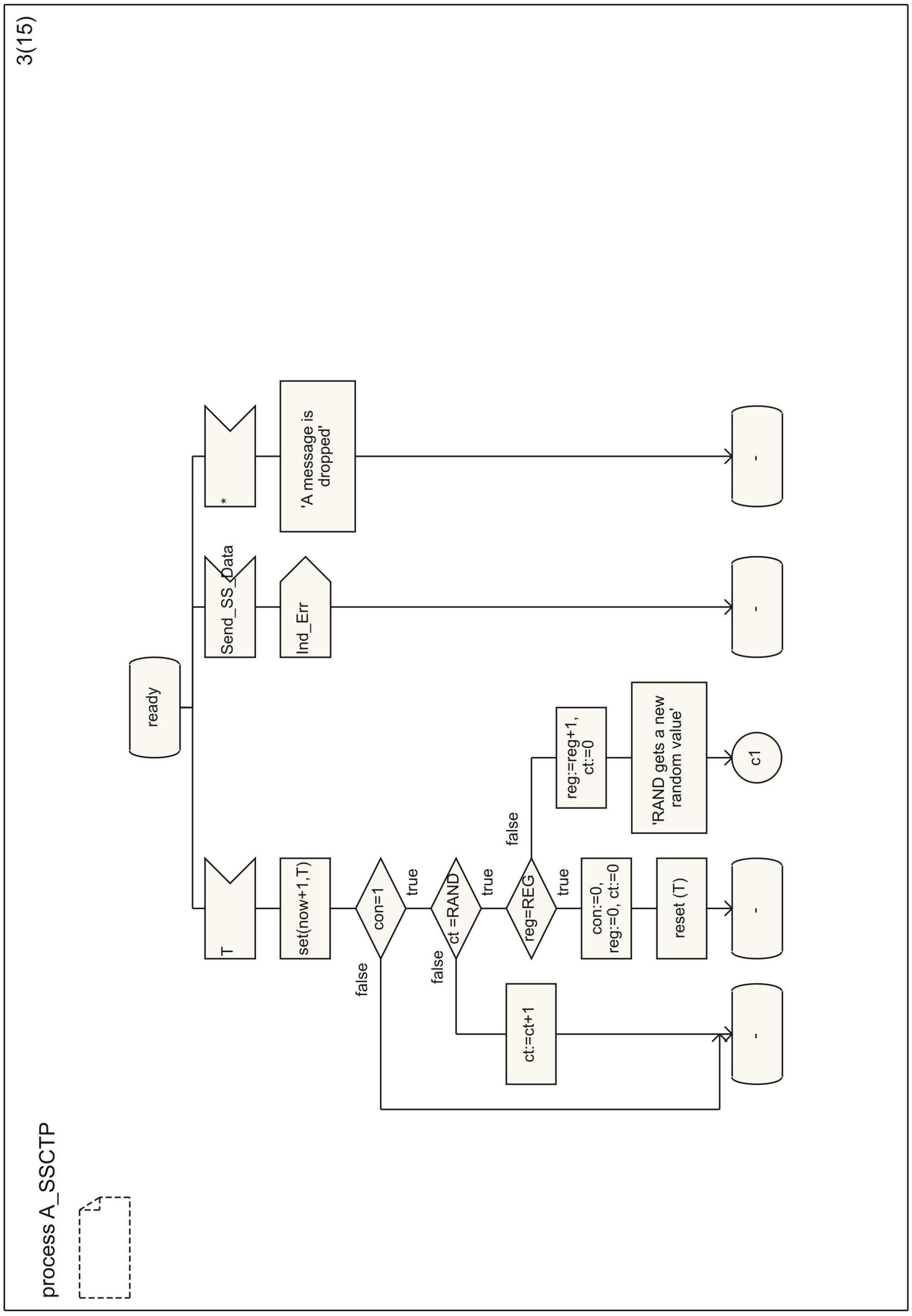


Figure A.5: S-SCTP SDL diagram 3

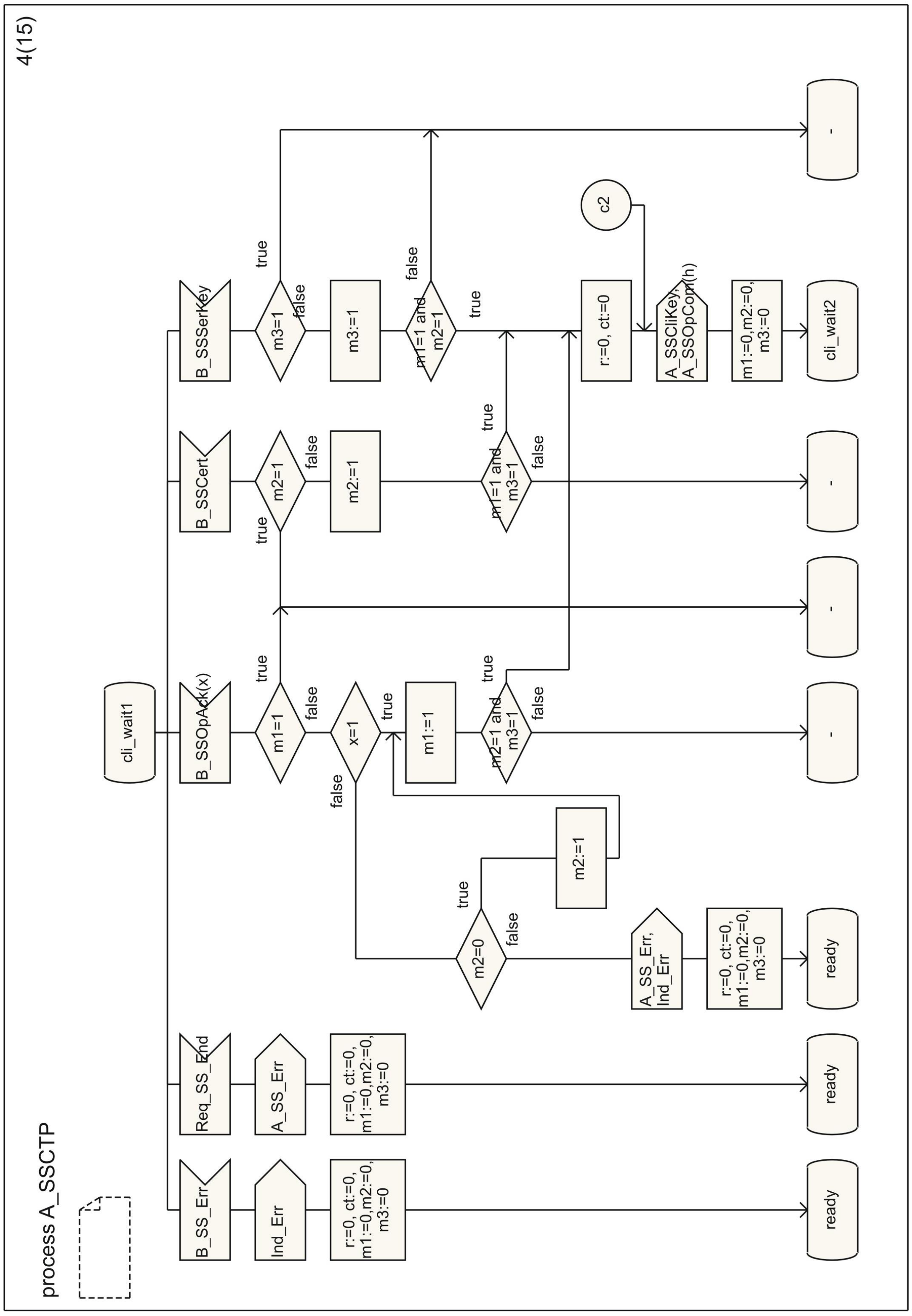


Figure A.6: S-SCTP SDL diagram 4

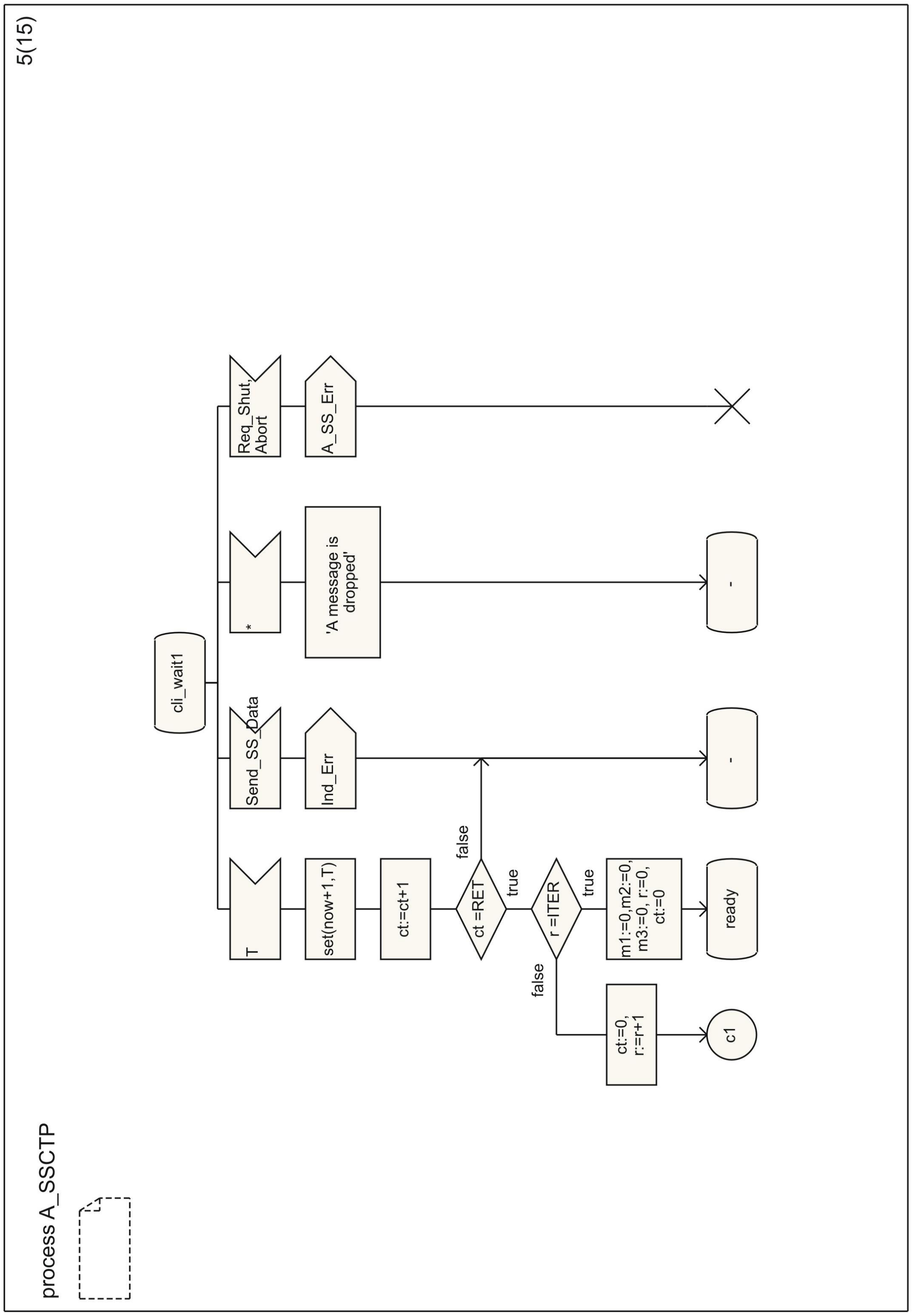


Figure A.7: S-SCTP SDL diagram 5

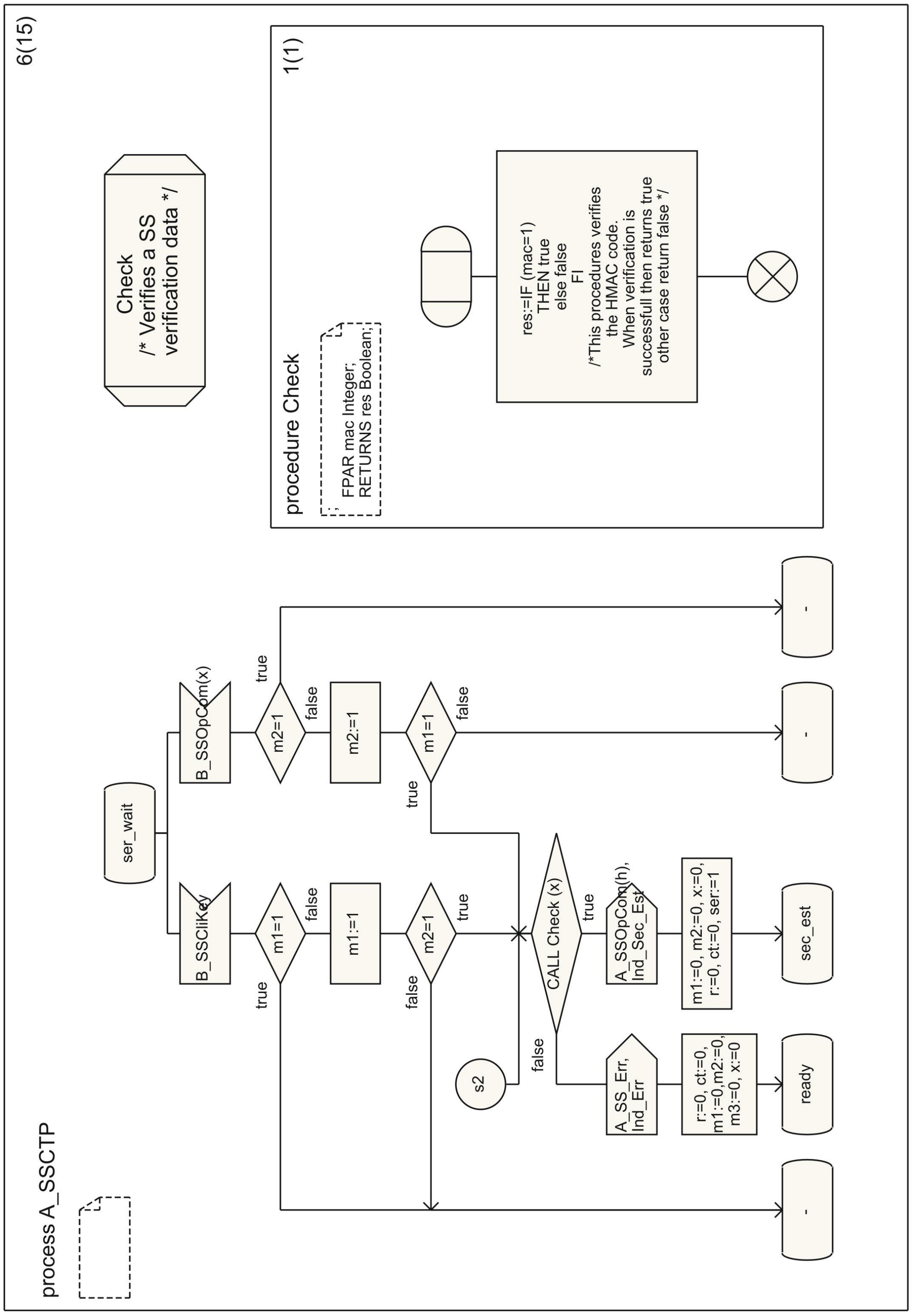

6(15)
process A_SSCTP
ser_wait
B_SSCliKey
B_SSOpCom(x)
m1=1
m2=1
m1:=1
m2:=1
true
false
s2
CALL Check (x)
A_SSOpCom(h), Ind_Sec_Est
A_SS_Err, Ind_Err
m1:=0, m2:=0, x:=0, r:=0, ct:=0, ser:=1
r:=0, ct:=0, m1:=0,m2:=0, m3:=0, x:=0
sec_est
ready
-
Check /* Verifies a SS verification data */
procedure Check
1(1)
FPAR mac Integer; RETURNS res Boolean;
res:=IF (mac=1) THEN true else false FI /*This procedures verifies the HMAC code. When verification is successfull then returns true other case return false */

