# Optimization of Transport Security for Securing Peer-to-Peer Communication in Heterogeneous Networks

CHEN TA-WEI


KTH Microelectronics
and Information Technology

Master of Science Thesis
Stockholm, Sweden 2005

IMIT/LCN 2005-04


ROYAL INSTITUTE
OF TECHNOLOGY

# Optimization of Transport Security for Securing Peer-to-Peer Communication in Heterogeneous Networks

*Chen Ta-wei*
*iw03_tch@it.kth.se*

Master's Thesis in Telecommunication Systems
at the School of Information and Communication Technology,
Royal Institute of Technology. Commissioned by Ericsson AB

Examiner: Gerald Q. Maguire Jr.
Supervisor at Ericsson AB: Karl Normman

Last Updated: 03/21/2005

# *Abstract*

*This thesis concerns the security of tomorrow's peer-to-peer real-time communication in heterogeneous networks. Because of the additional delay caused by inband handshake and the poor compatibilities of some transport protocols, it was determined that existing security protocols such as transport layer security (TLS) and datagram transport layer security (DTLS) are not suitable in such a user scenario and a new security protocol should be designed. This new security protocol is called transport encapsulation security payload (TESP). TESP not only has the advantage of low initialization delay, but also fully supports transport protocols including TCP, UDP, stream control transmission protocol (SCTP), and datagram congestion control protocol (DCCP). Also a security analysis of TESP was carried out and no security flaws were found.*

# *Sammanfattning*

*Denna uppsats behandlar säkerheten för morgondagens "peer-to-peer" (P2P) realtidskommunikation i heterogena nät. På grund av den adderade fördröjning som orsakas av inbandssignalering och dålig kompabilitet hos många transportprotokoll, så kan man fastställa att existerande säkerhetsprotokoll, såsom "(Datagram) Transport Layer Security" (TLS och DTLS), inte är lämpade för denna typ av kommunikation och att ett nytt säkerhetsprotokoll bör tas fram. "Transport Encapsulation Security Payload" (TESP) är ett sådant protokoll. TESP har inte bara fördelar såsom låg uppstartsfördröjning, utan har också stöd för många transportprotokoll, t.ex. "Transport Control Protocol" (TCP), "User Datagram Protocol" (UDP), "Stream Control Transmission Protocol" (SCTP) och "Datagram Congestion Control Protocol" (DCCP). Även en säkerhetsanalys av TESP har gjorts, där inga säkerhetsproblem har kunnat påvisas.*

# Table of Contents

# 1. List of Abbreviations

| | |
|---|---|
| AES | Advanced Encryption Standard |
| AH | Authentication Header |
| CBC | Cipher Block Chain |
| CS | Crypto Session |
| CSB | Crypto Session Bundle |
| DCCP | Datagram Congestion Control Protocol |
| DTLS | Datagram TLS |
| ESP | Encapsulation Security Payload |
| IETF | Internet Engineering Task Force |
| IKE | Internet Key Exchange |
| IP | Internet Protocol |
| ISAKMP | Internet Security Association and Key Management Protocol |
| IV | Initialization Vector |
| MIKEY | Multimedia Internet Keying |
| NAT | Net Address Translation |
| OFB | Output Feedback |
| PR-SCTP | Partial Reliable SCTP |
| SCTP | Stream Control Transmission Protocol |
| SDP | Session Description Protocol |
| SIP | Session Initiation Protocol |
| SRTP | Secure Real-time Transport Protocol |
| SSL | Secure Socket Layer |
| TCP | Transport Control Protocol |
| TEK | Traffic Encryption Key |
| TESP | Transport Encapsulation Security Payload |
| TGK | TEK Generation Key |
| TLS | Transport Layer Security |
| UDP | User Datagram Protocol |
| WTLS | Wireless TLS |

# 2. Preface

This master's thesis was performed at the Communication Security Lab, Ericsson Research. The Communication Security Lab has been actively involved networking technology development within the Internet Engineering Task Force (IETF) in the area of real-time multimedia security in both the Multicast Security (MSEC) and the Multiparty Multimedia Session Control (MMUSIC) working groups.

# 3. Introduction

It is believed that the following user scenario will be quite common in the near future.

**The user scenario:**

A sales representative working for Ericsson, Alice, is on the train from Stockholm to Gothenburg for a conference. Alice wants to talk to one of her customers, Bob, who is sitting in his office with wired residential broadband Internet connection, about an important deal. Alice connects her PDA to the Internet via her 3G cell phone, launches a communication application and starts a text-based conversation with Bob, who has the same application on his laptop computer connected to the Internet, to exchange ideas about the content of the contract. Bob is not satisfied with one of the diagrams in the contract. Alice launches a whiteboard application and shares access to the whiteboard with Bob. They modify the diagram together on the whiteboard. Bob starts a videoconference session with Alice because he likes to speak face to face on important issues. Alice modifies the contract on her computer according to Bob's request and sends the new version of the contract to Bob by file transfer. During the communication, Alice wants to make sure that she is really talking to Bob, rather than someone impersonating Bob, and vice versa. Additionally, both Alice and Bob would like to keep the communication private because it contains vital business secrets. They also want to make sure the information exchanged (e.g. the contract sent to Bob from Alice) is not modified by a third party during transmission.

As described earlier, Alice is connected to a 3G wireless network and Bob is connected to the wired Internet. In such an environment, there might be several intermediate nodes (see section 4.4 for more details on these intermediate nodes) between them. These nodes provide services such as header compression, TCP performance enhancement, QoS provisioning, packet filtering, and Network Address Translation (NAT). Many of these intermediate nodes need information in the transport layer headers.

Peer-to-peer multimedia communication is already popular on the Internet. Examples of such applications are Microsoft's MSN Messenger, Yahoo! Messenger, AOL Instant Messenger, ICQ, Jabber, and Skype. Two phases are involved in peer-to-peer communication. In the first phase, peers send signaling protocol messages to locate each other and exchange information for the initiation of the subsequent sessions. In the second phase, peers communicate with each other over multiple media sessions, e.g., voice sessions, video sessions, and text sessions.

In these applications, it is quite common for a user to open multiple media sessions *at the same time*. These sessions may make use of multiple transport layer protocols: Voice and video streaming sessions as well as gaming signaling messages are typically carried by unreliable transport layer protocols such as UDP, and possibly the Partial Reliability Extension of the Stream Control Transmission Protocol (PR-SCTP) [5] or, in the future, the Datagram Congestion Control Protocol (DCCP) [7]. Whiteboard, text messaging, and file-exchange typically use reliable transport protocols such as TCP or, in the future possibly, the Stream Control Transmission Protocol (SCTP) [4].

The minimal security functions that should be provided during communication include peer-to-peer authentication, key management, end-to-end data confidentiality and integrity protection, and protection against possible attacks on the communication. The later include data replay attacks, data dropping/truncation attacks, and various types of denial of service (DoS)

attacks. Additionally, most of the sessions in peer-to-peer communication are used for real-time application. Therefore, the processing delay for these security functions should be minimized. One way to minimize the initialization delay is to piggyback the management protocol onto the session setup messages as part of the session description.

Network layer security protocols such as the IP Authentication Header (AH) [48] and the IP Encapsulating Security Payload (ESP) [50] provide a general solution for securing IP traffic. However, in the network environment of the scenario just described, because of the existence of intermediary-based transport services that need to access information in the transport headers of data packets or to manipulate the parameters in transport headers, there is a need for moving the security up to above the transport protocols.

Therefore, the goal of this thesis was to find a suitable solution to end-to-end data security on top of both reliable and unreliable transport protocols for peer-to-peer real-time communication. Taking into consideration the need to support newer transport protocols such as SCTP and DCCP to meet the requirements of the future.

The Transport Layer Security (TLS) protocol [22], a descendant of Secure Socket Layer (SSL) 3.0 [22] is a transport data security protocol that has evolved for a long time and has been carefully studied and analyzed [25]. It can be used for securing peer-to-peer real-time communication on top of reliable transport sessions (i.e. TCP connections).

Unfortunately, TLS was designed to run on top of reliable TCP connections in a *client-server* model. When it comes to peer-to-peer communication, there are several limitations that make TLS an unsuitable solution. First of all, TLS has its own authenticated key management mechanism. It is not possible to establish a TLS connection without performing the inband TLS handshake even if an external key management protocol has already exchanged all the security parameters needed beforehand (i.e. during session setup phase). In fact, it is not clear how TLS can work with external key management protocols such as MIKEY. Secondly, TLS does not support unreliable transport such as UDP sessions and DCCP sessions. Additionally, TLS does not fully support all the features of SCTP.

In this thesis, after careful study, it is determined that a more flexible and general security protocol should be designed, hence a new security protocol called Transport Encapsulation Security Protocol (TESP) is proposed. It has an **explicit** interface to external key management protocols. It supports both reliable and unreliable transport. Additionally, it supports all the features of newer transport protocols including SCTP and DCCP.

# 4. Background Knowledge

In this section, an overview of the following technologies is provided to the reader:

- Transport layer protocols: it is assumed that the reader is already familiar with the TCP/IP protocol stack including TCP, UDP, and IP. Hence, only a short introduction to the newer transport layer protocols such as SCTP and DCCP is given.
- Security protocols located in different layers of the protocol stack, including secure real-time transport protocol (SRTP) in the application layer, TLS, wireless transport layer security (WTLS) on top of the transport layer, and AH and ESP in the network layer.
- Key management protocols including MIKEY and ISAKMP.
- Intermediary-based transport services including several header compression protocols and network address translation (NAT). The later has been widely adopted because available IP addresses have become more and more scarce in the IPv4 address domain.

It is assumed that the reader is already familiar with the SIP protocol family (including SIP, SDP, and real-time data transfer protocols such as RTP). It is also assumed that, prior to reading this paper, the reader already has some basic knowledge of cryptography and is familiar with and understands most common terms, concepts, and algorithms in the area of cryptography and security. More information about these cryptographic terms, concepts, and algorithms can be found in books about cryptography, such as [8].

## 4.1. Transport Layer Protocols

### 4.1.1. The SCTP Protocol

SCTP [4] was originally designed for carrying telephony signaling (e.g. Signaling System 7 (SS7) [55]) in IP networks. The transport of SS7 signaling messages has strict requirements for reliability and timely delivery, and the available transport protocols before SCTP was developed (UDP and TCP) cannot meet these requirements. Similarly to TCP, SCTP provides a connection-oriented, reliable transport service with flow control and congestion control. However, in addition to what TCP provides, it also supports the following functions:

- Message oriented data delivery, which is more suitable for the transport of signaling messages, instead of byte-stream oriented delivery.
- Multi-streaming, which allows application data to be multiplexed onto one association (the term used by SCTP to represent a connection). This feature, when used to carry signaling messages, can separate signaling messages for different calls into different message flows thus avoiding head-of-line blocking. This makes the transport timelier.
- Multi-homing (i.e. multiple IP addresses), which allows an established SCTP association to be maintained when the either one of IP addresses used by the communicating endpoints changes. This feature enables endpoints have backup network interfaces and also decreases the time required for connection recovery from link failures. This later feature is useful for applications that require high availability.
- Unordered delivery, which allows a data message marked as unordered to bypass the ordering mechanisms and be immediately delivered to the upper layer in the receiving endpoint, while all the other normal data messages are delivered in order.
- A cookie mechanism to prevent denial-of-service attacks.
- Feature extensibility, which allows protocol developers to add new features to SCTP.

Feature extension is done by defining new chunk types. When two endpoints start an SCTP association, the variable-length parameters of the INIT messages allow them to negotiate the use of optional features.

### 4.1.2. SCTP Partial Reliability Extension

The SCTP Partial Reliability Extension (PR-SCTP) [5] allows an SCTP endpoint to signal the other endpoint of the SCTP association to explicitly move the cumulative acknowledgement pointer forward to abandon one or more messages. This extension provides a partially reliable data transport to the upper layer application. The use of this extension is negotiated by communicating endpoints during the SCTP association establishment and only when both endpoints agree with the use of this extension, the PR-SCTP data transport can be used. If both endpoints of an SCTP association agree to the use of this extension, then all the message streams within the association are partial reliable.

From the perspective of the upper layer that uses the partially reliable data transport, although it is unreliable and some messages might be lost, all the messages except the unordered delivery messages are still delivered to the upper layer in order at the receiving endpoint. Therefore, excluding unordered delivery messages, the data transport provided by PR-SCTP is *unreliable* but *ordered*. This is different from the *unreliable*, *unordered* transport provided by UDP or DCCP. The PR-SCTP specification allows definition of different services provided by PR-SCTP data transport to the upper layer. One service that is already defined is the "timed reliability service". The upper layer protocol that uses this service can specify the 'lifetime' of each message it sends. If by the time the transmission of a certain message in the SCTP protocol stack takes place the message has already 'expired', then instead of sending the message down to the network layer, the SCTP protocol stack sends a 'Forward-TSN' control chunk to the other endpoint as a signal to abandon this expired message.

### 4.1.3. SCTP Dynamic Address Reconfiguration Extension

The SCTP dynamic address reconfiguration extension [6] is still a work-in-progress. This SCTP extension defines two new chunk types (ASCONF, and ASCONF ACK) for mobility management signaling. An endpoint can use these newly defined chunk types to notify its peer about the change of its address list dynamically, whereas in the SCTP specification, although multi-homing is supported, endpoints have to send a list of **static** IP addresses that potentially will be used during the initialization of an SCTP association. The SCTP dynamic address reconfiguration extension makes SCTP a possible solution for handling mobility at the transport layer.

### 4.1.4. The DCCP Protocol

DCCP [7] is still a work-in-progress. However, since it is already awaiting the last call for final comments, it is very likely that DCCP will become a proposed standard pretty soon. Like TCP, DCCP is a connection-oriented transport protocol and provides reliable connection establishment and teardown mechanisms. However, unlike TCP, which provides reliable transport with possibly long delay caused by retransmission of data, DCCP provides *unreliable* but *low delay* transport for applications that care about timely delivery (i.e. latency avoidance) and can tolerate packet loss and *unordered* delivery. Today, these applications use UDP to avoid latency. However, they have to implement congestion control on their own. However, DCCP, however provides built-in congestion control.

In short DCCP is a transport protocol that establishes bidirectional connections to perform congestion control over the transmission of unreliable datagrams. DCCP has the following features:

- Reliable connection establishment and teardown. The establishment of a DCCP connection makes it possible to negotiate options including negotiation of a suitable congestion control mechanism and exchange of state parameters to perform congestion

control. It also helps firewall traversal of DCCP sessions.

- Reliable feature negotiation (such as mobility, acknowledgement, and congestion control)
- Acknowledgement of received packets, which provides information about packet loss and network congestion.
- Congestion control.
- Path MTU discovery to avoid IP fragmentation which is considered harmful [1].
- Firewall traversal thanks to connection setup and teardown
- Multihoming and mobility management: DCCP supports mobility. Mobility management is done by notifying the peer about the change in IP address by sending a DCCP-Move packet containing the new IP address. DCCP supports mobility of only one endpoint at a time.