Figure A.8: S-SCTP SDL diagram 6

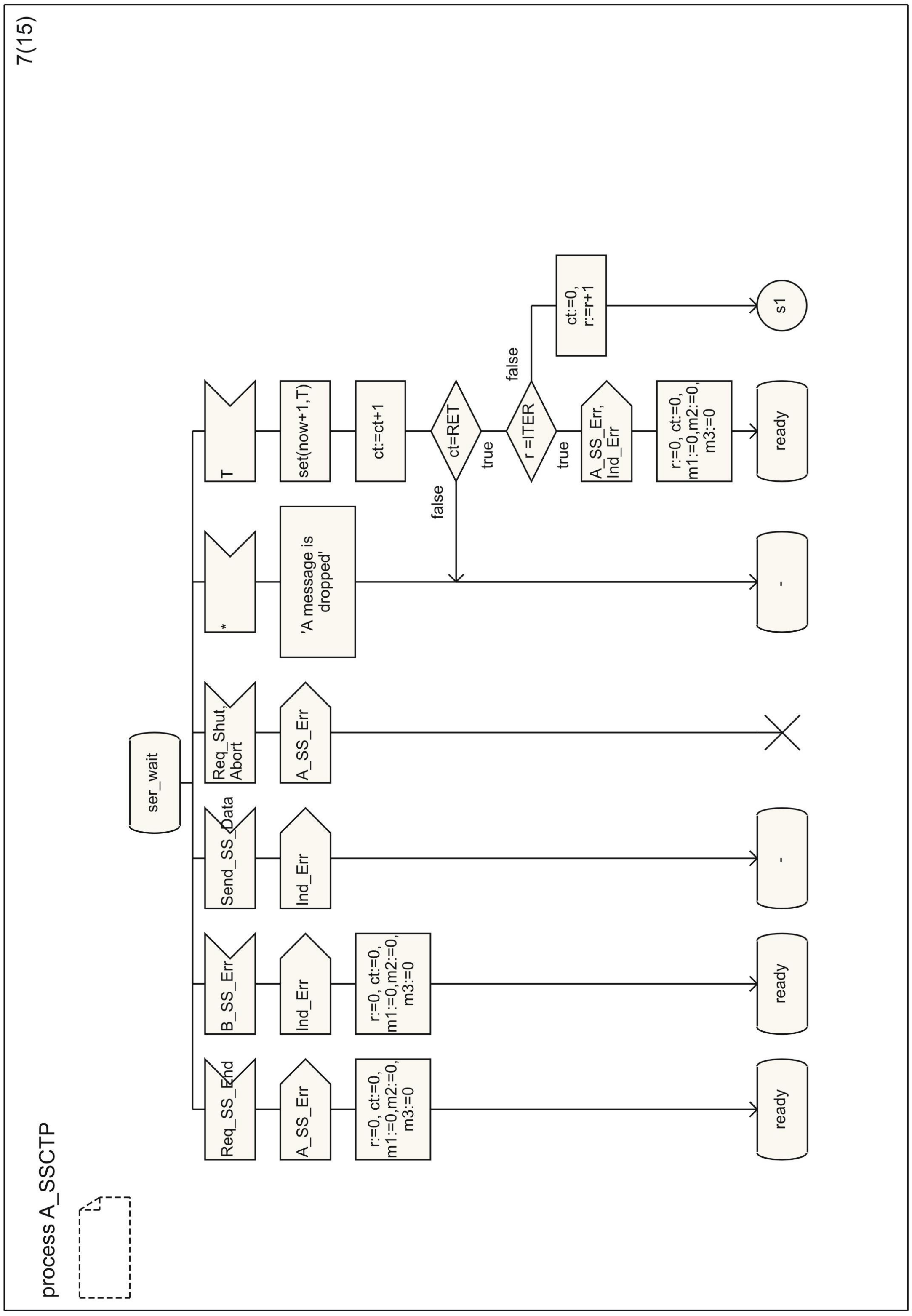


Figure A.9: S-SCTP SDL diagram 7

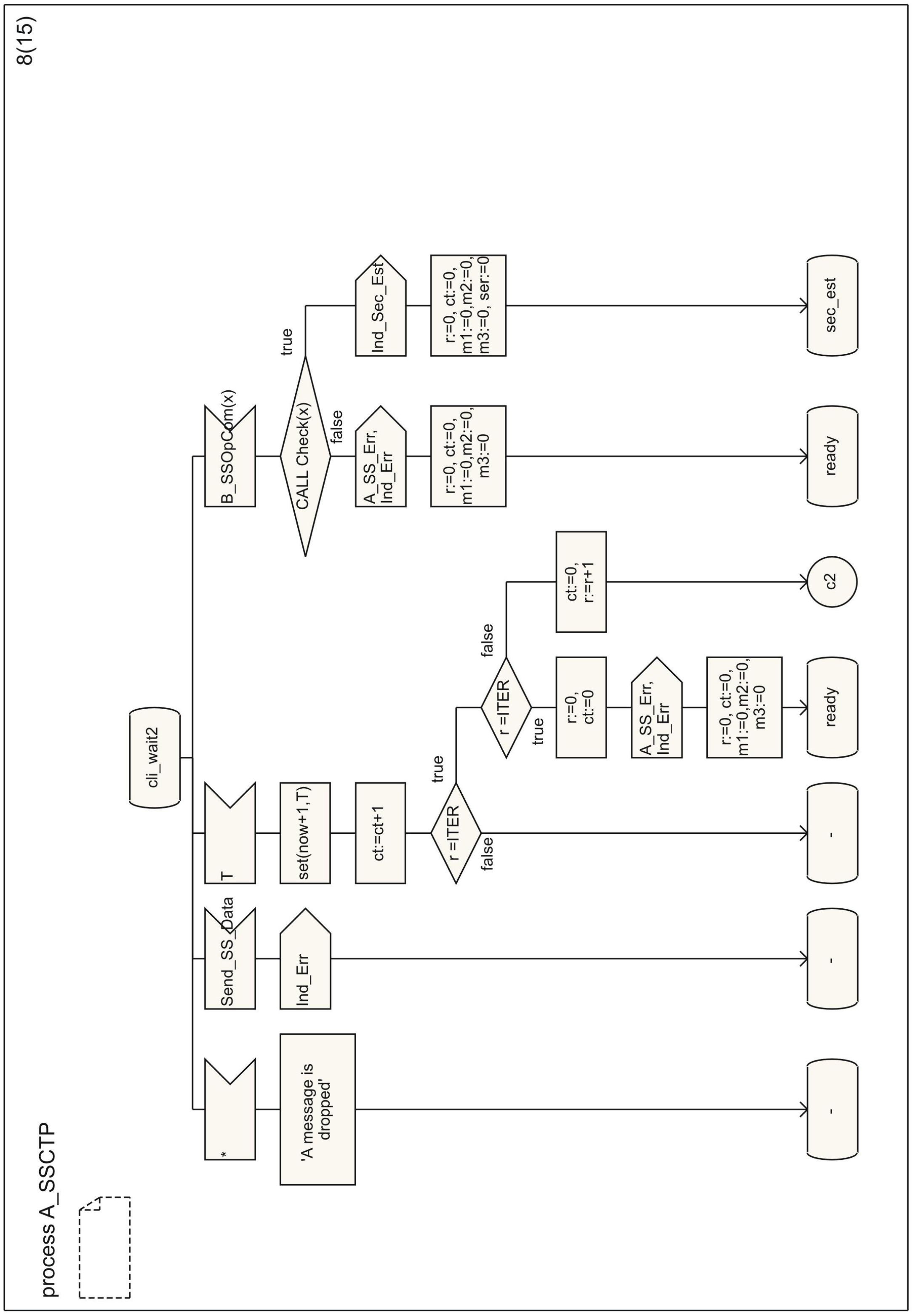


Figure A.10: S-SCTP SDL diagram 8

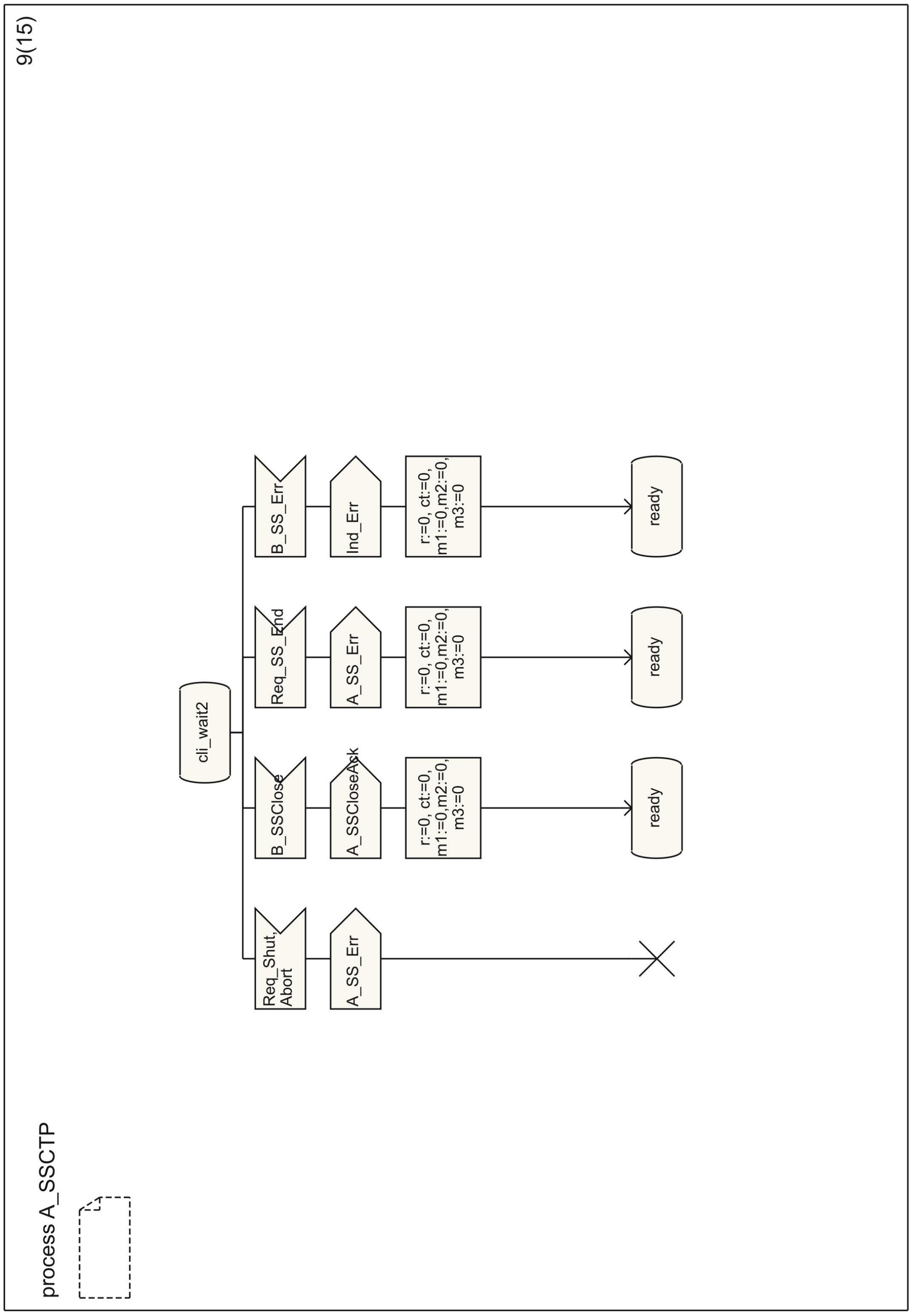


Figure A.11: S-SCTP SDL diagram 9

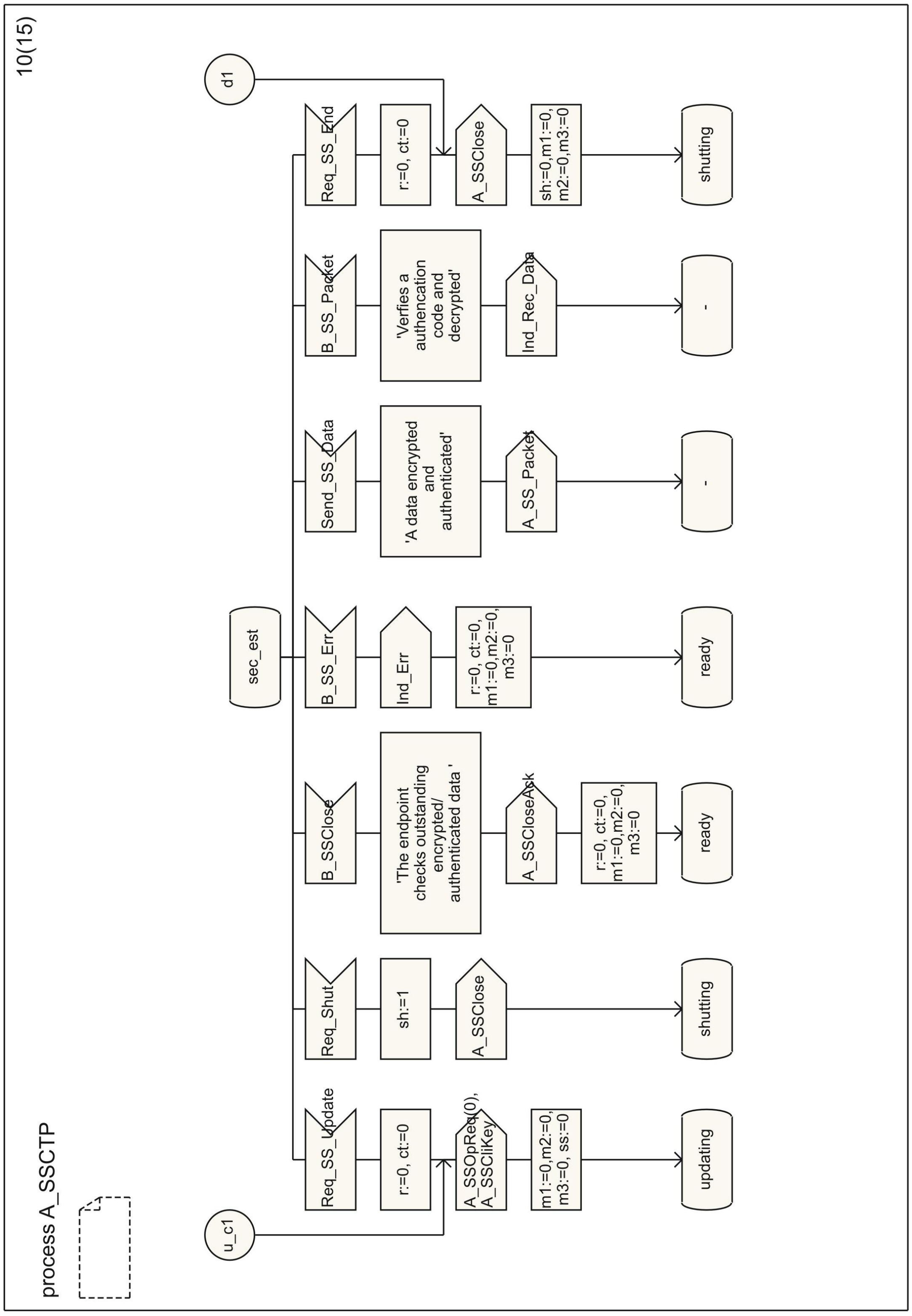


Figure A.12: S-SCTP SDL diagram 10

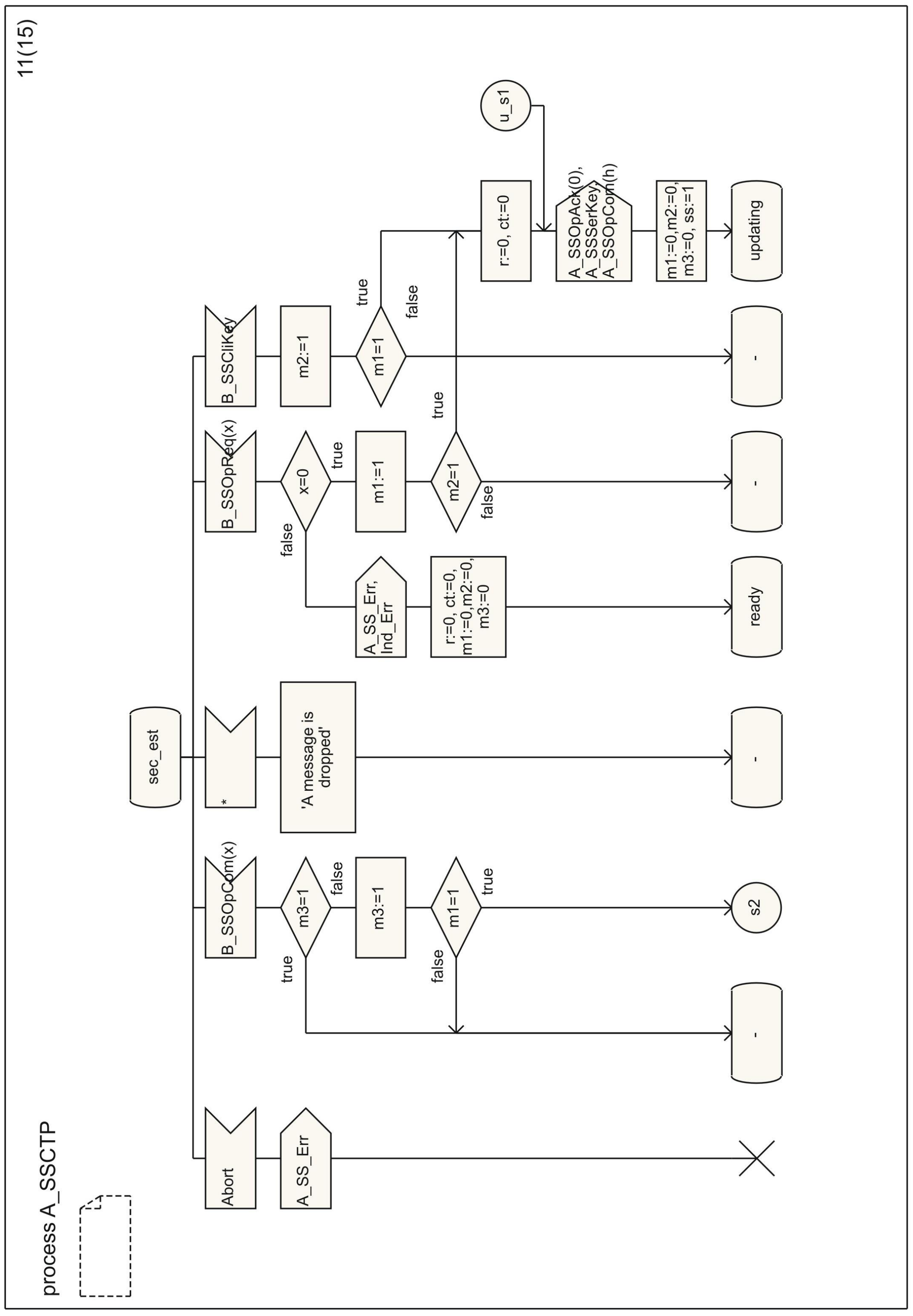


Figure A.13: S-SCTP SDL diagram 11

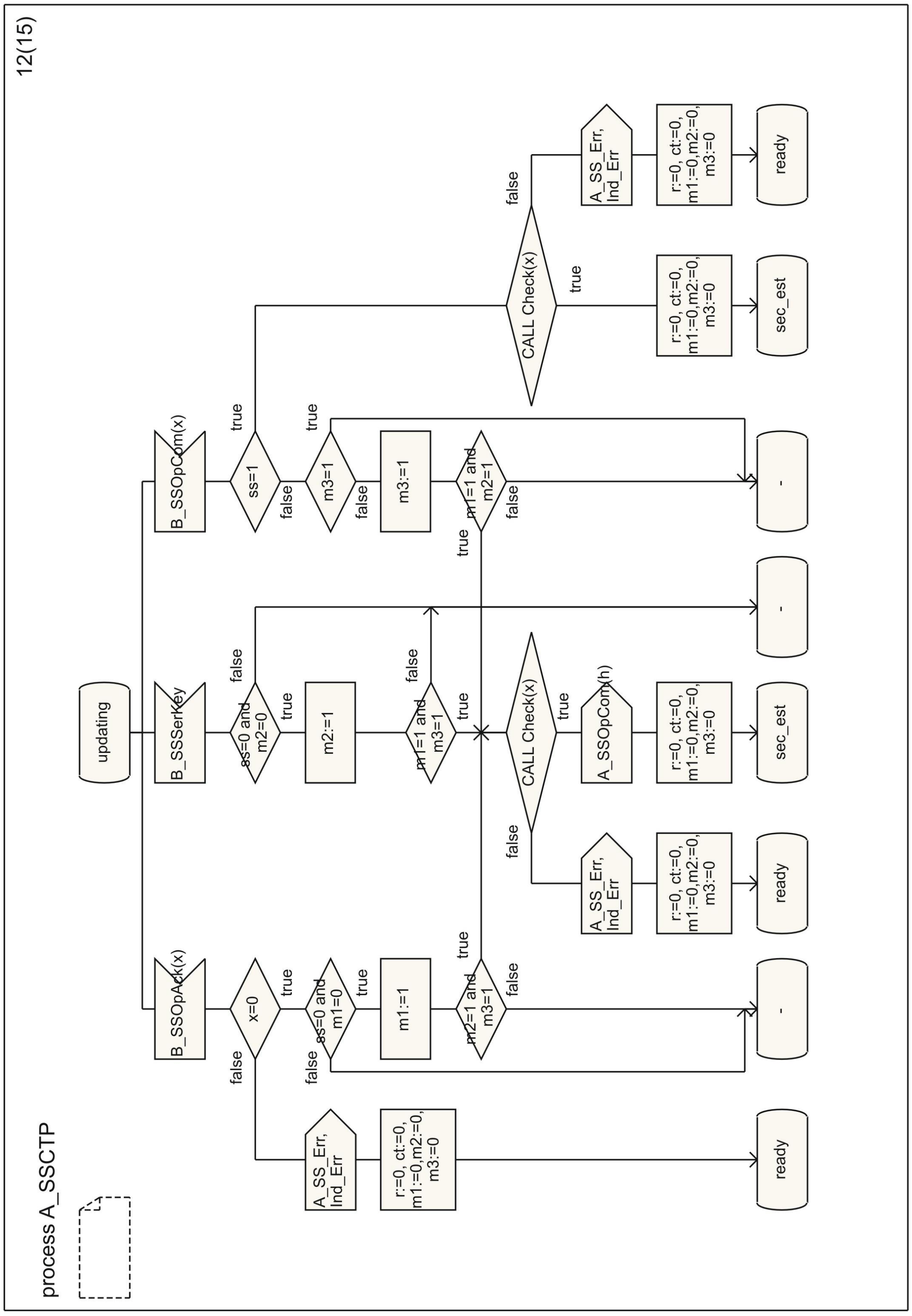


Figure A.14: S-SCTP SDL diagram 12

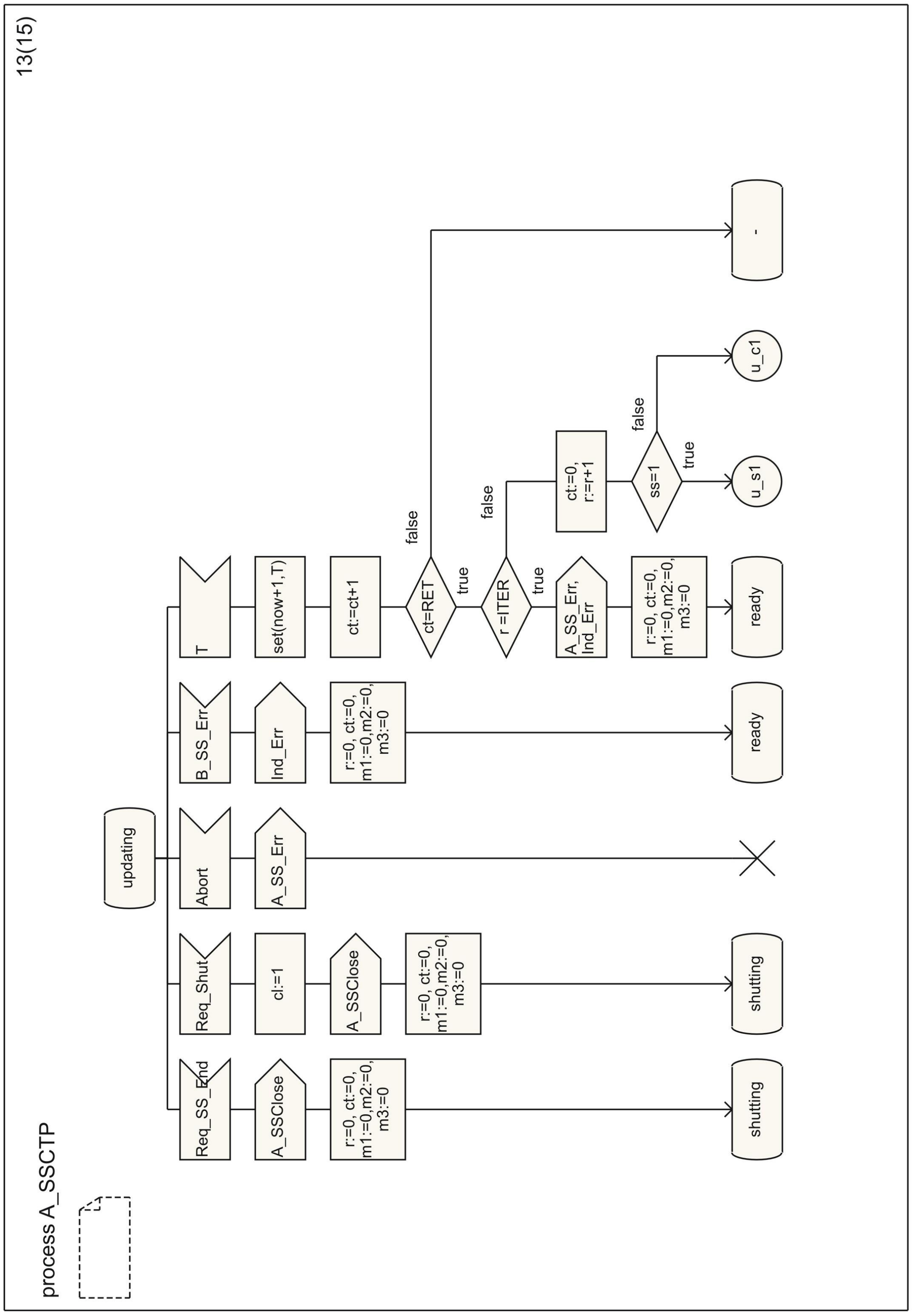


Figure A.15: S-SCTP SDL diagram 13

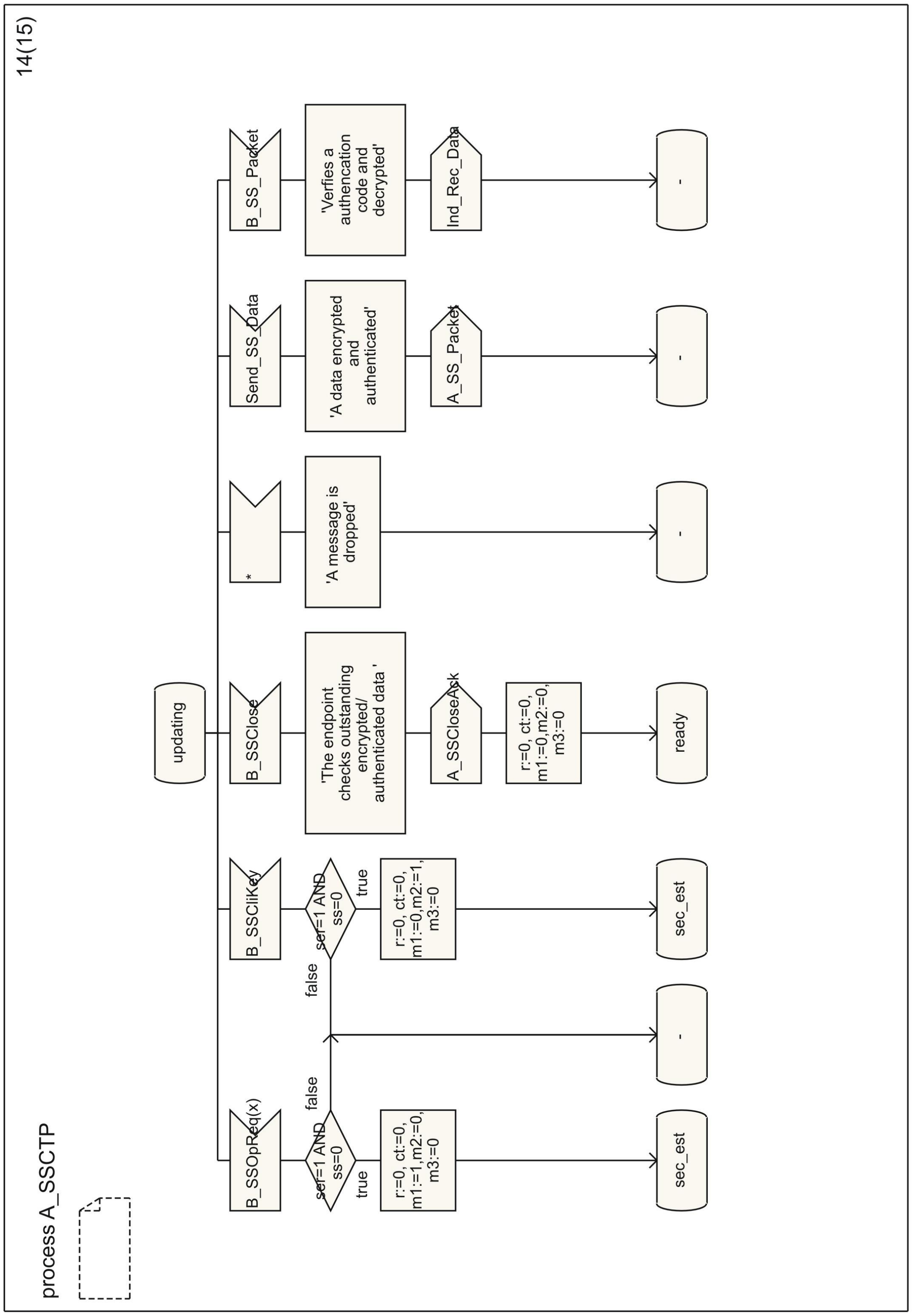

process A_SSCTP
14(15)
updating
B_SSOpReq(x)
ser=1 AND ss=0
true
false
r:=0, ct:=0, m1:=1,m2:=0, m3:=0
sec_est
-
B_SSCliKey
ser=1 AND ss=0
true
false
r:=0, ct:=0, m1:=0,m2:=1, m3:=0
sec_est
B_SSClose
'The endpoint checks outstanding encrypted/ authenticated data '
A_SSCloseAck
r:=0, ct:=0, m1:=0,m2:=0, m3:=0
ready
*
'A message is dropped'
-
Send_SS_Data
'A data encrypted and authenticated'
A_SS_Packet
-
B_SS_Packet
'Verfies a authencation code and decrypted'
Ind_Rec_Data
-

Figure A.16: S-SCTP SDL diagram 14

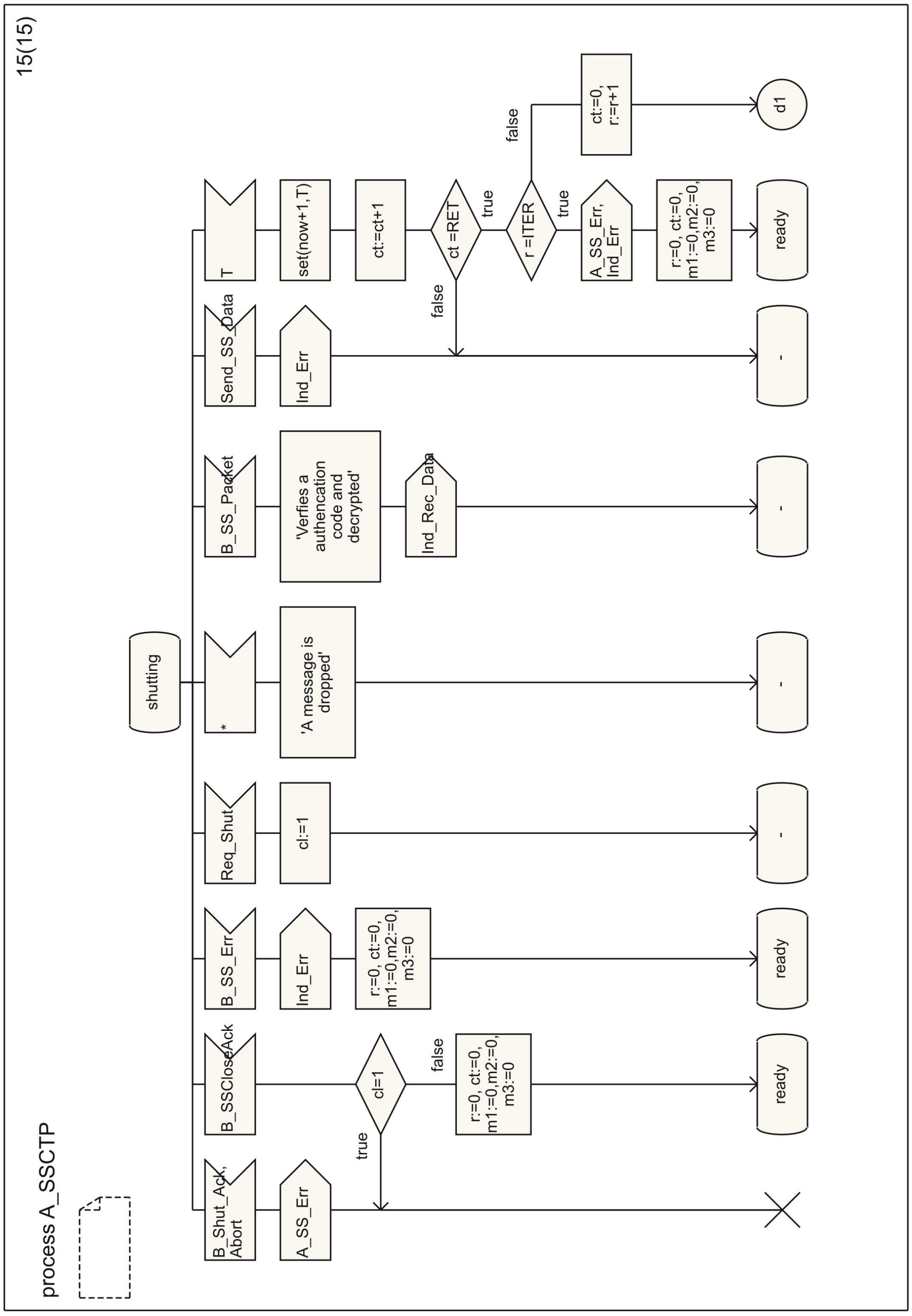


Figure A.17: S-SCTP SDL diagram 15

Appendix B

Technical report of Secure SCTP (S-SCTP)