## 4.2. Key Management Protocols

The purpose of key management protocols is to perform *mutual authentication* between the communicating parties and to negotiate *keys and security policies* needed by the security protocol. Typical security policies include protocol-specific parameters such as encryption algorithm, authentication algorithm, and key length.

### 4.2.1. The ISAKMP Protocol

The Internet Security Association and Key Management Protocol (ISAKMP) [52] is not a complete protocol itself but rather a framework specifying which parameters could be exchanged to create a key management protocol. Since ISAKMP is designed to be an independent framework protocol, another document called a Domain of Interpretation (DOI) must be written for each key management protocol that in the ISAKMP framework defines the format of the actual payloads of ISAKMP message for a specific key management protocol. The key management protocol for IP security, the Internet Key Exchange (IKE) protocol [53], was developed in the context of ISAKMP. A DOI for IKE was also written [51].

### 4.2.2. The MIKEY Protocol

The Multimedia Internet KEYing (MIKEY) protocol [18] performs key management for real-time multimedia peer-to-peer or group communication applications on thin clients. The requirements of real-time multimedia communication applications on thin clients include low delay, low computational cost, and small footprint. MIKEY minimizes processing delay by piggybacking its handshake messages on signaling messages in session setup protocols such as SIP. MIKEY supports shared secret authentication and key establishment that is not computationally demanding, hence it is more suitable for thin clients than other key management protocols that only support public key authentication. MIKEY achieves a small footprint by reusing cryptographic modules in different parts of the protocol.

MIKEY has been used as a key management protocol for setting up the security for multiple Real Time Protocol (RTP) [13] media sessions protected by the Secure Real-time Transport Protocol (SRTP) [21]. MIKEY can be further extended to provide key management for sessions protected by various payload security protocols other than SRTP.

MIKEY defines two types of keys to be generated and exchanged: a traffic-encryption key (TEK) used to secure the data in a specific multimedia session and a TEK Generation Key (TGK) exchanged via the MIKEY protocol used as an entropy source to generate TEKs for *multiple* media sessions of a specific multimedia session. Once a TGK is exchanged, it can be used to generate several TEKs for related media sessions to eliminate the need for additional

key exchanges.

MIKEY calls the trust relationship and security parameters derived from the TGK between two parties a Crypto Session Bundle (CSB) and gives a group ID to it (i.e. the CSB ID). Within one CSB, multiple media sessions can be protected by the TEKs derived from a single TGK. MIKEY calls an individual data stream (e.g., a RTP stream) a Crypto Session (CS) and gives it a unique ID (i.e. CS ID) to it. A CS is similar to the concept of a data Security Association (SA) defined in the MSEC Group Key Management Architecture [19]. The relationships between CSB, CS, TGK, and TEK are illustrated in Figure 1.

In a MIKEY handshake, the two parties (the initiator and the responder) authenticate each other and agree on one key (TGK). The authentication and the key exchange are based on a shared secret method, public keys, or Diffie-Hellman.

After the TGK is exchanged (i.e. a CSB is established), the TEKs are derived via a key derivation function with the TGK, the CS ID, and the CSB ID together with a random number exchanged during MIKEY handshaking as input parameters. A TEK is then used as a "master key" by security protocols such as SRTP (see Section 4.3.6) to further generate encryption keys, an authentication key, and a salting key for actually securing a CS. In a bi-directional voice stream, each direction of the RTP stream is typically deemed a separate CS and is therefore given a unique CS ID and a specific TEK.

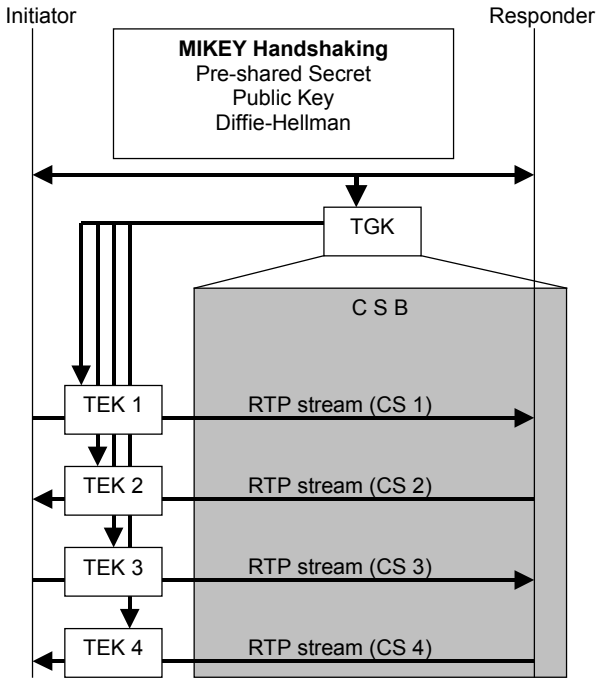



Figure 1. Illustration of the MIKEY protocol

## 4.3. Security Protocols

Security protocols are responsible for the confidentiality and integrity of packet payloads traveling through the network. Security protocols can reside in different layers of the protocol stack. For instance, the Authentication Header (AH) [49] and Encapsulation Security Payload (ESP) [50] are security protocols located at the network layer; TLS [23] and WTLS [33] are security protocols running on top of the transport layer; and SRTP is an application-layer security protocol.

### 4.3.1. AH and ESP

AH and ESP are the security protocols used by the IP security architecture. Both AH and ESP operate at the network layer. AH provides data *integrity covering the entire* IP packet. On the other hand, ESP provides data *confidentiality* and *integrity* covering the IP packet *payload*.

### 4.3.2. TLS version 1.0 Protocol

TLS [22] is a security protocol for securing the data transmitted in a "reliable" transport-layer (e.g., TCP, SCTP, etc.) connection between a server and its client. In addition to the provision of payload security, TLS has its own inband key management protocol so that no external key management protocol is needed. The process of setting up a TLS connection is described as follows:

1. The client opens a TCP connection to the server. All the handshake messages and the protected data records afterward will be sent via this TCP connection.
2. The two endpoints (client and server) exchange a full TLS handshake to exchange information for mutual authentication, negotiating the cipher suite, deriving the cryptographic parameters needed for establishing a TLS connection, and acknowledging the other party starting cryptographic transformation (i.e. encryption and integrity protection). A TLS connection is then established.
3. The endpoints now can send data (encapsulated in data records) securely over this TLS connection.
4. When either one of the endpoints wishes to terminate a TLS connection, it sends an alert message to the other endpoint to acknowledge the unidirectional closure of the TLS connection. All the data that arrives later than this alert message will be ignored. Upon receiving this closure alert message, the receiving end sends a closure alert message back to the other endpoint to acknowledge the closure of the other direction of the TLS connection.

Both data and handshake messages are sent over the same TCP/SCTP connection/stream. However, before they are delivered to the TCP layer, they are fragmented into blocks with limited size, cryptographically transformed, and encapsulated into data records.

TLS was originally designed for securing HTTP traffic, which usually opens many TCP connections in parallel between the web browser (the client) and the web server (the server). Session resumption is an important feature provided by TLS to reuse the master key derived by a previously established TLS connection to establish a new TLS connection. It saves computing power and time to use session resumption because in a resumed handshake, only the session ID and two random numbers are exchanged and no public key operation is needed. Session resumption works as follows:

1. The client has already established a TLS connection to the server.
2. The client wants to open a new TLS connection to the server.
3. For this new connection, the client sends the session ID of the previously established TLS session in a hello message.
4. The server checks if it remembers the received session ID (if it already exists). If it does, the server does not need to send the certificate back. It simply sends back the same session ID in another hello message to the client. Both parties resume the previous session in this way and reuse the master key to generate new encryption keys, authentication keys, and initialization vectors (IV) in both directions.
5. If the server does not remember the received session ID or it decides to initiate a new session anyway, it sends back a different session ID in the hello message to the client. A full handshake will be performed and hence a new session will be created instead of

reusing an old one.

### 4.3.3. TLS version 1.1 Draft

TLS version 1.1, still a work-in-progress [24], is a major change from TLS version 1.0; TLS version 1.1 adds an explicit cipher block chain (CBC) state to the TLS records. This means that, when the block ciphers in CBC mode are used for encryption, TLS version 1.1 puts an explicit IV in the record header in plaintext to prevent the possible attacks described in [25] instead of using the last block of the previous encrypted record as the IV. Since TLS version 1.1 is still a draft, whenever TLS is mentioned in this thesis without specifying the version, it is the TLS version 1.0 that is referred to.

### 4.3.4. SCTP support of TLS

TLS over SCTP is defined in RFC 3436 [30]. In this specification, a TLS connection should be setup within a bi-directional stream rather than an association. The reason behind this is that only messages within the same stream are guaranteed to arrive in the right sequence and the CBC mode of block cipher used by TLS will only work if the records arrive in the correct sequence. The following requirements are specified by RFC 3436 for SCTP to be able to carry a secure TLS connection:

- TLS must run over a bi-directional stream.
- SCTP must provide fragmentation of messages to be able to transmit all TLS records as SCTP user messages, since the maximum length of a TLS ciphertext record is 18437 bytes.
- The "unordered delivery" feature of SCTP must **not** be used on streams with TLS protection running on top.
- The PR-SCTP (Partial Reliable SCTP) extension of SCTP cannot be used.

TLS must support TLS_RSA_WITH_AES_128_CBC_SHA in order to be able to support SCTP.

### 4.3.5. WTLS Protocol

WTLS [33] is the security protocol of the wireless application protocol (WAP) version 1 protocol stack [35]. The design of WTLS basically follows TLS, but with some minor changes that adapts WTLS to the characteristics of a wireless environment and to make WTLS fit into the WAP version 1 protocol stack. As illustrated in Figure 2, WTLS sits between the Wireless Transaction Protocol (WTP) and the Wireless Datagram Protocol (WDP), the transport layer of the WAP version 1 protocol stack. WTLS is able to support an unreliable transport protocol (i.e. WDP). Unfortunately the adaptations from TLS to WTLS also weakened its security. Amny flaws have been discovered in WTLS [34]. Possible attacks on WTLS include a chosen plaintext data recovery attack, a datagram truncation attack, a message forgery attack, and a key-search shortcut for some exportable keys.

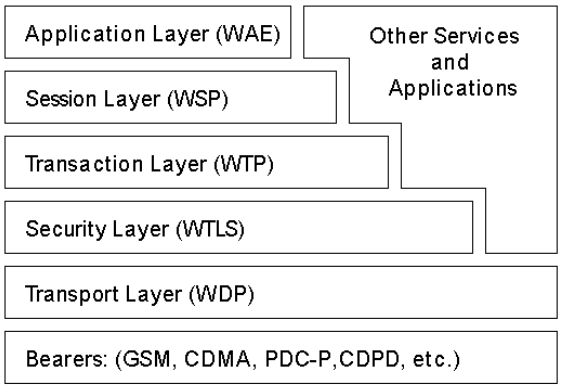



Figure 2. WAP 1 Protocol Stack

### 4.3.6. The SRTP Protocol

SRTP [21] is an application-layer security protocol for securing the RTP payload (i.e., the RTP payload requires confidentiality and integrity protection, whereas the RTP header is only integrity protected, so as not to not obstruct header compression). For each RTP session (note that in the case of a bi-directional RTP stream, there are **two** sessions), SRTP maintains a cryptographic context. One node might have several cryptographic contexts for each ongoing RTP session. There are several pieces of information in the context, including the encryption key, the message authentication key, the rollover counter, the replay list, and the highest received sequence number.

SRTP is designed to cooperate with authentication and key management protocols, such as MIKEY and ISAKMP. The key management of SRTP can also be done by using Session Description Protocol (SDP) cryptographic attributes [15],[16], but only in restricted cases in which SIP messages (including piggybacked SDP media description) are secured end-to-end by Secure/Multipurpose Internet Mail Extensions (S/MIME) [17] or other security protocols such as TLS or ESP.

SRTP takes the master key and master salting key from the key exchange protocol it cooperates with and puts them into the cryptographic context. The master key and the master salting key, together with the SRTP packet index are input to the key generator to generate session keys (encryption key, authentication key, and salting key) and the session keys are used to encrypt and authenticate each RTP packet.

When SRTP receives a SRTP packet or protects an RTP packet, it searches for the cryptographic context for the RTP session this packet belongs to. If the context is found, it uses the information in the context to encrypt/decrypt as well as to authenticate the packet. When SRTP receives a packet, it also checks the replay list to protect a receiver from replay attacks.

The “lifetime” of the session keys (the number of packets secured by the same set of session keys) is determined by the key derivation rate (which is part of the cryptographic context of the session). The derivation of a new session key from the same master key is called “refreshing” of the session key in SRTP.

The rekeying of the master key is synchronized by the master key index (MKI) field appended to SRTP packets or the <from, to> information stored in the cryptographic context specifying the starting packet index and the ending packet index of the RTP packets to be protected by a specific master secret. The renewal of the master key is called “re-keying” in SRTP. The index number of a received SRTP packet corresponds to the roll over counter (ROC), part of the cryptographic context, and the “sequence number” field in the SRTP packet header itself.

## 4.4. Intermediary-based Transport Services

In today's IP networks, there are quite a few intermediate nodes serving various purposes. The services provided by these nodes can be categorized into the following two types:

- Performance optimization: the end-to-end performance of TCP and other transport layer protocols can be degraded by the link characteristics of a certain link within the data path [38]. This degradation can be mitigated by placing Performance Enhancing Proxies (PEP) [57] of various types at the ends of a link or a subnetwork where transmission performance suffers due to the characteristics of the link or the links in the subnetwork. Several mechanisms [57] can be used to improve performance of a link or a subnetwork, including TCP ACK handling, tunneling, header and payload compression, handling periods of link disconnection with TCP, priority-based multiplexing, and protocol booster mechanisms. Most of the PEPs need to access the transport headers (some of them even need to access information contained in the application data). Therefore if IP packets are protected by network layer security protocols such as ESP, PEPs along the path will fail to apply its performance enhancement.
- Intermediary-based transport services include the services provided by the middle boxes defined in [58], e.g., firewall (policy-based filtering), NAT, Proxy, and application Level Gateway (ALG). These middle boxes, like PEPs, also need to access or even manipulate the transport headers (some of them even need to access information contained in the application data). Again, in an environment where these middle boxes are operating, network layer security protocols are not feasible.

### 4.4.1. Header Compression Protocols

It is a trend for telecomm networks (including the wireless networks) and data networks to converge into a single all-IP network. However, sending IP packets over a link (e.g. a wireless link) with limited bandwidth, high frame-error-rate and high latency [38], [39] could be inefficient because of the overhead of network layer, transport layer, and application layer headers. The inefficiency becomes extremely obvious when the link is carrying audio traffic with packet payloads (typically 20-60 bytes), which is even shorter than the packet headers (over 40 bytes for IPv6+UDP+RTP).

Under these circumstances, header compression on such a link can effectively save bandwidth (perhaps improving capacity), reduce packet loss, and lower transmission latency [40], [41]. Several header compression protocols were developed by the IETF. Examples are the Compressed Transport Control Protocol (CTCP) [42], the IP Header Compression (IPHC) protocol [43], the Compressed Real-Time Protocol (CRTP) [44], and the RObust Header Compression (ROHC) protocol [45]. Among them, ROHC [45] gives by far the most efficient header compression. It works well in the wireless environment and is resistant to bit errors. Additionally, it can compress several profiles (e.g. IP/UDP/RTP, IP/UDP, IP/ESP) of header combinations down to one byte at minimum (two bytes on average).