This section 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 RFC2960, is designed to integrate crypto functions into SCTP.

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 address. SCTP provides unordered delivery, so that a receiver immediately delivers user data to the upper layers upon receipt. For more details, see RFC2960 [75].

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

The main issues for the security solutions TLS over SCTP [37] and SCTP over IPSec [12] 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. 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 IP-sec, is not so flexible and other hand SCTP over IPsec has to be establish new security associations (SA) for new 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.

Figure B.1 shows the S-SCTP architecture. The new blocks (which are not part of standard SCTP) are responsible for the S-SCTP handshake procedure, update secret keys, assortment of data chunks, encryption, decryption and authentication. S-SCTP endpoints are named client and server, as for an SCTP association.

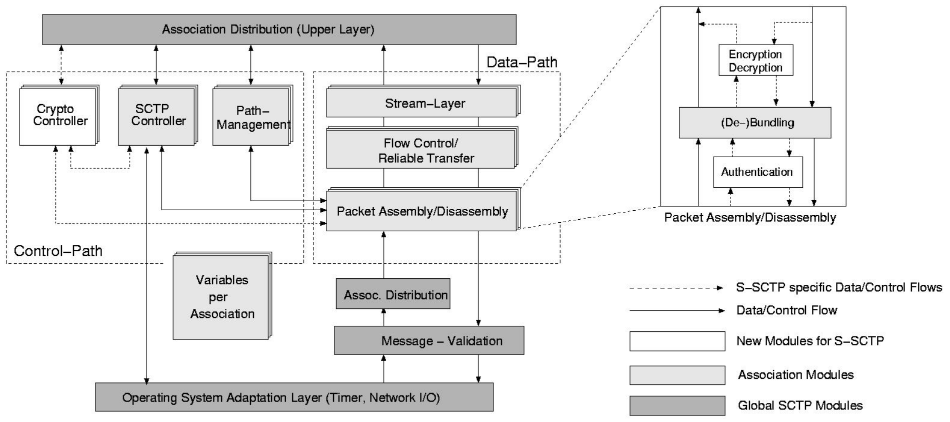


Figure B.1: Secure SCTP architecture

S-SCTP negotiation begins after the SCTP association's establishment.