Header compression is normally done in the middle of a path between endpoints (generally between a pair of nodes connecting a wireless link). Therefore, when a security protocol is used to protect end-to-end, the higher the layer the security protocol lies in, the more efficient ROHC could be.

In the case of securing RTP traffic, for example, the ROHC RTP profile (the profile achieves the highest compression ratio as it compresses the IP header, the UDP header, and the RTP header) can be applied only if an application layer security protocol (SRTP) is used (Figure 3). On the other hand, if the IPsec Encapsulating Security Payload (ESP) [50] is used to protect RTP traffic, only the ROHC ESP profile can be applied. In this case, the

compression ratio is not as high as for the RTP profile because only the IP header and the ESP header can be compressed (Figure 4). Of course this decreased efficiency does provide greater privacy of the communication. Besides the three profiles mentioned above, another profile for the header compression of TCP connections (ROHC-TCP) is also proposed as an Internet draft [46]. ROHC has been adopted by the 3GPP standard as part of their WCDMA system.

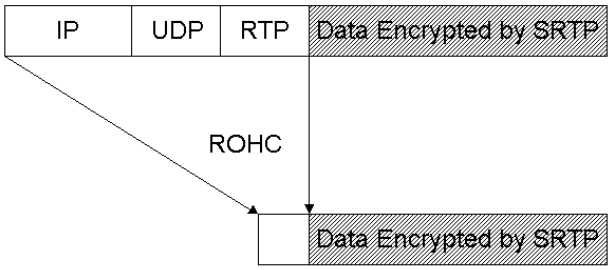



Figure 3. The ROHC RTP profile

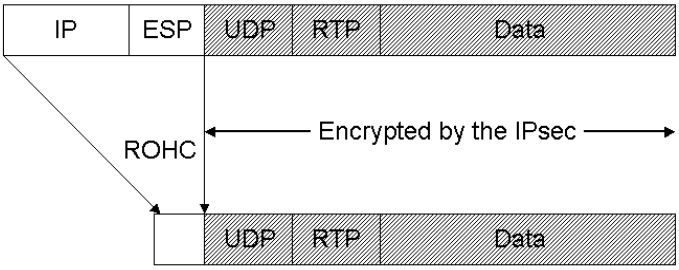



Figure 4. The ROHC ESP profile

### 4.4.2. Network Address Translation

Network Address Translation (NAT) [56] devices connect an isolated network with private IP addresses to the Internet with public addresses. There are several flavors of NAT. In this thesis Network Address Port Translation (NAPT) is of main concern. NAPT is also the most commonly used NAT mechanism in NAT devices currently on the market. The advantage of using NAPT is that multiple hosts in this isolated network can transparently communicate with hosts in external networks via a single public address. This is done by the translation of transport identifiers in transport layer headers such as port numbers in TCP and UDP, and the query identifier in ICMP. NAPT devices use information in transport layer headers of IP packets to perform transport identifier translation.

# 5. Related Work

In this section, recent works that are related to the development of the security solution for peer-to-peer communication on top of the transport layer are described.

The new means of media transport described in section 5.1 bring TCP connections and SCTP associations into the context of peer-to-peer communication when SIP is used as the session setup protocol. This implies that the security solution for peer-to-peer communication on top of the transport layer should be able to protect not only voice sessions carried by UDP/DCCP but also TCP connections and SCTP associations.

DTLS, described in section 5.2, can be used to complement TLS to support unreliable transport protocols in the security solution for peer-to-peer communication.

The TLS PSK ciphersuites, which are described in section 5.3, though not originally designed for this purpose, could potentially be used as an *implicit* interface for TLS and DTLS to external key management protocols

## 5.1. Description of a Connection-Oriented Session in SDP

There is an Internet draft [60] that describes the expression of media transport over connection-oriented protocols such as TCP using SDP. It defines a new value of protocol identifier, TCP, in the media field. It also recommends the usage of "TCP/" in the protocol identifier for protocols between TCP and the media, such as "TCP/TLS" for TLS connections. This Internet draft also defines an attribute called "setup" to indicate which of the communicating parties should initiate the TCP connection described. Another new attribute defined is the "connection" attribute indicating if the SDP description is a request for a new connection or an exchange of attributes of the current connection.

An example of such a TCP connection is as follows:

```
m=image 5000 TCP t38
c=IN IP4 192.0.8.2
a=setup:active
a=connection:existing
```

There are four sub-fields in the media description line (the "m" line). The first sub-field (image) is the media type. The second sub-field (5000) is the transport port. The third sub-field (TCP) is the transport protocol. The last sub-field (t38) is the media format. "image/t38" is a MIME sub-type encoding definition intended to be used as an SDP media descriptor [62].

There is another expired Internet draft that describes the specification of media transport over SCTP [61].

## 5.2. DTLS

Datagram Transport Layer Security (DTLS) is still a work-in-progress [37]. However, since it is already awaiting the last call for final comments, it is very likely that DTLS will soon become a proposed standard. DTLS is a security protocol on top of the transport layer designed as a complementary protocol to TLS to provide data confidentiality and integrity for datagram transport layer protocols (e.g. UDP, DCCP). It is based on TLS, but has a number of differences from TLS version 1.1. DTLS supports datagram transport based upon the following changes from TLS:

- To avoid IP fragmentation, which is considered harmful to the transmission efficiency [1], DTLS specifies that each record must fit within a single IP packet. DTLS also

supports the discovery of Path MTU (PMTU). PMTU is the minimum of the MTUs of all links along the path between the communicating endpoints.

- Since the size of a record is limited (by the MTU or the PMTU), fragmented handshake messages are unavoidable. To deal with the possibility of records carrying fragments of the same message arriving in the wrong order, two new fields are added to the handshake message header: "fragment_offset" and "fragment_length". The way these fields are used to reassemble a handshake message is similar to the mechanism used in IP fragmentation/reassembly. To deal with the possibility of messages arriving in the wrong order, the "message_seq" field is added to the handshake message header. With the help of the "message_seq" field, the receiver of a sequence of messages (e.g. ServerHello, Certificate, and ServerHelloDone) can check if the messages arrive in the right order and enqueue the messages that arrive earlier than expected; while waiting for messages that are late.
- DTLS uses *explicit* sequence numbering instead of TLS's *implicit* sequence numbering in the record layer so that DTLS can tolerate lost, duplicated, or out-of-order records. DTLS also supports the detection of duplicate records (i.e. replay detection).
- The RC4 stream cipher cannot be used in DTLS, because it does not allow random access to the key stream.
- In DTLS, both the server and the client must retransmit messages if the expected response messages are not received from the other end.
- For block ciphers running in CBC mode, consecutive records cannot be chained together because of the possibility of packet loss in a datagram service. This issue can be solved by adding explicit CBC state to the TLS records. The support of explicit CBC state has already been specified in TLS version 1.1.

DTLS also deals with possible attacks during the handshake period. Since the handshake messages of DTLS are carried by connectionless datagram transport layer protocols such as UDP, it is possible to do a resource consumption attack on the server by sending large amounts of "ClientHello" messages. It is also possible to make another similar attack on a victim by sending large amount of "ClientHello" messages with the source address all pointing to the victim. In this way, the server will "attack" the victim with a large amount of the very long Certificate messages and consume the victim's network bandwidth. DTLS prevents the attacks mentioned above by adding a "cookie" exchange to the protocol. The authentication process with the "cookie" exchange is:

1. The client sends the ClientHello message to the server with an empty cookie field.
2. The server replies to the client with a HelloVerifyRequest containing a stateless cookie.
3. The client sends the ClientHello message to the server with the cookie from the server.
4. The server verifies the cookie and responds with the ServerHello and following messages.

## 5.3. PSK Ciphersuites for TLS

Pre-Shared Key (PSK) ciphersuites for TLS [27] are new ciphersuites with PSK as the authentication/key exchange mechanism. They are added to the ciphersuites for use in a constrained environment (e.g., thin clients) or an environment without PKI. To be able to do PSK authentication, both the client and the server may have several PSKs with several different parties. Thus, the handshake messages for a PSK authentication must utilize the right PSK for authentication between a specific pair of parties. PSK authentication in TLS works as follows:

1. The client sends a "ClientHello" message to the server with one or more PSK

ciphersuites in the list of ciphersuites.

2 The server selects one of the PSK ciphersuites and puts it in the “ServerHello” message and sends this message to the client together with the “ServerKeyExchange” message with “psk_identity_hint” which helps the client to decide which identity to use (The client might have several PSKs for different servers. An example of the “psk_identity-_hint” could be the server’s domain name).

3 The client sends a “ClientKeyExchange” message with “psk_identity” which indicates the key for this specific identity. An example of the “psk_identity” could be the username of the client.

Now, both the client and the server have enough information to generate the premaster secret.

# 6. Problem Statement

As described in the introductory section, this thesis concerns the security of peer-to-peer real-time communication in a heterogeneous network in the scenario described in the introductory section. This environment has several characteristics:

- A session setup protocol such as SIP is used for peer-to-peer session setup.
- There may be several intermediate boxes are between the peers. These intermediate boxes need information from the transport layer header in the data packets.
- Users are sensitive to initialization latency.

Security functions including end-to-end data confidentiality and integrity, replay attack protection, and denial-of-service protection against various attacks should be provided. Additionally, the security protocols for securing peer-to-peer communication in such an environment should meet the following requirements:

- Initialization delay caused by key management should be reduced. In other words, it is best if key management handshake messages are piggybacked on session setup messages. This could be achieved by using lightweight key management protocols such as MIKEY that can, not only be piggybacked on session setup protocols, but can also perform one key management handshake for multiple crypto sessions. This implies that the payload security protocols must be able to work with external key management protocols.
- Payload security should be done above transport layer.
- Both reliable and unreliable transport layer protocols should be supported.
- Newer transport protocols such as SCTP and DCCP should be supported to meet the requirements of the applications developed in the future.

Among the existing security protocols running on top of the transport layer, TLS is the most commonly used. TLS provides the security functions that reliable peer-to-peer communication needs. Additionally, TLS and its ascendants (SSLv1, SSLv2, and SSLv3) have been developed for a long time and have been carefully studied by security experts. TLS has been adopted as the de facto security protocol for web browsers and web servers. Although TLS was originally designed for client-server TCP connections, it is possible to use TLS to secure peer-to-peer communication. However, there are some problems with TLS when it is used as a solution for securing peer-to-peer real-time communication in general:

- TLS has its own inband key management mechanism and has no explicit interface for using external authenticated key management protocols such as MIKEY.
- In the specification of TLS, a full handshake or a resumed handshake must be performed before a secure TLS connection is setup. The function of a full TLS handshake is to perform key management. The function of a resumed handshake is to reuse the master secret of an existing TLS connection. In peer-to-peer communication, where key management can be performed during session setup *before* a TCP connection is made, it does not make sense to perform inband key management when the key management can be done during session setup. As inband key management only adds additional initialization latency. According to [54], the time needed for a TLS full handshake with mutual authentication is around 118.6 msec in a 100Mbps Ethernet environment with Pentium 4, 2.4GHz CPUs on the computers running TLS. In lower bandwidth, higher delay wireless environment, the latency for a TLS handshake will be worse (see the measurement results in section 10.3.3). Unnecessary initialization latency may degrade the user's experience.
- TLS does not fully support all the features and extensions of SCTP. TLS, for instance,

does not support the unordered delivery feature and PR-SCTP associations. Further more, TLS does not support unidirectional data transport, which is supported by SCTP. SCTP streams must be paired into bidirectional streams to run TLS.

- TLS does not support unreliable or unordered transport. Some changes must be made to TLS to extend its capability to support best-effort datagram transport. This change can be presented as protocol variant of TLS that can replace TLS and supports both reliable and unreliable transport layer protocols or as a complementary protocol to TLS for best-effort datagram support so that TLS coexists with this complementary protocol, one serving reliable transport layer protocols and one serving unreliable datagram transport layer protocols.

When WAP was designed, WTLS was introduced to replace TLS in order to support both reliable and unreliable transport protocols. However, WTLS, as mentioned in section 4.3.5, has been found to have some critical weaknesses. Of course it is possible to fine-tune the protocol to avoid some of the weaknesses discovered. However, the scale of the change might be similar to the scale of the change to enable TLS to support datagram service. Additionally, there is already work in progress within IETF to design a variant of TLS (i.e. DTLS, which was described in section 5.1) that is able to support unreliable datagram transport. Therefore, WTLS is not considered further.

Another alternative to support both reliable and unreliable transport is to use TLS and DTLS together. However, this is not seen as a feasible solution. First, there is no explicit interface for TLS and DTLS to work with external key management protocols. The TLS PSK ciphersuites, which are described in section 5.3, could probably be used as an *implicit* interface of TLS and DTLS to external key management protocols. However, with TLS PSK ciphesuites, inband TLS/DTLS handshake is still unavoidable and the DTLS handshake takes one round trip *more* than the TLS handshake because of the cookie exchange in the beginning of the handshake. Additionally, when PSK ciphersuites are used as the interface to key management protocols, part of the key management is still performed by the inbound TLS handshake instead of external key management protocols; i.e., the communicating endpoints still negotiate the cipher and the message authentication function using the hello messages of the inband TLS/DTLS handshake. Only the shared secret is exchanged by the external key management protocol. This makes the interface between key management protocols and TLS ambiguous. It is non-trivial how to resolve these issues while still maintaining compatibility with the original TLS/DTLS specification.

# 7. Goals

The goal of this thesis was to find a suitable solution for securing peer-to-peer communication in heterogeneous networks. Since existing security protocols on top of transport layer such as TLS and DTLS do not suffice, as described in the previous section, there was a need to develop a new security protocol running on top of the transport layer for securing peer-to-peer real-time communication. This new security protocol should be developed based on the design of TLS and DTLS so that the design will not be built totally from scratch and components of TLS and DTLS that have been carefully analyzed can be reused. Building upon TLS and DTLS also makes the reuse of source code possible, reducing the implementation effort and providing better security. This new security protocol should provide sufficient generality and flexibility to support both reliable and unreliable transport protocols and to support all the features provided by newer transport protocols including SCTP and DCCP. This new security protocol should use external key management protocols to do key management instead of inband handshaking as TLS does. An explicit interface to external management protocols should be defined and described in detail. Additionally, a security analysis must be done to discover weaknesses and flaws of this proposed solution.

Note that there is no need for this new security protocol to be optimized for RTP traffic. As it is assumed that this kind traffic is already well secured by SRTP.

# 8. Approach

The approach used can be divided into several steps and is briefly described as follows:

- Analyze the characteristics of the transport protocols that are concerned (i.e. TCP, UDP, SCTP, DCCP). The characteristics of a transport layer protocol include the form of application data it accepts, the transport services it provides, the level of reliability and security of the services it provides, and other functions of a transport layer protocol (if there is any).
- Develop a new security protocol based on the design of TLS that is able to support all the features of TCP, UDP, SCTP, and DCCP using the information provided by the analysis of these transport protocols from the previous step.
- Although key management protocols are not the focus of this thesis, the feasibility of the solution is studied by examining if existing key management protocols such as MIKEY can properly exchange the security parameters for the proposed new security protocol.
- Perform a security analysis of the new security protocol.

# 9. Analysis of the Transport Protocols

Before starting to find the solution for securing peer-to-peer communication running on top of the transport layer, one should analyze the characteristics of different types of services that are provided by the various underlying transport layer protocols. In this thesis, the transport layer protocols that are of interest are TCP, UDP, SCTP, and DCCP. UDP and TCP are the de facto transport layer protocols that exist in every implementation of the TCP/IP protocol stack. SCTP and DCCP are two newer transport layer protocols and have already been described in sections 4.1.1 and 4.1.4. The characteristics of transport layer protocol can be compared in the provision of the following services, as shown in

Table 1:

- Traffic multiplexing: To enable multiple applications to share services provided by one transport layer protocol. This is usually done by adding port number information to the transport layer protocol header and assigning different port numbers to traffic from different applications.
- Connection setup and teardown: The exchange of signaling packets setups a connection. A connection can be seen as a pair of states maintained by the endpoints with parameters needed for flow control, congestion control, data sequencing, or other services. These parameters are synchronized due to the connection setup signaling packets. In today's Internet architecture, this explicit connection setup and teardown helps intermediate nodes such as NAT boxes and firewall to handle the data session correctly (as a side effect).
- Duplex modes: the duplex modes including:
    - Bi-directional, full duplex: The application at both ends of the communication path over a bi-directional full duplex transport can send and receive data at the same time.
    - Bi-directional, half duplex: The application at both ends of the communication path over a bidirectional half duplex transport can only send data or receive data at a given time. This mode is not seen in any IP network's transport layer protocols, but may occur at lower layers.
    - Unidirectional: The application can only send or receive data.
- Flow control: the mechanism that is used to avoid overflowing the receiver's buffer.
- Congestion control: adapting the sender's transmission data rate to avoid network congestion.
- Data boundary: the data boundary is the minimum unit data delivered or received by the upper layer of a transport layer protocol. Possible data boundaries include:
    - Byte: The data is seen as a stream of independent bytes. TCP is a byte-oriented transport layer protocol.
    - Message-oriented: Each data segment that is delivered by the upper layer is seen as a message. A message could be further fragmented by the transport layer to form multiple IP packets if needed. Each time the upper layer of the receiving endpoint receives some data, it receives a complete message from the sending application. SCTP treats data as a sequence of messages.
    - Datagram: Each time a segment of data is sent by the application, it is encapsulated into an IP packet (if IP fragmentation is not needed) and delivered to the upper layer of the receiving endpoint as a complete datagram. UDP and DCCP are transport layer protocols dealing with datagrams.
- Data delivery: the data delivery services provided by a transport layer protocol can be categorized into the following types:
    - Reliable, ordered delivery: provides guaranteed, ordered delivery of data.
    - Reliable, unordered delivery: provides guaranteed, unordered delivery of data.
    - Unreliable, ordered delivery: provides best-effort data transmission. Data may be lost during transmission, but data always arrives at the receiving end in the same order as it is sent at the sending end.
    - Unreliable, unordered delivery: provides best-effort data transmission. Data may be lost during transmission. Data may arrive at the receiving end in a different order from when it is sent.

Among all the services listed in Table 1, the provision of the services that are *shaded* in the table of a certain transport protocol (i.e. data boundary, service reliability, ordered delivery, unordered delivery, data fragmentation, PMTU discovery, and checksum) are the factors that would affect the design of the security protocol on top of the transport protocol:

- Data boundary: the header of a data record should provide *explicit* information of data length if the data boundary of a transport protocol is based on bytes. Otherwise the length information of a data record is *implicit*.
- Multi-streaming: if a transport protocol supports multi-streaming, the security protocol either utilizes one master secret per stream or has to derive multiple master secrets for from one master secret and stream IDs.
- Service reliability and delivery order: the sequence number for each data record is implicit if the service provided by the underlying transport protocol is reliable and ordered. Otherwise each data record must explicitly carry a sequence number. If the underlying transport is unreliable or unordered, additional replay protection mechanism should be provided. If the underlying transport is reliable but unordered, data dropping detection should be provided.
- Data fragmentation and PMTU discovery: the provision of data fragmentation and PMTU discovery affects the maximum size of a data record that can be delivered.
- Checksum: if the checksum verification of a transport protocol is optional or allows coverage selection, the security protocol on top of the transport protocol could make data authentication optional.

The analysis of the services provided by transport protocols done in this section help us to examine how well the security protocol developed in this thesis supports the services provided by the transport protocols (TCP, UDP, SCTP, and DCCP).

Table 1. Comparison Table of Services provided by TCP, UDP, SCTP, and UDP

| Service\Protocol | TCP | UDP | SCTP | DCCP |
|---|---|---|---|---|
| **Traffic Multiplexing** | √ | √ | √ | √ |
| **Connection Setup/teardown** | Explicit | Implicit | Explicit association setup | Explicit |
| **Duplex mode** | Bidirectional full-duplex | Bidirectional full-duplex | Unidirectional | Bidirectional full-duplex |
| **Flow control** | √ | × | √ | √ |
| **Congestion control** | √ | × | √ | √ |
| **Multi-homing** | × | × | √ | √ |
| **Transport-layer mobility** | × | × | √ (with SCTP dynamic address reconfiguration extension) | √ |
| **Data boundary** | Byte | Datagram | Message-oriented | Datagram |
| **Multi-streaming** | × | × | √ | × |
| **Service reliability** | Reliable | Unreliable | -Reliable -Partial reliable (PR-SCTP associations) | Unreliable |
| **Ordered delivery** | √ | × | Ordered delivery within a stream | × |
| **Unordered delivery** | × | √ | √ | √ |
| **Data Fragmentation** | √ | × | Optional | × |
| **PMTU Discovery** | √ | × | √ | √ |
| **MPS Discovery (Max Packet Size)** | × | × | × | √ |
| **Checksum** | Mandatory | Optional | Mandatory | Mandatory but with coverage selection |

# 10. Developing a New Transport Security Protocol

As described, the development of the new security protocol should be based on TLS and DTLS. Before this, an analysis of the components of TLS and the relationship between them was made. The interface between the TLS handshake protocol and the TLS record protocol was carefully studied. DTLS is very similar to TLS in its structure and the interface between its components. Therefore, the result of the analysis of TLS also applies to DTLS.

I used the result of the analysis to guide the development of the new security protocol. The initial approach was to remove the handshake sub-protocol and the changecipherspec sub-protocol of TLS and DTLS. I called this reduced version of TLS and DTLS 'Raw TLS'. An implementation and performance evaluation of 'Raw TLS' was done. According to the results of the performance evaluation, the removal of inband handshake does reduce initialization delay outstandingly especially in networks with high latency. However, after further analysis, it was concluded that a more radical change than 'Raw TLS' must be made to the original specification of TLS and DTLS basically due to the lack of key refreshment and rekeying mechanisms and the compatibility problems with some transport protocols.

Based on the above, I designed and evaluate another security protocol called transport encapsulation security payload (TESP), which is the main result of this thesis. TESP fully supports all the transport protocols and can be used to reach the goal of this thesis. A security analysis was also done on TESP.

## 10.1. Analysis of Components of TLS

**A Modular View of TLS**

As described in section 4.3.2, and as illustrated graphically in Figure 5, TLS is actually composed of two protocols; the TLS record protocol and the TLS handshake protocol. The TLS record protocol can be seen as a payload security protocol. In other words, the TLS Record Protocol is the lower layer of the TLS protocol and is the carrier of both data and handshake messages. It takes security parameters negotiated by the TLS handshake protocol to setup cryptographic states and provides data confidentiality and integrity protection service to the upper layer.

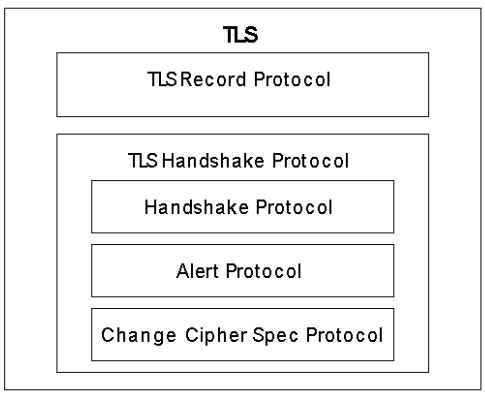



Figure 5. A modular view of TLS

The TLS handshake protocol is further divided into three sub-protocols: the handshake sub-protocol, the 'change cipher spec' sub-protocol, and the alert sub-protocol. The handshake sub-protocol can be seen as the inband key management protocol. It sends handshake messages to perform key management. As stated earlier, the handshake messages are also carried by the record protocol just as the application data are. It is the type field in the record header that is used to differentiate application data records from handshake messages. The 'change cipher spec' sub-protocol is for the synchronization of rekeying. All the data records following the "ChangeCipherSpec" message are encrypted and authenticated using the new security parameters exchanged by the TLS handshake just performed. The alert sub-protocol is designed to exchange warnings and fatal error notifications. An example is the "closure alert" that is sent from the end that wants to terminate the connection. The closure alert could help the upper layer detect attacks on TCP connections by sending a fake FIN to maliciously terminate an ongoing TCP connection.

**Interface Between TLS Record Protocol and TLS Handshake Protocol**

As illustrated in Figure 6, the security parameters and the 'change cipher spec' sub-protocol together can be seen as the interface between the TLS record protocol and the TLS handshake protocol. The security parameters are used to setup cryptographic states in the TLS record protocol. The 'change cipher spec' sub-protocol is used to synchronize the rekeying each time a new set of security parameters is exchanged by the handshake sub-protocol. It is actually possible for the TLS record protocol to use the same interface to cooperate with an external key management protocol instead of the handshake sub-protocol of TLS. The alert sub-protocol can be seen as an assistant protocol that helps the TLS record protocol in exception handling.

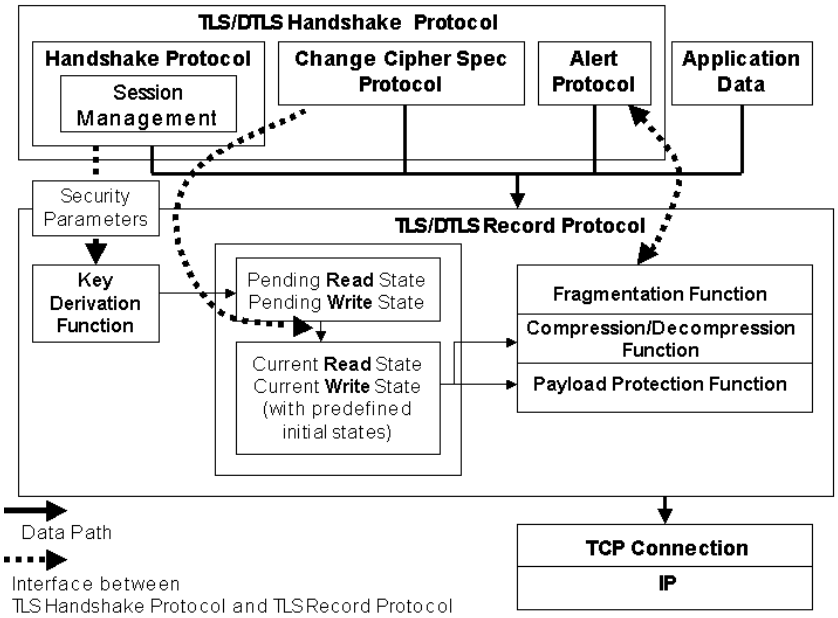



Figure 6. Relationship and Interfaces between TLS/DTLS components

**Data Objects of TLS**

There are three major data objects in the TLS protocol: the TLS security parameter data object, the TLS connection state data object, and the TLS session data object.

- TLS Security Parameters: they are part of the interface between the TLS handshake protocol and the TLS record protocol. According to the TLS specification, the security parameters include connection end, bulk encryption algorithm, cipher type, key size, key material length, MAC algorithm, hash size, compression algorithm, master secret, client random number, and server random number. The reader can refer to the TLS specification [23] for a detailed description of each parameter.

- TLS Connection State: the TLS connection state contains all the parameters needed by the TLS record protocol to perform compression and cryptographic transformation on plaintext data. The TLS record protocol maintains four security states: the current write state, the current read state, the pending write state, and the pending read state. The current read/write states contain the parameters that are currently used to encrypt and authenticate application data and handshake messages. After the handshake sub-protocol exchanges a set of security parameters, they are handed to the key derivation function of the TLS record protocol to derive new parameters for the read/write connection states. These new connection state parameters are stored into the pending read/write state as illustrated in Figure 6. It is the 'change cipher spec' sub-protocol that synchronizes the record protocol of both parties participating in the TLS connection by overwriting the current state security parameters with the pending state security parameters. As stated above, the record compression/ decompression function and the record payload protection function of TLS record protocol always use the current read state and the current write state parameters to process data payloads. A write connection state of a client consists of the following parameters:
    - Compression state: the current state of the compression algorithm, which should be dependent on the algorithm chosen.
    - Cipher state: should include the keys and IVs for encryption/decryption. They are derived from the security parameters as described earlier.
    - Sequence number: the sequence number of the record being processed.
    - Message Authentication Code (MAC) secret
- TLS Session: the TLS 'session' concept is used to reuse the master secret that is already shared by both of the communication parties for setting up more than TLS connections. The TLS session management is part of the handshake protocol. The master secrets are indexed by the session ID. If a TLS client that starts a new TLS connection intends to reuse a certain master secret, it sends the client hello message with the session ID pointing to the master secret it wants to reuse. If the server agrees, then both parties could derive parameters for the TLS connection states by reusing the old master secret. This approach is called session resumption. The benefit of doing session resumption is that if the client opens multiple TLS connections, it saves time and computing power by not doing a full TLS handshake for each TLS connection. A session contains the the session ID and the master secret as its parameters

**Key Derivation Function**

After a TLS handshake is performed, a secret (i.e. the premaster secret) and two random numbers ('client random' and 'server random') are shared between the server and the client. This premaster secret is first converted into the master secret before further parameters of the TLS connection states are derived. PRF stands for pseudo-random function and is used to expand a secret into a longer block of data. It takes a secret and several seeds and generates an output of arbitrary length. However, as most PRFs generate a cycle, the output can only be used for a limited portion of the cycle.

master secret = PRF(pre_master_secret, "master secret", client random + server random)

The master secret is always exactly 48 bytes (384 bits) in length.

For non-export block ciphers:

key block = PRF (master secret, "key expansion", client random + server random )

The key block is then partitioned into the following parameters of a TLS connection state: 'Client write MAC secret', 'Server write MAC secret', 'Client write key', 'Server write key', 'Client write IV', and 'Server write IV'.