B.1 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 require 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.

B.2 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 RFC2960.

B.2.1 New type chunks and definitions

Table B.1 shows the new chunks. All new chunks, except for the Encrypted Data (Enc-Data) 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 2960, by the receiver.

Chunk type	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

Table B.1: New chunk types

The new parameters are defined in the current chunk definition section.

Secure Session Open Request chunk (SSOpReq)

An endpoint creates the Secure Session Open Request chunk (table B.2) when it wishes to establish a secure session. The SSOpReq chunk also can be used to update the master secret key or cipher suite after a secure session establishment (see section B.6.6). The chunk can be bundled with other chunks. 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.

Type=0xD0	Reserved=0	CF	Length
Version			Key material length
Optional parameter(s)			

Table B.2: Secure Session Open Request chunk format

- 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 error and terminates a 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 (table B.3). 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. Section B.6.2 describes selection of the cipher suite for a secure session.

Type=30	Length
Cipher suite 1	Cipher suite 2
- - - - - - -	- - - - - - -
Cipher suite N-1	Cipher suite N

Table B.3: Cipher suite parameter

- Cipher suite: 16 bits unsigned integer
  This field indicates a cipher suite, which is supported by the client. Table B.4 includes cipher suites supported in S-SCTP. Additional cipher suites can be specified.

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.

Value	Cipher suite	Key exchange algorithm	Encryption algorithm	Hash algorithm
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

Table B.4: Cipher suites

Compression method parameter (table B.5). This parameter contains compression methods, which are used for data compression. The data compression uses lossless compression methods. The user can choose several compression methods and sends it to the receiver. The receiver choose a one compression method. Selection of the compression method described in section B.6.2.

Type=31		Length	
Compression method 1	Compression method 2	Compression method 3	Compression mwthod 4
- - - - - - -	- - - - - - -	Compression method N-1	Compression method N

Table B.5: Compression parameter

- Compression method: 8 bits unsigned char
  This field indicates a compression method, which is supported by the client. Table B.6 includes compression methods supported in S-SCTP. Additional compression methods can be specified.

Value	Compression method
1	Huffman code
2	Lempel-Ziv code

Table B.6: Compression methods

The secure session uses null compression when the SSOpReq chunk contains no compression parameters. Section B.6.2 contains a detailed explanation how to choose a compression method when the compression parameter contains many compression methods.

Secure Session Certificate chunk: (SSCert)

This chunk (table B.7) can be sent by both endpoints. The certificate helps to authenticate the endpoint, that establishes a S-SCTP session. This chunk contains only parameter, the certificate parameter. The SSCert chunk is optional. For security reasons, both endpoints SHOULD use a certificate to authenticate each other.

Type=0xD1	Reserved=0	Length
Certificate		
Optional parameter		

Table B.7: Secure Session Certificate chunk format

- 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 RFC2459.

- Optional parameter:
  SSCert chunk has only one optional parameter.

Certificate parameter. The SSCert chunk uses the certificate parameter for additional certificates (table B.8), when the endpoint has two or more certificates.

Type=0x33	Length
Certificate	

Table B.8: Certificate parameter format

- Certificate: Variable length
  The endpoint can send two or more certificates. This case, the certificate field contains the endpoint's additional certificate. S-SCTP uses the X.509v3 certificate, which is defined in RFC2459.

Secure Session Open Acknowledge chunk (SSOpReq_Ack)

The Secure Session Open Acknowledge chunk (table B.9) is sent by the server to tell the client that the secure session open request is acceptable.

- 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.

Type=0xD2	Reserved=0	CF	Length
Version			Cipher suite
Compression method			Reserved=0

Table B.9: Secure Session Open Acknowledge chunk format

- 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 have 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: 8 bits unsigned char
  This field indicates the compression method, which is used for data compression in the secure session.

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. Table B.10 shows the chunk header.

- Security level (SL): 2 bits
  This 2-bit value indicates a server's security level of the reserved flags.

Type=0xD3	Reserved=0	SL	Length
Parameter			

Table B.10: Secure Session Server Key chunk format

- 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 tables B.11 and B.12. When a user specifies a new cipher suite with a new key exchange algorithm, then they must define a new parameter.

Diffie-Hellman parameter (table B.11). The SSSerKey chunk uses this parameter when the handshake is done via the DH key exchange algorithm.

Type=0xD001	Length
Length of DH prime number, P	Length of DH primitive root, R
Length of DH public key, Y	Reserved=0
DH prime number, P	
DH primitive root, R	
DH value, Y	
Signature	

Table B.11: Server DH parameter format

- 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 (table B.12). The SSSerKey chunk uses this parameter when the handshake uses the RSA key exchange algorithm.

Type=0xD002	Length
Encrypted(random number, R)	
Signature	

Table B.12: Server RSA parameter format

- 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 (see section B.6.5). The random number generation algorithm is defined in section B.6.7. 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.

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. Table B.13 shows the chunk.

Type=0xD4	Reserved=0	SL	Length
Parameter			

Table B.13: Secure Session Client Key chunk format

- Security level (SL): 2 bits
  This 2-bit value indicates a client's security level.

- Length: 16 bit unsigned integer
  The length field contains 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 tables B.14 and B.15.

Diffie-Hellman parameter (table B.14). The SSCliKey chunk uses this parameter when the handshake uses the DH key exchange algorithm.

Type=0xD003	Length
DH value, Y	
Signature	

Table B.14: Client DH parameter format

- 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 (table B.15). The SSCliKey chunk uses this parameter when the handshake uses RSA key exchange algorithm.

Type=0xD003	Length
Encrypted (Random number, R)	
Signature	

Table B.15: Client RSA parameter format

- 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 (see section B.6.5). The random number generation algorithm is defined in section B.6.7.

- 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.

Secure Session Open Complete chunk (SSOpCom)

This chunk (table B.16) 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.

Type=0xD5	Reserved=0	Length
Verification data		

Table B.16: Secure Session Open Complete chunk format

- 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.

Secure Session Close chunk (SSClose)

This chunk (table B.17) 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.

Type=0xD6	Reserved=0	OF	Length
TSN			

Table B.17: Secure Session Close chunk format

- 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.

Secure Session Close Acknowledge chunk (SSClose_Ack)

This chunk (table B.18) 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.

Type=0xD7	Reserved=0	Length=4

Table B.18: Secure Session Close Acknowledge chunk format

Security Level Changed chunk (SecLevCHD)

This chunk (table B.19) 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.

Type=0xD8	Reserved=0	SL	Length=4

Table B.19: Security Level Changed chunk format

- Security level (SL): 2 bits
  This 2-bit value indicates the sender's new security level.

Security Level Changed Acknowledge chunk (SecLevCHD_Ack)

This chunk (table B.20) is used to acknowledge the receipt of the SecLevCHD chunk. When the endpoint receives the SecLevCHD chunk, it immediately sends the SecLevCHD_Ack chunk.

Type=0xD9	Reserved=0	Length=4

Table B.20: Security Level Changed Acknowledge chunk format

Encrypted Data Chunk (EncData)

Each S-SCTP packet includes at most one encrypted data chunk (table B.21), 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. The packet format is defined in section B.2.

Type=0x10	Reserved=0	Length
Master secret key reference number		Random number length
Random number		
Encrypted data		

Table B.21: Encrypted data chunk format

- 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 keeping 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 fulfill 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.

Padding chunk (PADDING)

This padding chunk (table B.22) is used with EncData chunk. The symmetric encryption algorithms use a block oriented encryption of the user data. For example DES uses a 64 bit blocks, and AES uses a 128 bit block. 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 byte 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 byte 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.

Type=0x12	Reserved=0	Length
Padding		

Table B.22: Padding data chunk format

- 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.

Authentication chunk (AUTH)

This chunk (table B.23) is dedicated to the authentication of an S-SCTP packet. S-SCTP inserts this chunk into the packet when the security level requires authentication.

Type=0x11	Reserved=0	Length
Master secret key reference		Reserved=0
HMAC		

Table B.23: Authentication chunk format

- 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 keeping 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.

B.3 New error causes

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. Table B.24 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

Table B.24: New error causes

B.3.1 Secure Session failure

Cause Code=0x20	Cause length=8
Error code	Reserved=0

Table B.25: Secure Session failure format

If any error happens in the secure session open and update process, endpoints alert their peers with these error codes. Table B.26 shows error codes for what can happen.

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

Table B.26: Secure Session Open errors

- 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.

B.3.2 Secure Session Certificate failure

Cause Code=0x21	Cause length=8
Error code	Reserved=0

Table B.27: Secure Session Certificate failure format

The certificate failure signals that an error has occurred in processing the certificates. Table B.28 shows error codes for what can happen.

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

Table B.28: Secure Session Certificate failure code

- 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 is either unidentified, or is not an X.509v3 certificate.

B.3.3 Decryption failure

This error (table B.29) 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 to a limit, the secure session is terminated. The limit of the errors depends on the implementation.

Cause Code=0x22	Cause length=4

Table B.29: Decryption failure format

B.3.4 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 (table B.30).

Cause Code=0x23	Cause length=4

Table B.30: Authentication failure format

B.3.5 Decompression failure

This error (table B.31) 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).

Cause Code=0x24	Cause length=4

Table B.31: Decompression failure format

B.4 S-SCTP packet format and security levels

B.4.1 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 or 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. See section B.9 for more details.

B.5 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 belong to encryption group are concatenated and encrypted into EncData chunk. May be a PADDING chunk inserted in the encryption group. Insertion on of a PADDING chunk is done depending on data length and encryption block size (see figure B.2).

In figure B.2, the chunks which are to be encrypted are marked with dark background color.