**Key Refreshment**

Key refreshment refers to the refreshment of the session keys and can be achieved by performing session resumption within an already established TLS connection. See section 4.3.2 for the description of TLS session resumption.

**Rekeying**

Rekeying refers to the renewal of master secret and can be achieved by performing a full TLS handshake within an already established TLS connection.

## 10.2. Raw TLS

The idea was to extract the TLS/DTLS record protocol and make it a new security protocol itself without inband handshake, but with explicit interface to external key management protocols. The intention was to reuse as much as possible of TLS and DTLS so that the implementation of the new protocol would be both less difficult and more likely to be secure. This security protocol was given the name 'Raw TLS' to imply that this security protocol is 'TLS/DTLS without inband handshake'.

The change from TLS/DTLS to 'Raw TLS' is illustrated in Figure 7. As one can see, the TLS/DTLS handshake protocol is removed except for the alert protocol. The alert protocol is kept because it is still needed by 'Raw TLS'. The pending connection states in the record protocol are removed. When the key derivation function receives the security parameters from the external key management protocols, it derives the connection state parameters and then directly puts them in the current states before the data transmission starts as illustrated in Figure 8.

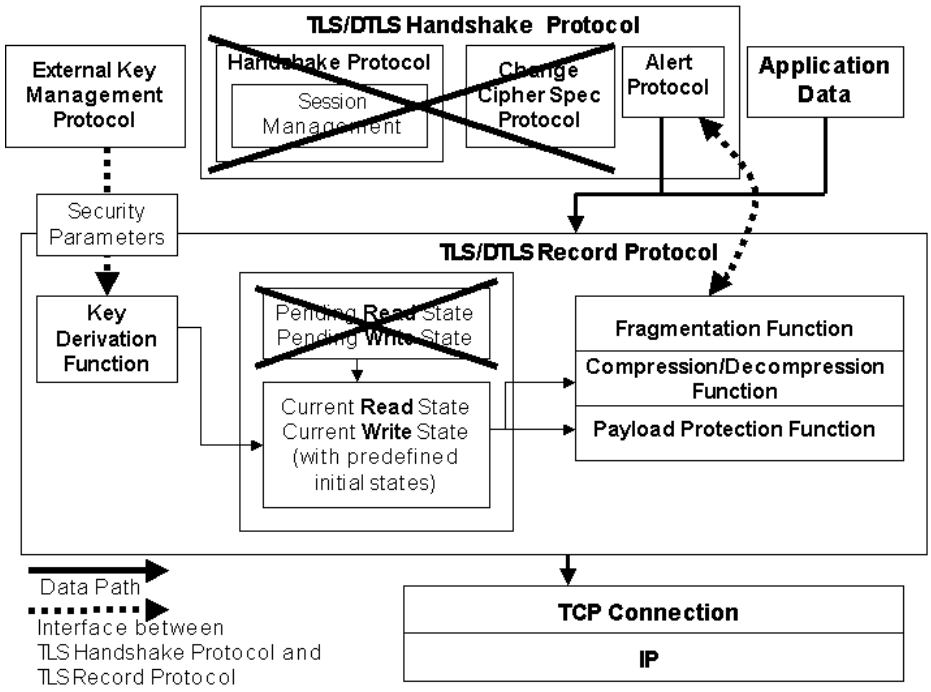



Figure 7. The Change from TLS/DTLS to Raw TLS

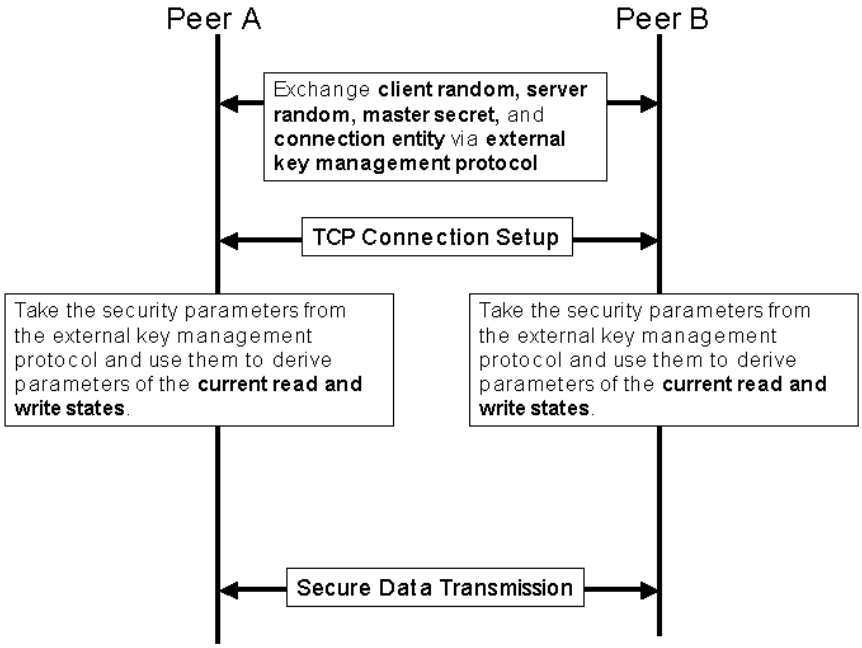



Figure 8. The Process of Performing Secure Data Transmission with 'Raw TLS'

**Sub-Protocols**

'Raw TLS' is composed of two sub-protocols: the TLS/DTLS record protocol and the alert protocol. The TLS record protocol is used when the underlying transport layer is a TCP connection or a SCTP stream. The DTLS record protocol is used when the underlying transport is an UDP session, an PR-SCTP stream, or a DCCP session.

**Security Parameters**

The security parameters of 'Raw TLS' are the same security parameters used in TLS/DTLS as the interface between the TLS/DTLS handshake protocol and the TLS/DTLS record protocol. The security parameters are used as the interface to the external key management protocols.

**Connection States**

The connection states in the record protocol of 'Raw TLS' contain the same parameters as are contained in the current/pending connection states of TLS/DTLS. There is a set of connection states for each TCP connection, UDP session, SCTP unidirectional stream, or DCCP session.

**Key Derivation**

The key derivation function basically follows the key derivation function of TLS/DTLS.

**The Alert Protocol**

The alert protocol of 'Raw TLS' is the same as the TLS/DTLS alert protocol. However, some alert descriptions regarding the TLS handshake protocol in the TLS/DTLS alert protocol are not needed by 'Raw TLS', as they are not relevant.

## 10.3. Implementation of 'Raw TLS'

The implementation of 'Raw TLS' was based on the OpenSSL version 0.9.7e library and look like an extension of the library. The development was done in the RedHat 9.0 distribution environment with a version 2.4.20-8 Linux kernel.

### 10.3.1. Introduction to OpenSSL

In this section, only a very short introduction to the most important data structures of the OpenSSL library is given. Those who are interested in OpenSSL programming should refer to [71] and [72] for more details.

The first data structure to introduce is the method object called 'SSL_METHOD'. This method object represents the internal implementation of OpenSSL's functionality. It contains the function pointers to some important internal functions. There is a different implementation of OpenSSL's internal functionality for each version of SSL/TLS protocol OpenSSL supports. The SSL/TLS protocol versions OpenSSL currently supports include SSL version 1, SSL version 2, and TLS version 1. The first thing an application must do to initialize the library is to specify the version of SSL/TLS to use by creating a method object and point the function pointers in this object to the functions of the SSL/TLS version the application wants to use.

After the method object is created, an application then uses the method in the method object to create a single context object called 'SSL_CTX'. The SSL context object is a factory that produces SSL connection objects for each SSL connection. The SSL context object can be seen as a container of the default values of all the SSL connections that are going to be setup afterwards. Examples of these default values of the SSL connections include protocol version, certificate information, certificate verification requirements, etc.

OpenSSL is designed in such a flexible way that secure connections can be run on top of different types of transport I/O and a TCP connection is only one of them. Therefore, before an SSL connection object can be created, an I/O object called 'BIO' needs to be created by an application and the type of the I/O object should be specified.

After the method object (SSL_METHOD), the context object (SSL_CTX), and the I/O object (BIO) are created, the application can then create an SSL connection object called 'SSL'. The values of the default parameters are copied from the context object. The application then performs SSL/TLS handshaking using the SSL connection object on the transport I/O provided by the I/O object. The version of method is determined by the method object.

The following is an incomplete section of sample source code written in C demonstrating the initialization and the usage of the data objects introduced above with the OpenSSL API on the client side:

```
/*The following code creates a method object called 'method' and specifies the
version to be TLS version 1*/
SSL_METHOD *method;
method = TLSv1_method();

/*The following code creates and initializes a context object called 'ctx' with
the method object */
SSL_CTX *ctx;
ctx = SSL_CTX_new(method);
SSL_CTX_use_certificate_chain_file(ctx, CERTFILE);
SSL_CTX_use_PrivateKey_file(ctx, CERTFILE, SSL_FILE_PEM);

/*The following code creates I/O object called 'conn' for a TCP connection*/
BIO *conn;
conn = BIO_new_connect(SERVER:PORT);
BIO_do_connect (conn);

/*The following code creates an SSL connection object called 'ssl' with the
context object and the I/O object*/
SSL *ssl;
ssl = SSL_new(ctx);
SSL_set_bio (ssl, conn, conn);

/*The following code performs the SSL handshake and establishes an SSL
connection*/
```

```
SSL_connect(ssl);

/*The following code sends and receives data using the established SSL
connection*/
SSL_write(ssl, WRITE_BUFFER, LENGTH);
SSL_read(ssl, READ_BUFFER, LENGTH);
```

The initialization and the usage of the data objects in the server side are very similar to that of the client. One major difference is that in the server, SSL_accept( ) instead of SSL_connect( ) is called to passively wait for the client endpoint to start TLS handshake.

Another important data structure is the session object called 'SSL_SESSION'. This session object is used by the session resumption function of SSL/TLS. Each time a SSL connection is established with a full SSL/TLS handshake, a session ID is assigned to the SSL connection by the server. The session ID and other security parameters such as the master secret and the protocol version are stored in memory in the form of a session object in both endpoints. In this way, a resumed handshake can be performed by both endpoints reusing the parameters stored in the session object.

### 10.3.2. Details of the Implementation

The implementation is a realization of 'Raw TLS' as described in the previous section. The implementation serves the following purposes:

- A proof-of-concept: the implementation was done to examine the feasibility of the idea of separating the TLS handshake protocol and the record protocol.
- As practice to gain some hands-on experience with OpenSSL programming
- To improve my understanding of the TLS specification and security protocols in general
- So as to allow a performance evaluation.

'Raw TLS', which is illustrated again in Figure 9, can be considered TLS without inband handshaking. As already stated clearly, the intention of a design was to reuse the components of TLS to make the implementation of 'Raw TLS' easier. In fact, it can be implemented by simply removing the change cipher spec sub-protocol and the handshake sub-protocol together with the session management function for session resumption from the TLS protocol.

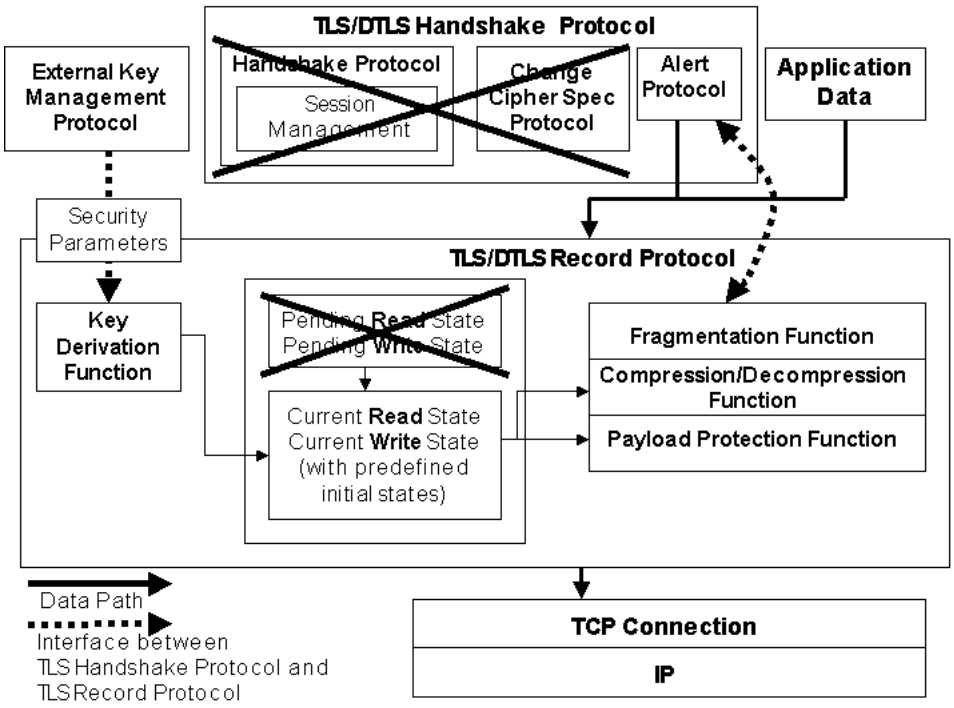



Figure 9 The Change from TLS/DTLS to 'Raw TLS'

In the OpenSSL library, 'Raw TLS' is presented as a new method. One cannot perform any inband handshake for key management with this new method. An application that uses the 'Raw TLS' method to setup TLS connections needs to use external key management protocols to exchange security parameters and configures TLS connections with these security parameters. To do this, a new data structure called 'RTLS_SESSION' is introduced. This data structure serves as the interface to the external key management protocols. RTLS_SESSION contains the security parameters that are needed by the TLS record protocol to establish a TLS connection.

```
struct {
    int type;
    int master_key_length;
    unsigned char master_key [SSL_MAX_MASTER_KEY_LENTH];
    unsigned char server_random [SSL3_RANDOM_SIZE];
    unsigned char client_random [SSL3_RANDOM_SIZE];
    unsigned int  compress_meth;
    unsigned int  mac_algorithm;
} RTLS_SESSION;
```

After an application successfully performs key management externally, it fills the RTLS_SESSION data object with the security parameters exchanged and calls the SSL_ctrl() function with the pointer of this data object as one of the parameters for the configuration of a 'Raw TLS' connection. After configuration, the application can immediately start sending and receiving data with the SSL_write() and SSL_read() functions.

The following is incomplete sample source code written in C. It demonstrates the initialization and the usage of the data objects with the new 'Raw TLS' method on the client side:

```
/*The following code creates a method object called 'method' and specifies the
version to be 'Raw TLS' */
SSL_METHOD *method;
method = RTLS_method();

/*The following code creates and initializes a context object called 'ctx' with
the method object */
SSL_CTX *ctx;
ctx = SSL_CTX_new(method);
```

```
//The following code creates I/O object called ‘conn’ for a TCP connection
BIO *conn;
conn = BIO_new_connect(SERVER:PORT);
BIO_do_connect (conn);

/*The following code creates an SSL connection object called ‘ssl’ with the
context object and the I/O object*/
SSL *ssl;
ssl = SSL_new(ctx);
SSL_set_bio (ssl, conn, conn);

/* The following code feeds the RTLS_SESSION data object with the security
parameters that are exchanged by an external key managemet protocol */
RTLS_SESSION rtls_session;
rtls_session.type = RTLS_SERVER;
rtls_session.master_key_length = SSL_MAX_MASTER_KEY_LENGTH;
strncpy (rtls_session.master_key, master_key, SSL_MAX_MASTER_KEY_LENGTH);
strncpy (rtls_session.server_random, server_random, SSL_RANDOM_SIZE);
strncpy (rtls_session.server_random, client_random, SSL_RANDOM_SIZE);
rtls_session.compress_meth = COMP_NULL;
rtls_session.cipher = cipher;
rtls_session.mac_algorithm = mac _algorithm;
SSL_ctrl (ssl, SSL_CTRL_RTLS_SETUP_NEW_SESSION, 0, &rtls_session);

//The following code send and receive data using the established SSL connection
SSL_write(ssl, WRITE_BUFFER, LENGTH);
SSL_read(ssl, READ_BUFFER, LENGTH);
```

I also implemented key refreshment and rekeying (refreshment of the master secret) by adding the TLS change cipher spec protocol and resumed TLS handshake but soon realized that these mechanisms are not compatible for other transport session than TCP connections (see section 10.4.2). The attempt to find a suitable mechanism of doing rekeying and key refreshment leads to the conclusion that Raw TLS is still not a complete solution. The author then started designing TESP.

### 10.3.3. Performance Evaluation

Since ‘Raw TLS’ retains the design of the TLS record protocol and the DTLS record protocol, the only performance gained by adopting ‘Raw TLS’ is the decrease in the initialization delay because of the removal of the inband handshake. The measurement of this gain is described in this section.

To demonstrate the performance difference between TLS connections and ‘Raw TLS’ in the user scenario concerned in this thesis, the following information was acquired during the measurement performed in this thesis work:

- The initialization delay for opening several TLS sessions in parallel: the total time needed to setup one TLS connection with full handshake and the setup of TLS connections in parallel with TLS resume handshakes, when there is more than one TLS connection to be opened at once, was measured. The time measured includes time for performing TCP three-way handshakes for each TLS connections. The number of TLS connections to be opened in parallel ranges from one to ten. **Note**: Only **after** the communicating endpoints complete a TLS full handshake can they start opening several TLS connections in parallel with TLS resume handshakes.
- The initialization delay for opening several ‘Raw TLS’ sessions in parallel: the total time needed to setup several ‘Raw TLS’ session in parallel was measured. Since the key management is done by external key management protocol and that is piggybacked on the session setup protocol, the initialization delay for opening ‘Raw TLS’ sessions is only the time needed for performing TCP’s three-way handshakes.

Since the user scenario concerned in this thesis happens in a heterogeneous network environment, the measurement was done in a network environment where one of the two communicating endpoints is attached to a wireless network while as another endpoint is attached to a nearby wired network, as illustrated in Figure 10. It is assumed that the round trip time (RTT) of the path between the wireless network and the wired network is under 10 milliseconds and is negligible, to reduce the number of factors of the measurement. Therefore, it is the delay in the wireless network that primarily affects the delay between the endpoints in the emulated network environment.

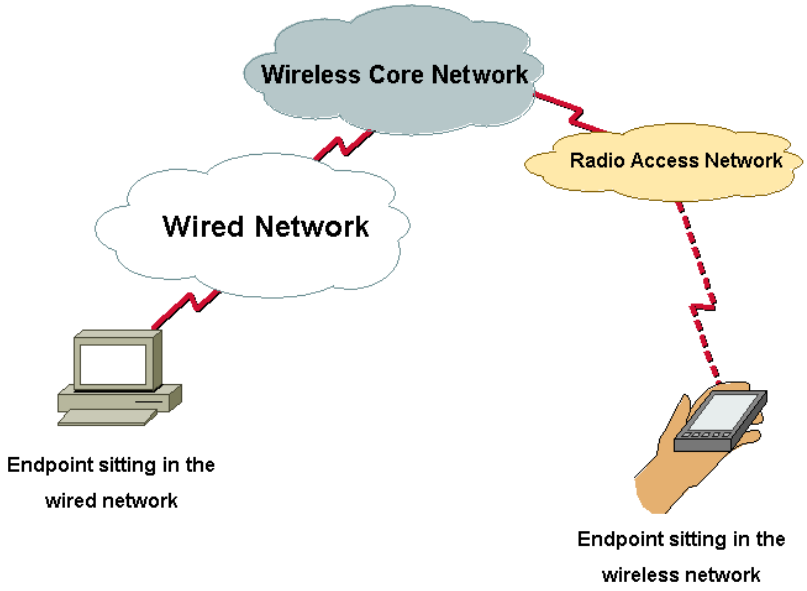
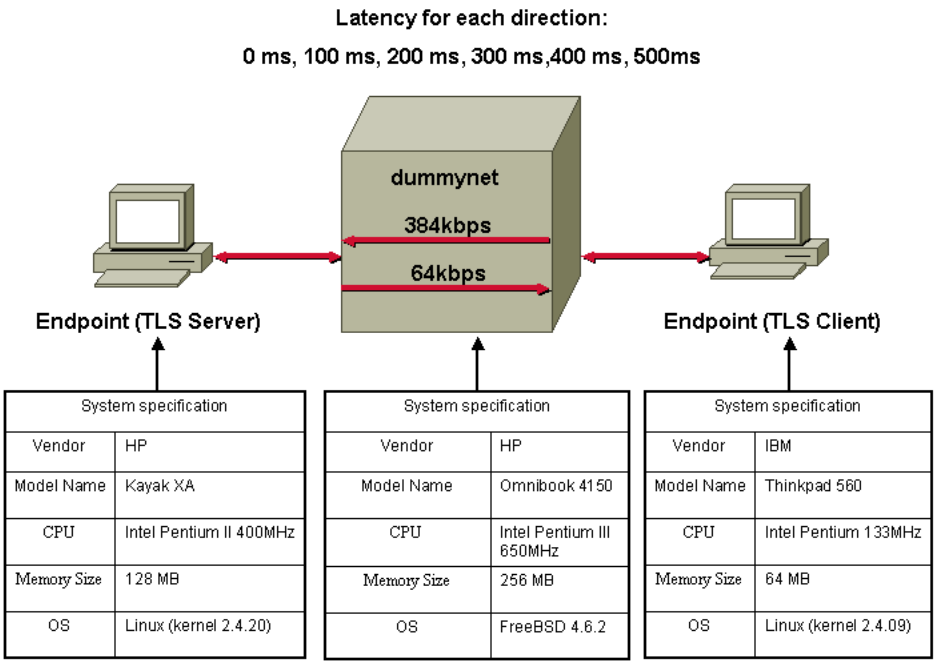



Figure 10. The heterogeneous network environment

Regarding the radio technology of the wireless network, this thesis only considered 3G cellular technologies (e.g. WCDMA and cdma2000). According to RFC 3481[38], the round trip time (RTT) of 3G wireless links varies between a few hundred milliseconds and one second. Initial 3G systems have bit rates around 64 kbps in uplink and 384 kbps in downlink. The link layer retransmission and forward error correction schemes provide a packet service with a negligibly small probability of undetected errors. A 3G link here includes the air interface, the radio access network, and the core network (i.e. from mobile station to the gateway to the wired network of the core network).

Measurements with both a real 3G wireless link and a emulated 3G wireless link were performed. The measurement over the real 3G wireless link is done by using a WCDMA PC card (a Vodafone Mobile Connect 3G/GPRS datacard, which is manufactured by Option Wireless Technology. The model number of the card is 129 and firmware version is 3.1.2) to access the WCDMA network of the Ericsson experience center with a indoor base station situated one floor under my office within the same building. The emulated wireless link used the dummynet [73], a tool for testing networking protocols. The characteristics of 3G wireless links described in RFC 3481 were used to parameterize the emulated wireless link. The emulation environment is illustrated in Figure 11. The reason of performing measurements over the emulated wireless link is that the link condition of the real 3G wireless link used in the measurement does not represent a general link condition of a 3G wireless link. The link conditions between a mobile station and the base station connected varies according to many factors such as the distance from the base station of a mobile station, the movement- both speed and direction of a mobile station, the obstacles in between them, etc. The emulated wireless link was used to explicitly emulate the RTT of various link conditions.




| System specification | |
|---|---|
| Vendor | HP |
| Model Name | Kayak XA |
| CPU | Intel Pentium II 400MHz |
| Memory Size | 128 MB |
| OS | Linux (kernel 2.4.20) |

| System specification | |
|---|---|
| Vendor | HP |
| Model Name | Omnibook 4150 |
| CPU | Intel Pentium III 650MHz |
| Memory Size | 256 MB |
| OS | FreeBSD 4.6.2 |

| System specification | |
|---|---|
| Vendor | IBM |
| Model Name | Thinkpad 560 |
| CPU | Intel Pentium 133MHz |
| Memory Size | 64 MB |
| OS | Linux (kernel 2.4.09) |

Figure 11 The dummynet simulation environment

The initialization delay measured does not include the delay caused by the handshake of the session setup protocols, as the delay caused by the handshake of session setup protocols does not vary with different security protocol examined.

In each environment, the measurement was repeated 100 times. The statistics, such as the mean, the median, the upper quartile, the lower quartile, the maximum, and the minimum value were calculated from these data sets. All of the measurement results are shown in the figures in Appendix A. The results of one of the measurements done with the real WCDMA link are shown in Figure 12. As one can see from the figures in Appendix A, using 'Raw TLS' significantly reduces the initialization delay in network environments, (both emulated wireless link and real WCDMA wireless link). Additionally, the measurements done with the emulated wireless link shows that the greater the latency of the wireless link is, the more obvious the gain becomes.

To represent the median values of the measurement results mathematically, a linear least square fitting of these results was done (where 't' is the initialization delay, 'n' is the number of sessions opened in parallel) because it looks obvious from the figures of the measurement results that t(n) is a linear function:

```
For RTT = 0 ms;
     t(n) = 245.5                          for n = 1
     t(n) = 92.1 * n + 151.96              for 10 ≥ n ≥ 2
For RTT = 200 ms:
     t(n) = 828.25                         for n = 1
     t(n) = 41.16 * n + 1232.72            for  10 ≥ n ≥ 2
For  RTT = 400 ms
     t(n) = 1431                           for n = 1
     t(n) = 38.34 * n + 2241.51            for  10 ≥ n ≥ 2
For  RTT = 600 ms
     t(n) = 2031.25                        for n = 1
     t(n) = 39.17 * n + 3241.67            for  10 ≥ n ≥ 2
For  RTT = 800 ms
     t(n) = 2631                           for n = 1
     t(n) = 38.41 * n + 4249.06            for  10 ≥ n ≥ 2
For  RTT = 1000 ms
     t(n)= 3235.5                          for n = 1
     t(n) = 38.84 * n + 5246.34            for  10 ≥ n ≥ 2
```

It is also obvious that the t-intercept of the function is a linear function of RTT; and the slope of the function is constant except when RTT is zero. By introducing RTT into the t() function as another factor, I got:

```
t(n, RTT) = 245.5                          for n =1, RTT = 0 ms
t(n, RTT) = 92.1 * n + 151.96              for 10 ≥ n ≥ 2, RTT = 0 ms
t(n, RTT) = 3.01 * RTT + 224.85            for n = 1,  1000 ms ≥ RTT ≥ 200 ms
t(n, RTT) = 40.55 * n + (5.07 * RTT + 191.35)
                                           for 10 ≥ n ≥ 2, 1000ms ≥ RTT ≥ 200 ms
```

One can observe from the result of the least square fitting that when RTT is zero, the initialization delay is more affected by the number of sessions opened (higher slope). It is because when there is no delay on the network, it is the processor's processing time that makes the most of the initialization delay.

The sharp rise of the initialization delay between one session opened and two sessions results from the fact that only **after** the communicating endpoints complete a TLS full handshake can they start opening several TLS connections in parallel with TLS resume handshakes when there are more than one session opened in parallel.

One strange thing is the high variance of the initialization delay in the result of measurement in the real WCDMA environment both in the case of running TLS and 'Raw TLS'. The max values of the initialization delay in the measurement are very far away from the median value. For example, when the number of sessions opened in parallel is eight, the max value of the initialization delay for running TLS went up to around seven seconds, whereas the median value is less than 2 seconds. With the same number of sessions, the max value of the initialization delay for running 'Raw TLS' is more than three seconds, whereas the median value is merely several hundred msecs. However, since the upper quartiles and the mean values are still close to the median value despite of this high variation, it was determined that this kind of long initialization delay only happens occasionally. The reason of this abnormal observable fact is unclear and further investigation should be done.

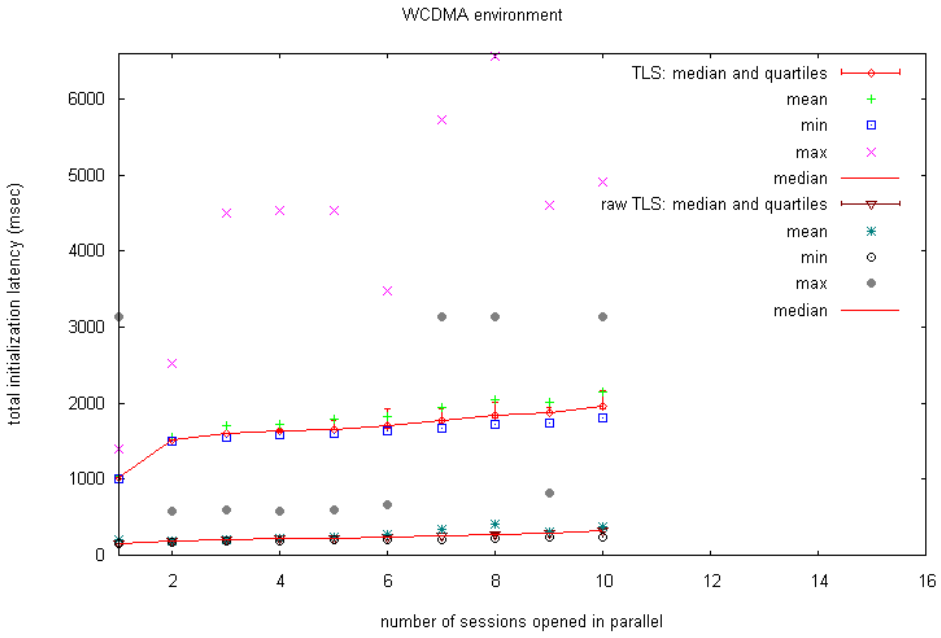



Figure 12. Performance of TLS and 'raw TLS' in real WCDMA environment

## 10.4. Analysis of 'Raw TLS'

### 10.4.1. Compatibility of 'Raw TLS' with Different Transport Layer Protocol

**Compatibility with TCP**

'Raw TLS' fully supports the services provided by TCP connections.

**Compatibility with SCTP**

'Raw TLS' supports both SCTP associations and PR-SCTP associations. It can also run over one unidirectional stream and data streams do not have to be paired into bi-directional streams because there is no inband handshake. However, the 'unordered delivery' feature of SCTP still cannot be used because 'Raw TLS' has no mechanism to do replay protection over reliable but unordered delivery.