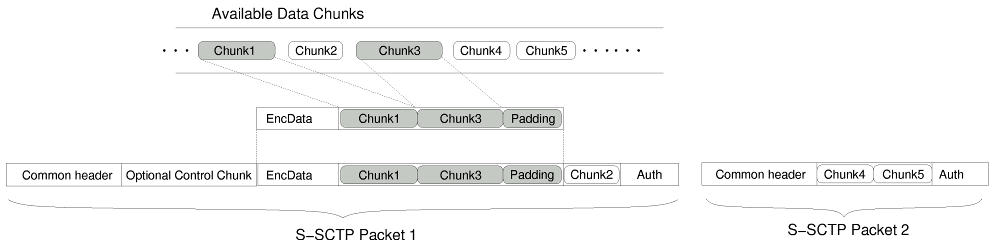


Figure B.2: S-SCTP packet format

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, a SCTP packet construction is slightly different. The figure B.2 shows the packet format in security level 2. In security level 0 the packet format is same as the SCTP packet format.

B.6 Procedures

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

B.6.1 Establishment of a secure session

The following process (figure B.3) 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 are 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 RFC2960. 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 2960).

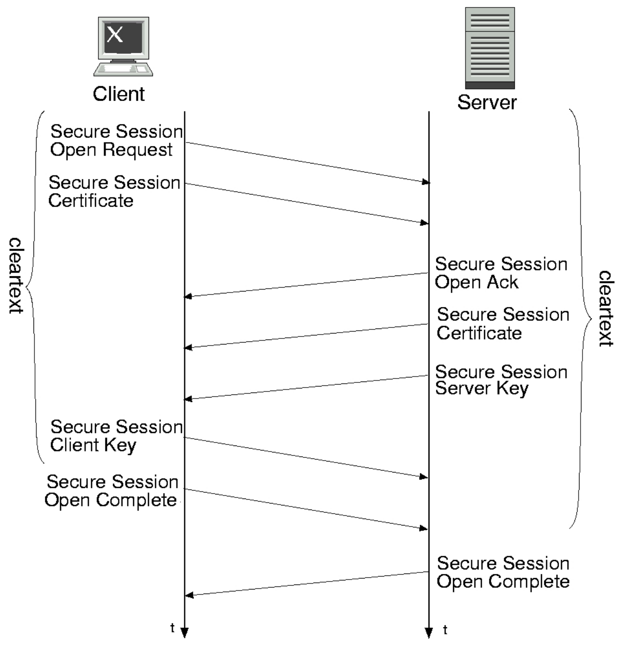

Client
Server
Secure Session Open Request
Secure Session Certificate
Secure Session Open Ack
Secure Session Certificate
Secure Session Server Key
Secure Session Client Key
Secure Session Open Complete
Secure Session Open Complete
cleartext
cleartext
t
t

Figure B.3: S-SCTP initialisation handshake

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 RFC2960. 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.

1. 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.

2. 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 (see section B.6.2) 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. More details are described in section B.6.6. 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 drop second SSCert chunk.

3. The client receives the handshake messages and checks the certificate in the SSSerKey chunk. If the client detects any error, 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.

4. 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.

B.6.2 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 indicated 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:

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

2. 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 looks, its first cipher suite, for a match in the cipher suite sequence of the client. When does not matches, the server looks, the client's first cipher suite, for match in itself 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 matches the cipher suite 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.

3. 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.

B.6.3 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.

B.6.4 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 (see SS-Close chunk format B.17). 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 SS-Close_Ack chunk. After receiving the SSClose_Ack chunk, the association clears the TCB parameters belonging to the secure session (see section B.9). 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.

B.6.5 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.

Client's random number						Server's random number					
DH secret key											
D1	D2	...	...	...	Dn	D(n+1)	...	...	...	...	D2n

Table B.32: Key material is split into DES blocks

$Key: \ D - Data\ block$
$IV - initialisation\ vector, \ IV = D_{2n}$

$$K = key\ for\ the\ 3DES\ encryption,\ K_1 = D_1,\ K_2 = D_2$$

The following formula is used for secret key generation.

$$C_i = E_{3DES}(D_{(i+1)}, IV, K)IV = C_{(i-1)}(0 < i < 2_{(n-2)})$$

The outputs of the last block are used for the master secret (secret key for the data encryption) and HMAC secret keys (authentication key for the S-SCTP packet).

B.6.6 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 (figure B.4). 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 using allways 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 to the peer an error. 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 to the peer an error. 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.

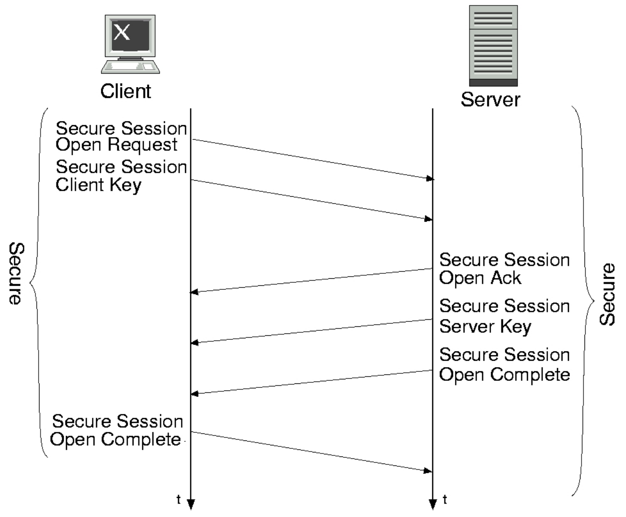

Client
Server
Secure Session Open Request
Secure Session Client Key
Secure Session Open Ack
Secure Session Server Key
Secure Session Open Complete
Secure Session Open Complete
Secure
Secure
t
t

Figure B.4: S-SCTP abbreviated handshake

The new master secret key generation uses the same algorithm as described in section B.6.5. 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.

B.6.7 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 RFC1750 [22].

B.6.8 HMAC algorithm

S-SCTP uses the HMAC algorithm which is defined in RFC 2104 for the packet authentication. The HMAC secret key generation is defined in section B.6.5.

B.7 ULP to S-SCTP

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 RFC2960. The text written in italic indicates the respective functions that have been developed in my implementation. 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))$\rightarrow$ 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)* $\rightarrow$ *result*

- secure session ID: local handle to the secure session
- security level: This parameter indicates the new security level

GETSECLEVEL: This pritimive gets the current security level of a secure session.
Format: *getSecLevel(secure session ID)* $\rightarrow$ *security level*

- secure session ID: local handle to the secure session

SENDSEC: This primitve 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 RFC2960 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 primivite 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.

B.8 S-SCTP to ULP

S-SCTP defines new notifications to deliver information to the upper layer. The notifications extend the section 10.2 of RFC2960 [75]. The text written in italic indicates the respective functions that have been developed in my implementation. 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 SCTP DATA chunk, a one new flag defined which indicates that the data is compressed or not. On the receiver side there are no changes.

B.9 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 RFC2960.

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 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.

B.10 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 [78] 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.

Bibliography

[1] *Cryptolib Project*. http://www.cs.auckland.ac.nz/ pgut001/cryptlib.

[2] *FreeS/WAN Project*. http://www.freeswan.org.

[3] *KAME Project*. http://www.kame.net.

[4] *yaSSL Project*. http://www.yassl.com.

[5] *New directions in Cryptography*, November 1976.

[6] *A Method for Obtaining Digital Signatures and Public Key Cryptosystems*, February 1978.

[7] *Performance Evaluation of the Stream Control Transmission Protocol*, 2000.

[8] *On the Use of SCTP in Failover-Scenarios*, July 2002. Orlanda/USA.

[9] *A New Scheme for IP-based Internet Mobility*, October 2003. Koeln/Germany.

[10] B. Aboba, P. Calhoun, S. Glass, T. Hiller, P. McCann, H. Shiino, P. Walsh, G. Zorn, G. Dommety, C. Perkins, B. Patil, D. Mitton, S. Manning, M. Beadles, Alcatel X. Chen, S. Sivalingham, A. Hameed, M. Munson, S. Jacobs, B. Lim, B. Hirschman, R. Hsu, H. Koo, M. Lipford, E. Campbell, Y. Xu, S. Baba, and E. Jaques. *Criteria for Evaluating AAA Protocols for Network Access*. IETF, Network Working Group, November 2000. RFC2989.

[11] B. Aboba and J. Wood. *Authentication, Authorization and Accounting (AAA) Transport Profile*. IETF, Network Working Group, June 2003. RFC3539.

[12] S. Bellovin, J. Ioannidis, A. Keromytis, and R. Stewart. *On the Use of Stream Control Transmission Protocol (SCTP) with IPsec*. IETF, Network Working Group, July 2003. RFC3554.

[13] R. Braden and Ed. *Requirements for Internet Hosts - Communication Layers*. IETF, Network Working Group, October 1989. RFC 1122.

[14] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko. *Diameter Base Protocol*. IETF, AAA Working Group, September 2003. RFC3588.

[15] Pat R. Calhoun, Tony Johansson, Charles E. Perkins, and Tom Hiller. *Diameter Mobile IPv4 Application*. IETF, AAA Working Group, February 2004. draft-ietf-aaa-diameter-mobileip-16.txt, work in progress.

[16] Pat R. Calhoun, Glen Zorn, David Spence, and David Mitton. *Diameter Network Access Server Application*. IETF, AAA Working Group, February 2004. draft-ietf-aaa-diameter-nasreq-14.txt, work in progress.

[17] P. Chown. *Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)*. IETF, Network Working Group, June 2002. RFC3268.

[18] Netscape Communicator. *Secure socket layer application program apparatus and method*. United States Patent, August 1997. http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=/netahtml/ srch-num.htm&r=1&f=G&l=50&s1='5657390'.WKU.&OS=PN/5657390&RS =PN/5657390 ,US Patent.

[19] Computer Networking Technology Group of Duisburg-Essen University, Germany. *SCTP prototype implementation*. http://www.sctp.de.

[20] F. Cuervo, N. Greene, A. Rayhan, C. Huitema, B. Rosen, and J. Segers. *Megaco Protocol Version 1.0*. IETF, Network Working Group, November 2000. RFC3015.

[21] T. Dierks and C. Allen. *The TLS protocol*. IETF, Network Working Group, January 1999. RFC2246.

[22] D. Eastlake, S. Crocker, and J. Schiller. *Randomness Recommendations for Security*. IETF, Network Working Group, December 1994. RFC1750.

[23] Ralf S. Engelshall. *OPENSSL Project*. http://www.openssl.org.

[24] P. Eronen, T. Hiller, and G. Zorn. *Diameter Extensible Authentication Protocol (EAP) Application*. IETF, AAA Working Group, February 2004. draft-ietf-aaa-eap-04.txt, work in progress.

[25] P. Ferguson and D. Senie. *Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing*. IETF, Network Working Group, January 1998. RFC2267.

[26] Free Software Foundation. *GNUTLS Project*. http://www.gnutls.org.

[27] Free Software Foundation. Gnu lesser general public license, February 1999. Version 2.1.