Another new issue is that the security parameters of 'Raw TLS' are per stream instead of per association. Therefore, each stream within a SCTP association needs a set of security parameters from the external key management protocol. If the application does not know the number of SCTP streams that will be used during the session setup phase for the SCTP association, a key management exchange should be performed each time it opens a new stream. This causes unnecessary latency.

Yet another issue here is that the record protocol of TLS was originally designed to support TCP, which sees data as byte streams. This is the reason why in the record header, there is a length field. But SCTP, on the other hand, is a message-oriented protocol. Each time the application receives some data from an SCTP stream, it must be a complete message. As long as each data record is sent to the network as a single SCTP message (it is quite natural to have this one-to-one mapping between TLS record and SCTP message), it makes no sense for TLS to fragment a record and send it in more than one message because SCTP does fragmentation itself. I do not see any performance gain from concatenating data records and sending them as one SCTP message, thus the length field of the record header could actually be removed to gain slightly better bandwidth utilization.

**Compatibility with UDP**

DTLS is only an IETF Internet draft. Although it claims that it is compatible with datagram transport layer protocols such as UDP, since it is still a work-in-progress, there is no existing open source implementation that proves that the protocol actually works well with UDP. In fact, I found several issues about the DTLS record protocol when it is used to protect UDP sessions. These issues were not addressed in the DTLS specification.

- UDP is a datagram-oriented transport layer protocol. The receiving application either receives a complete datagram or not. If each data record is sent to the network as a single datagram, the length field of the record header can be removed to achieve slightly better bandwidth utilization. However, the DTLS record protocol retains the length field in the record header. The epoch field of DTLS record header is also redundant.
- In UDP sessions, even a blind attacker can easily perform a resource attack by sending a large amount of records with arbitrary content if the decryption is done before authentication. The DTLS record protocol follows the design of the TLS record protocol and encrypts the message authentication tags of records. This makes the DTLS record protocol vulnerable to the resource attack.
- In the TLS specification, it says that upon sending or receiving a fatal alert message, both endpoints of the secure connection should immediately close the connection. The TLS record protocol is designed in such a strict way in reaction to receiving data records with a bad authentication tag because it relies on the security of the TCP

connection state to prevent a blind attacker from sending arbitrary data records with bad authentication tags. When the underlying transport session is UDP, it is rather easy for a blind attacker to do a DoS attack if the DTLS record protocol follows the design of TLS record protocol and reacts to a fatal alert message with a closedown of the transport session.

**Compatibility with DCCP**

DCCP protocol is a more intelligent datagram transport layer protocol. It implements Path MTU discovery and its own congestion control mechanisms. It maintains the maximum packet size (MPS), which is influenced by the PMTU, the maximum packet size allowed by the current congestion control mechanism, and the lengths of the IP and DCCP headers. The DTLS record protocol is not able to discover MPS when DCCP is the underlying transport protocol.

Additionally, since DCCP is also a datagram transport protocol, the 'length' field in the record header of the DTLS record protocol is also redundant.

### 10.4.2. **Other Issues**

'Raw TLS' does not provide any mechanism for doing key refreshment or rekeying. One possible means of giving 'Raw TLS' support for key refreshment and rekeying while still trying to be similar to TLS is as follows: key refreshment can be achieved by doing a renegotiation with a resumed TLS handshake within an already established TLS connection as illustrated in Figure 13. Since the session management is already removed, the session ID in the hello messages is ignored and a resume handshake is only valid in session renegotiation.

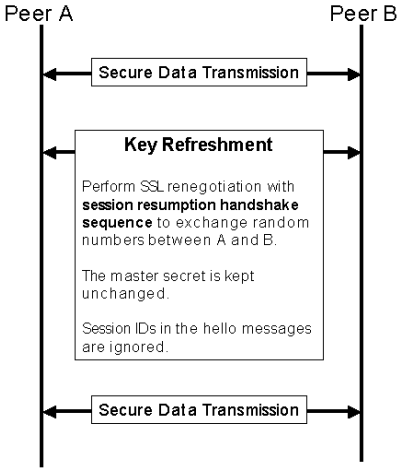



Figure 13. Key Refreshment

Rekeying, on the other hand, can be achieved by doing a renegotiation with a full handshake within an already established TLS connection. However, a full handshake costs two roundtrips. So an independent change cipher spec handshake message can be used for the synchronization of rekeying as illustrated in Figure 14.

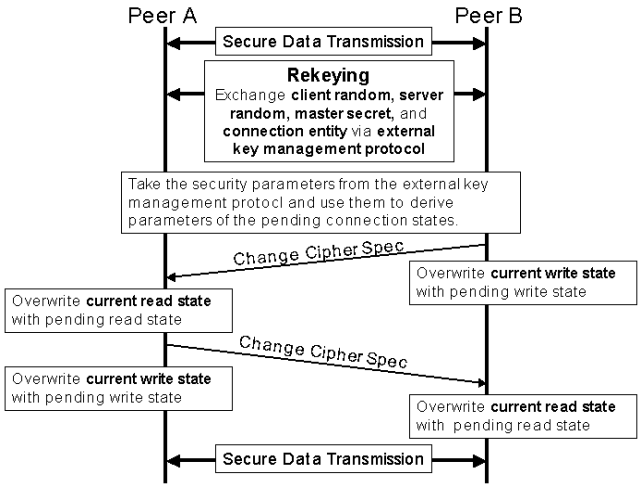



Figure 14. Rekeying with the 'Change Cipher Spec' Protocol

However, these solutions for key refreshment and rekeying have the following problems:

- The key refreshment does not support unidirectional sessions such as SCTP streams, because the resumed TLS handshake needs bidirectional transmission.
- The rekeying does not support unreliable sessions. If the change cipher spec message is lost during transmission, the following data will be incorrectly processed as illustrated in Figure 15.

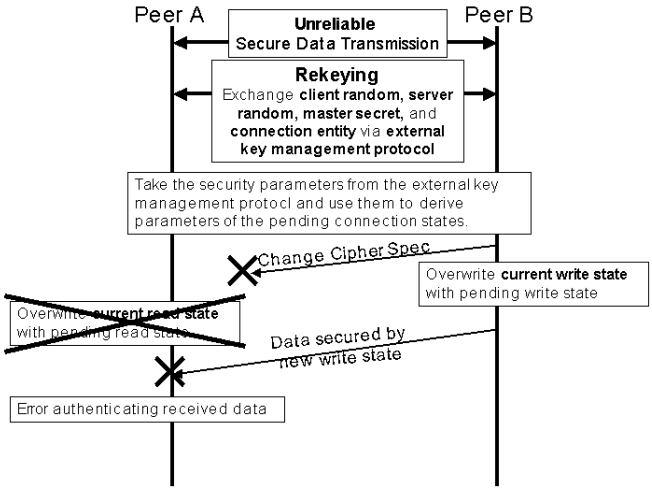



Figure 15. Rekeying Over an Unreliable Session

### 10.4.3. Conclusions

'Raw TLS' has several advantages over a solution with TLS and DTLS. There is no mandatory inband handshake before data transport can start. This helps the shortening reduce the initialization latency according to the measurement results shown in section 10.3.3 and Appendix A. There is also a clear interface of 'Raw TLS', which TLS does not have, to an external key management protocol.

However, 'Raw TLS' still has several shortcomings. It does not support single key management for multiple streams within a SCTP association. Key management has to be performed for each SCTP stream. Nor does it support the MPS discovery of DCCP. Additionally, the data encapsulation (i.e. the record header) is not optimal with regard to bandwidth utilization. Most important of all, it does not provide a mechanism for key refreshment and rekeying. It was

concluded that, in order to overcome the many shortcomings of this solution, a more radical change must be made to the original specification of the TLS record protocol and the DTLS record protocol to meet the following requirements:

- It should fully support all the services provided by TCP, UDP, SCTP, and DCCP, including the SCTP 'unordered delivery' feature.
- It should support single key management for multiple streams within a SCTP association.
- The data encapsulation should be optimized according to the type of the underlying transport layer protocol.
- It should be able to discover the maximum size of the data record that can be delivered using different approaches according to the type of the underlying transport layer protocol.
- It should provide a reliable mechanism to do key refreshment and rekeying.

## 10.5. TESP

Based on the conclusion drawn in the previous section, another protocol was designed in this thesis. This security protocol was given the name 'Transport Encapsulation Security Payload' (TESP) to indicate that this security protocol, on top of the transport layer, is similar to ESP in that it is a simple payload security protocol *without* inband key management, and also in that it fully supports all the features and services provided by TCP, UDP, SCTP, and DCCP transport. In this section, new terminologies for TESP were defined. A modular view of TESP will be given. The design of each sub-protocol of TESP including the TESP record protocol and the TESP alert protocol will be described.

### 10.5.1. TESP Definitions

Data unit: A Data Unit (DU) is a segment of data that is sent by the application or the TESP alert protocol to the TESP record protocol. A DU is compressed, cryptographically transformed, and encapsulated into a record by the TESP record protocol. The maximum size of DU that is allowed to be sent by the application is called a Maximum Data Unit (MDU).

Transport session: An instance of data transport service provided by the underlying transport layer protocol. TESP establishes secure connections over a transport session as illustrated in Figure 16. A transport session can be a TCP connection, a SCTP association, a UDP session, or a DCCP session.

Security parameters: security parameters are a set of parameters that need to be negotiated by both of the communicating endpoints to establish a secure connection as illustrated in Figure 17. The negotiation of security parameters is the job of external management protocols. Security parameters can be seen as the interface between key management protocols and TESP.

Secure connection: A secure connection is defined as a pair of states with a set of parameters maintained by both of the communicating endpoints as illustrated in Figure 17. A secure association is established over a transport session, such as a SCTP association, a TCP connection, an UDP session, or a DCCP session (or one may call it a DCCP connection), as illustrated in Figure 16. The secure connection state parameters are derived from a set of security parameters agreed by both endpoints.

Secure stream: A secure stream is defined as a pair of states with a set of parameters maintained by both of the communicating endpoints as illustrated in Figure 17. A secure stream provides a data transport service with data confidentiality and integrity to the upper layer. A secure stream belongs to a secure connection and there can be multiple secure streams within one secure connection. Each secure stream is given a secure stream ID. The secure stream state parameters

are derived from secure connection state parameters. Within one secure connection, a secure stream of a specific stream ID can be used only once. It is not allowed to reopen a secure stream with the same stream ID after it is closed unless rekeying is performed to renew the master secret of the secure connection. This is to prevent the occurrence of the possible two-time-pad when stream ciphers are used for encryption.

The number of secure streams over a certain secure connection depends on the type of the underlying transport session this secure connection is bound to. A secure connection can have multiple secure streams only if the underlying transport session can do data multiplexing. A secure connection can have multiple secure streams if it is established over a SCTP association. There can be only one secure stream per secure connection when it is over a TCP connection, a UDP session, or a DCCP session.

The reasoning behind this layering of security connection and security stream is that there can be multiple data streams in a transport data session (e.g. one can setup multiple SCTP streams over one SCTP association). In the case of this multiple-stream-in-one-session transmission, it is perfectly feasible for the key management protocol to exchange security parameters for each stream. However, it is very likely that the number of streams that are going to be setup for data transmission is not predictable during the phase of session setup (i.e. the key management phase). In this situation, the best the external key management protocol can do is to exchange the security parameters for a SCTP association (i.e. a secure connection). The security protocol then uses these parameters to further derive the parameters needed for each SCTP stream (i.e. a secure stream).

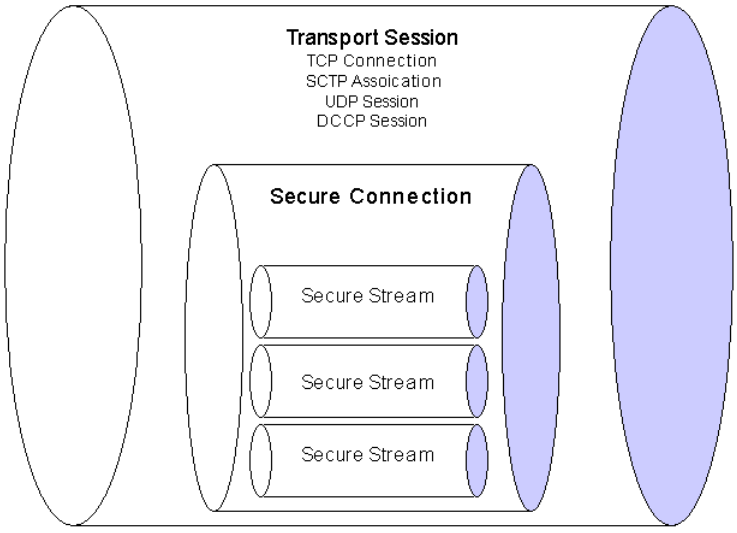



Figure 16. Transport session, secure connection, and secure stream

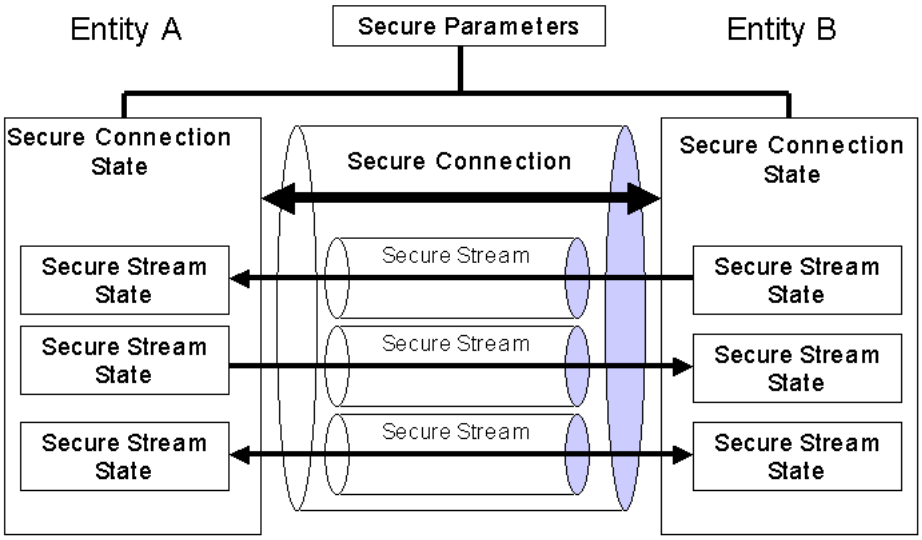



Figure 17. Security parameters and secure connection states and secure stream states

### 10.5.2. TESP Sub-Protocols

As illustrated in Figure 18, TESP is composed of two sub-protocols:

- Record protocol: The record protocol is the encapsulation protocol. It wraps application data and alert messages into encrypted and integrity protected data records. The record protocol of TESP is intended to be as similar to TLS as possible. Some minor changes are made to support various transport layer protocols and provide better bandwidth utilization.
- Alert protocol: the alert protocol serves the same purpose as the alert protocol in TLS and DTLS.