[28] S. Frankel, R. Glenn, and S. Kelly. *The AES-CBC Cipher Algorithm and Its Use with IPsec*. IETF, Network Working Group, September 2003. RFC3602.

[29] M. Garcia-Martin, M. Belinchon, M. Pallares-Lopez, C. Canales, P. McCann, J. Rajaniemi, and K. Tammi. *Diameter Session Initiation Protocol (SIP) Application*. IETF, AAA Group, April 2004. draft-ietf-aaa-diameter-sip-app-02.txt, work in progress.

[30] Tom George, Brian Bidulock, Ram Dantu, Hanns Juergen Schwarzbauer, and Ken Morneault. *SS7 MTP2-User Peer-to-Peer Adaptation Layer*. IETF, Network Working Group, January 2004. draft-ietf-sigtran-m2pa-11.txt, work in progress.

[31] Harri Hakala, Leena Mattila, Juha-Pekka Koskinen, Marco Stura, and John Loughney. *Diameter Credit-Control Application*. IETF, AAA Working Group, February 2004. draft-ietf-aaa-diameter-cc-03.txt, work in progress.

[32] M. Handley and V. Jacobson. *SDP: Session Description Protocol*. IETF, Network Working Group, Aprel 1998. RFC2327.

[33] M. Handley, C. Perkins, and E. Whelan. *Session Announcement Protocol*. IETF, Network Working Group, October 2000. RFC2719.

[34] D. Harkins and D. Carrel. *The Internet Key Exchange (IKE)*. IETF, IP Security Protocol Working Group, November 1998. RFC2409.

[35] R. Housley, W. Polk, W. Ford, and D. Solo. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. IETF, Network Working Group, April 2002. RFC3280.

[36] International Telecommunication Union (ITU). *Specification and Description Language*, November 1999. Z.100 Recommendation.

[37] A. Jungmaier, E. Rescorla, and M. Tuexen. *Transport Layer Security over Stream Control Transmission Protocol*. IETF, Network Working Group, December 2002. RFC3436.

[38] S. Kent and R. Atkinson. *IP Authentication Header*. IETF, IP Security Protocol Working Group, November 1998. RFC2402.

[39] S. Kent and R. Atkinson. *IP Encapsulating Security Payload (ESP)*. IETF, IP Security Protocol Working Group, November 1998. RFC2406.

[40] S. Kent and R. Atkinson. *Security Architecture for the Internet Protocol*. IETF, IP Security Protocol Working Group, November 1998. RFC2401.

[41] H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*. IETF, Network Working Group, February 1997. RFC2104.

[42] J. Loughney, G. Sidebottom, L. Coene, G. Verwimp, J. Keller, and B. Bidulock. *Signalling Connection Control Part User Adaptation Layer (SUA)*. IETF, Network Working Group, December 2003. draft-ietf-sigtran-sua-16.txt, work in progress.

[43] J. Loughney, M. Tuexen, and J. Pastor-Balbas. *Security Considerations for SIGTRAN Protocols*. IETF, Network Working Group, June 2003. draft-ietf-sigtran-security-03.txt, work in progress.

[44] C. Madson and N. Doraswamy. *The ESP DES-CBC Cipher Algorithm with Explicit IV*. IETF, IP Security Protocol Working Group, November 1998. RFC2405.

[45] D. Maughan, M. Schertler, M. Schneider, and J. Turner. *Internet Security Association and Key Management Protocol (ISAKMP)*. IETF, IP Security Protocol Working Group, November 1998. RFC2408.

[46] D. McDonald, C. Metz, and B. Phan. *PF_KEY Key Management API, Version 2*. IETF, Network Working Group, July 1998. RFC2367.

[47] J. Mogul and S. Deering. *Path MTU Discovery*. IETF, Network Working Group, November 1960. RFC1122.

[48] K. Morneault, R. Dantu, G. Sidebottom, B. Bidulock, and J. Heitz. *Signaling System 7 (SS7) Message Transfer Part 2 (MTP2) - User Adaptation Layer*. IETF, Network Working Group, September 2002. RFC3332.

[49] National Institute of Standards and Technology (NIST). *Digital Encryption standard*, January 1988. FIPS PUB 46.

[50] National Institute of Standards and Technology (NIST). *Digital Encryption standard*, January 1988. FIPS PUB 46.

[51] National Institute of Standards and Technology (NIST). *Secure Hash algorithm*, May 1993. FIPS PUB 180.

[52] National Institute of Standards and Technology (NIST). *Digital Signature standard*, January 2000. FIPS PUB 186.

[53] National Institute of Standards and Technology (NIST). *Advanced Encryption Standard*, November 2001. FIPS PUB 197.

[54] H. Orman. *The OAKLEY Key Determination Protocol*. IETF, IP Security Protocol Working Group, November 1998. RFC2412.

[55] Trevor Perrin. *TLS Lite Project*. http://trevp.net/tlslite.

[56] D. Piper. *The Internet IP Security Domain of Interpretation for ISAKMP*. IETF, IP Security Protocol Working Group, November 1998. RFC2407.

[57] J. Postel. *User Datagram Protocol*. IETF, Network Working Group, August 1980. RFC768.

[58] J. Postel. *Transmission Control Protocol*. IETF, Network Working Group, September 1981. RFC793.

[59] Eric Rescorla. *PureTLS Project*. http://www.rtfm.com/puretls.

[60] C. Rigney, S. Willens, A. Rubens, and W. Simpson. *Remote Authentication Dial In User Service (RADIUS)*. IETF, Network Working Group, June 2000. RFC2865.

[61] R. Rivest. *RC4 Cipher Encryption Algorithm*. RSA Security.

[62] R. Rivest. *The MD5 Message-Digest Algorithm*. Network Working Group, April 1992. RFC1321.

[63] R. Rivest. *A Description of the RC2(r) Encryption Algorithm*. IETF, Network Working Group, March 1998. RFC2268.

[64] Marshall T. Rose and Daniel C. Lynch. *Internet System Handbook*. Addison-Wesley, 1993.

[65] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. *SIP: Session Initiation Protocol*. IETF, Network Working Group, Juni 2002. RFC3261.

[66] Randall R.Stewart and Qiaobing Xie. *Stream Control Transmission Protocol*. Addison-Wesley, 2002.

[67] Travis Russel. *Signaling system 7*. McGraw-Hill, 2002.

[68] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. *RTP: A Transport Protocol for Real-Time Application*. IETF, Network Working Group, July 2003. RFC3550.

[69] H. Schulzrinne, A. Rao, and R. Lanphier. *Real Time Streaming Protocol (RTSP)*. IETF, Network Working Group, Aprel 1998. RFC2326.

[70] A. Shacham, B. Monsour, R. Pereira, and M. Thomas. *IP Payload Compression Protocol (IPComp)*. IETF, Network Working Group, November 1998. RFC3173.

[71] G. Sidebottom, K. Morneault, and J. Pastor-Balbas. *Signaling System 7 (SS7) Message Transfer Part 3 (MTP3) - User Adaptation Layer (M3UA)*. IETF, Network Working Group, September 2002. RFC3331.

[72] R. Stewart, L. Ong, I. Arias-Rodriguez, K. Poon, P. Conrad, A. Caro, and M. Tuexen. *Stream Control Transmission Protocol (SCTP) Implementor's Guide*. IETF, Network Working Group, November 2003. draft-ietf-tsvwg-sctpimpguide-10.txt, work in progress.

[73] R. Stewart, M. Ramalho, Q. Xie, M. Tuexen, and P. Conrad. *SCTP Partial Reliability Extension*. IETF, Transport Area Working Group, January 2004. draft-ietf-tsvwg-prsctp-03.txt, work in progress.

[74] R. Stewart, M. Ramalho, Q. Xie, M. Tuexen, I. Rytina, M. Belinchon, and P. Conrad. *Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration*. IETF, Transport Area Working Group, January 2004. draft-ietf-tsvwg-addip-sctp-08.txt, work in progress.

[75] R. Stewart, Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. *Stream Control Transmission Protocol*. IETF, Network Working Group, October 2000. RFC2960.

[76] R. Stewart, Q. Xie, and M. Stillman. *Endpoint Name Resolution Protocol (ENRP)*. IETF, Network Working Group, October 2003. draft-ietf-rserpool-enrp-07.txt, work in progress.

[77] R. Stewart, Q. Xie, M. Stillman, and M. Tuexen. *Aggregate Server Access Protocol (ASAP)*. IETF, Network Working Group, October 2003. draft-ietf-rserpool-asap-08.txt, work in progress.

[78] R. Stewart, Q. Xie, L. Yarroll, J. Wood, K. Fujita, and M. Tuexen. *Sockets API Extensions for Stream Control Transmission Protocol (SCTP)*. IETF, Network Working Group, April 2004. draft-ietf-tsvwg-sctpsocket-08.txt, work in progress.

[79] Maureen Stillman, Ram Gopal, Senthil Sengodan, Erik Guttman, and Matt Holdrege. *Threats Introduced by Rserpool and Requirements for Security in response to Threats*. IETF, Network Working Group, September 2003. draft-ietf-rserpool-threats-02.txt, work in progress.

[80] J. Stone, R. Stewart, and D. Otis. *Stream Control Transmission Protocol (SCTP) Checksum Change*. IETF, Network Working Group, September 2002. RFC3309.

[81] Telelogic Inc. *Telelogic TAU SDL Suite*. http://www.telelogic.com/products/tau/sdl/.

[82] R. Thayer, N. Doraswamy, and R. Glenn. *IP Security Document Roadmap*. IETF, IP Security Protocol Working Group, November 1998. RFC2411.

[83] Stephen Thomas. *SSL and TLS essentials*. Robert Ipsen, 2000. Published by John Wiley & Sons. Inc.

[84] M. Tuexen, R. Stewart, Q. Xie, M. Stillman, M. Shore, and J. Loughney. *Architecture for Reliable Server Pooling*. IETF, Network Working Group, October 2003. draft-ietf-rserpool-arch-07.txt, work in progress.

[85] International Telecommunication Union. *Introduction to CCITT Signalling System No. 7*. ITU-T, March 1993. Recommendation Q.700.

Curriculum vitae

Name:	Esbold Unurkhaan
20.01.1975	born in Ulaanbaatar, Mongolia
9/1982 - 6/1992	Studied in 12th secondary school, Ulaanbaatar
9/1992 - 01/1997	Student of Computer Science and Management School of Mongolian Technical University
02/1997 - 03/2001	As a lecturer in Computer Science and Management School of Mongolian Technical University
04/2001 - 07/2004	PhD student at the University Duisburg-Essen Computer Networking Technology Group, Prof. Dr.-Ing. Erwin P. Rathgeb, Institute for Experimental Mathematics
since 08/2004	As a lecturer in Computer Science and Management School of Mongolian Technical University