TESP is similar to TLS in the way that it reuses the sub-protocol structure of the TLS. In TESP, as in TLS, the record protocol is responsible for the encapsulation of both application data and the alert protocol.

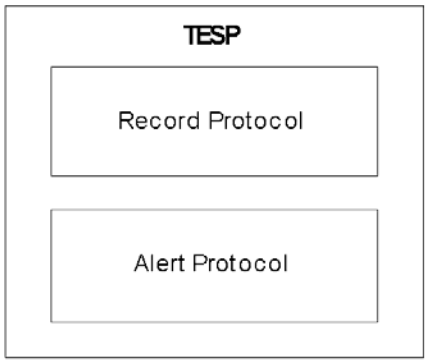



Figure 18. TESP sub-protocols

### 10.5.3. The TESP Record Protocol

I borrowed the presentation language used in the TLS specification to describe the state parameters as well as the input and output of each function blocks in the TESP record protocol.

#### 10.5.3.1. Message Authentication and Pseudo Random Number Generator

To be able to reuse the function blocks of TLS, TESP follows the design of TLS and uses the keyed hashing for message authentication (HMAC) [9] as the message authentication function with SHA-1 [11] as its hash algorithm. The HMAC function with SHA-1 algorithm is denoted as HMAC_SHA (secret, data). MD5 cannot be used because it has been found to have serious weaknesses [70].

TESP needs a secure pseudo random number generator to build key blocks of various lengths from a shared secret in a secure way. TESP borrows the pseudo random function PRF (secret, label, seed) from TLS. The PRF function can generate key blocks of arbitrary length according to the need of ciphers and HMAC functions used. HMAC_SHA and HMAC_MD5 are used together to build key blocks of arbitrary lengths in the PRF function. Therefore, even though weaknesses have been found in MD5, the PRF function of TLS is still considered a secure function. For detail of the design of this PRF, one should refer to the TLS specification.

#### 10.5.3.2. Ciphers and Modes

For flexibility, TESP is able to support both block ciphers and stream ciphers following the design of TLS.

The DES cipher is already considered not secure enough because of its short key length (56-bit long) making it vulnerable to brute-force attack [63].

RC4 is also not a suitable cipher for TESP. TESP supports unreliable and unordered transport sessions. However, RC4 is not suitable for unreliable and unordered data transport. The reason is that using RC4 with the same key to separately encrypt each data record could result in reuse of the same key stream for the encryption of different data segments, which is called “two-time pad” and can seriously compromise the confidentiality of data. The solution to this is to reseed the cipher engine for each encryption to generate different key streams. But by doing so, the generation of key streams with RC4 still has problems with efficiency because of the recommendation of discarding the first 512 bytes of RC4 key streams due to the lack of randomness in the first few bytes of the output key streams from the RC4 cipher [64].

3DES and AES in various operation modes are the default ciphers used by TESP. Both 3DES and AES can be used in CBC mode as a block cipher or in counter mode or in Output Feedback (OFB) Mode as a stream cipher.

In short, the default ciphers of TESP include two block ciphers: 3DES in CBC mode and AES in CBC mode, and four stream ciphers: 3DES in counter mode, AES in counter mode, 3DES in OFB mode, and AES in OFB mode. TESP is designed in a way that additional block or stream ciphers working in different modes can be defined.

### 10.5.3.3. Initialization Vector Generation

For block ciphers running in CBC mode, an IV is needed for encryption of each record. This IV for each record should be unpredictable before the record is encrypted, or the secure connection will be vulnerable to the chosen ciphertext attack [34].

For block ciphers running in counter mode (CM) or output feedback mode (OFB), an IV is also needed for encryption of each record. This IV does not have to be unpredictable before the record is encrypted because the chosen ciphertext attack does not apply to stream ciphers. However, the IV for counter mode of OFB mode block ciphers should be salted by a random number chosen to make the IV unpredictable before a secure connection is setup. In this way, the ciphertext is protected against pre-computation and time-memory tradeoff attack that can be done on the stream ciphers [65][66].

In the TLS specification, the IV for the first record is generated together with the session keys. The IV for subsequent records is the last ciphertext block from the previous record. In this way, the IVs except the IV for the first record are predictable before the records are encrypted and this makes TLS vulnerable to a chosen ciphertext attack.

In the TLS version 1.1 draft, the design of IV generation is changed so each record carries an IV that is generated by a cryptographically secure random number generator. In this way, IVs are not predictable and the same chosen ciphertext attack on TLS cannot be done. The drawback of this design is that when IVs are needed for encryption, each record has to carry its own IV. This adds additional overhead and is a waste of network bandwidth.

In SRTP, IVs for AES running in counter mode are salted by a salting key to make them (i.e. the IVs) unpredictable before a SRTP session is setup. However, the IV for each record is still predictable before the record is encrypted because the salting key can be public and exchanged during key management phase. For AES running in f8 mode (a variant of OFB mode), the IV for each record is unpredictable before the record is encrypted because the IV itself is first encrypted to get a ‘camouflage’ before it is used by the cipher for encrypting data.

In TESP, since the default cipher includes both block ciphers in CBC mode and block ciphers in counter mode, the IV should be unpredictable before the record is encrypted. There are two approaches to unpredictable IVs. The first way is to have a defined IV generation function in the security protocol so that both endpoints can generates IVs for each record. For example, an IV can be generated as the following:

```
IV =PRF (master secret, packet_index + stream ID + random number A + client
random number B)
```

The second way is to follow the design of the TLS version 1.1 draft so that each record carries its own IV. Each endpoint is free to choose its own IV generation function. TESP supports both of these approaches and the endpoints should negotiate which approach to use during the negotiation of the security policy.

### 10.5.3.4. Functional Blocks and Interfaces

The interfaces of the TESP record protocol are illustrated in Figure 19. The TESP record protocol takes security parameters from the external key management protocols to establish secure connections. The TESP record protocol supports two data paths, an outgoing one and an incoming one. In the outgoing data path, the TESP record protocol takes application data and alert messages as input and outputs data records. In the incoming data path, it takes data records as input and ouputs application data or alert messages.

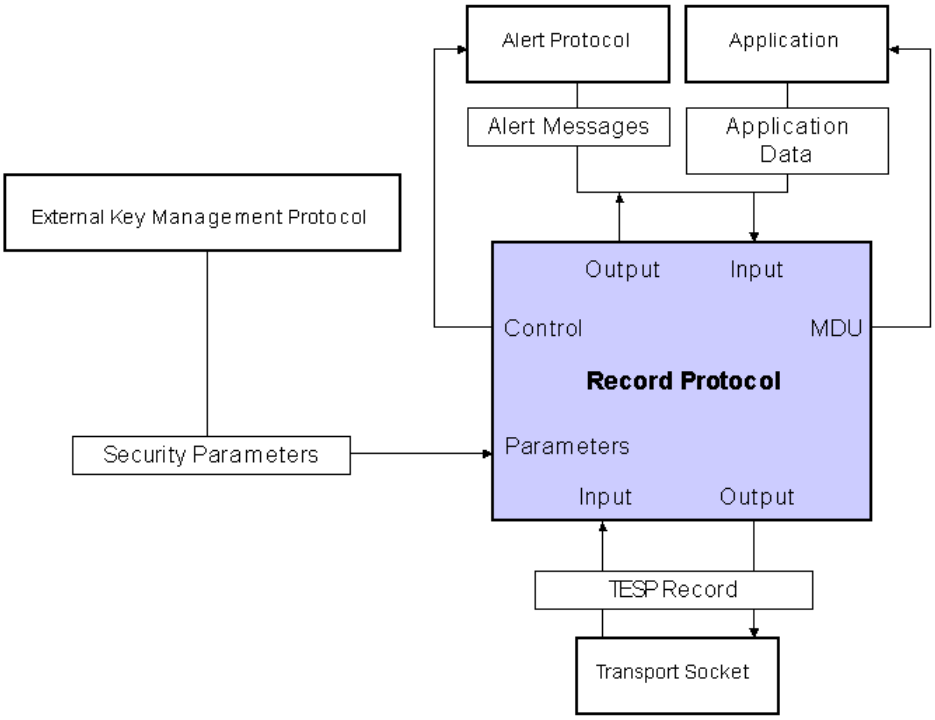



Figure 19 TESP Record Protocol

Internally, the TESP Record Protocol can be further divided into the following functional blocks, as illustrated in Figure 20. Each of function blocks of the TESP record protocol is described in the following sections. Before each function block in the TESP record protocol is described, one should first take a look at the following four data objects: security parameters, secure connection state, secure stream state, and data unit.

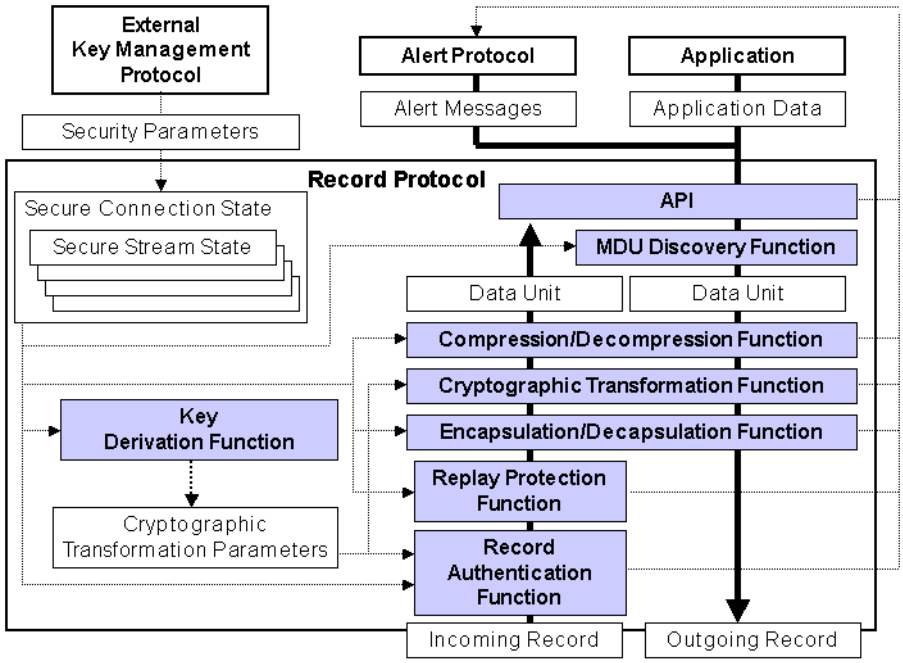



Figure 20 Function Blocks of the TESP Record Protocol

### 10.5.3.5. Security Parameters

Security Parameters are the interface to the external key management protocols of the TESP record protocol. These security parameters are used to setup a secure connection. The security parameters of TESP can be described as follows:

```
enum {TCP, UDP, SCTP, PR-SCTP, DCCP} TransportType;
enum {entity-A, entity-B} ConnectionEnd;
enum {stream, block} CipherType;
enum {3des-cbc, aes-cbc, aes-cm} CipherAlgorithm;
enum {default, external} IVGeneration;
struct {
     CipherAlgorithm cipher_algorithm;
     CipherType cipher_type;
     uint8 key_size;
     uint8 block_size; (for block cipher only)
} BulkCipher
enum {sha-1} HashAlgorithm;
struct {
     HasgAlgorithm;
     uint8 hash_size;
} MACFunction
enum { null (0), (255) } CompressionMethod;
enum { null, from-to, MKI } MasterKeyIDMethod
struct {
     MasterKeyIDMethod     master_ID_method;
     uint8                 MKI_size;
     opaque                MKI [MKI_size];
     uint48                from;
     uint48                to;
     uint48                lifespan;
     opaque                master_secret [48];
     opaque                random_source-A [64];
     opaque                random_source-B [64];
} MasterKeyInfo
struct {
     TransportType         transport_type;
     ConnectionEnd         entity;
     CompressionMethod     compression_method;
     BulkCipher            bulk_cipher;
     MACFunction           mac_function;
     IVGeneration          iv_generation;
     MasterKeyInfo         master_key_info;
     uint48                key_derivation_rate;
     uint16                max_no_of_streams;
} SecurityParamters;
```

**transport_type**

The type of transport session a secure connection is bound to. TESP supports TCP connections, SCTP associations, PR-SCTP associations, UDP sessions, and DCCP sessions.

**entity**

Each endpoint involved in a secure communication is called an entity. Unlike the entity parameter in the TLS specification, this parameter specifies the endpoint to be ‘entity-A’ or ‘entity-B’ instead of ‘server’ or ‘client’. This is because there is no handshake taking place in TESP and each endpoint is doing the same thing. Since TESP encrypts and authenticate data with separate keys for each direction, each endpoint should know which entity it is while performing key derivation. The entity should be negotiated during key management.

**compression_method**

The data compression method that is used for this secure connection.

**bulk_cipher**

Information about the cipher used including the algorithm of the cipher, type of the cipher (stream or block), the key size of the cipher, the block size of the cipher.

**mac_function**

The MAC function that is used in the cryptographic transformation function and the data authentication function. The default MAC function of TESP is HMAC SHA-1.

**master_key_info**

The master_key_info contains all the information about the master secret.

The master secret can be identified by either the Master Key Index (MKI) or the <from, to> pair. When the connection type is SCTP, it is strongly recommended that the <from, to> identification of the master secret should be avoided. The reason behind this is that the progress of each secure stream is different. If <from, to> is used to identify the master secret used in a particular secure connection, it might be possible that when two stream have different record indexes, one of the stream sees the master secret as invalid because the packet index has already exceeded the lifespan of the master secret specified by <from, to> while the other stream still sees the master secret as a valid one since its record index is still within the range of <from, to>.

**key_derivation_rate**

Establishes the rate of doing key refreshment. Key refreshment is updating session keys using the same master secret without doing key management signaling between the involved parties.

**max_no_of_streams**

The maximum number of secure streams that can be opened within a secure connection setup with the security parameters. The parameter is set to one when the secure connection is bound to a TCP connection, a UDP session, or a DCCP session. When the secure connection is bound to a SCTP association, this parameter is set to the sum of number of outbound streams (OS) and the number of inbound streams (MIS) of the SCTP association.

### 10.5.3.6. Secure Connection State

The concept of a 'secure connection' has already been stated in section 10.5.1. A secure connection is maintained by a pair of secure connection states in both endpoints and the secure connection state is described by the following data structure:

```
struct {
     SecurityParameters    security_parameters;
     uint16                MDU; (maximum data unit, in bytes)
     uint32                socket;
     SecureStreamState     secure_stream
     <1..security_parameters.max_no_of_streams>;
} SecureConnectionState;
```

**security_parameters**

A copy of security_parameters delivered by the external key management protocol. These parameters are needed by various functions of TESP and should be kept during the lifetime of a secure connection.

**MDU**

The MDU discovery function of TESP should discover the MDU for each secure connection and store this value is the SecureConnectionState.

**socket**

The socket of the transport session provided by the underlying transport layer protocol for the secure connection.

**secure_stream**

As stated earlier, in each secure connection, there can be one or more secure streams. Each secure stream opened within the secure connection has its own state as described in following section.

### 10.5.3.7. Secure Stream State

The concept of a 'secure stream' has already been stated in 10.5.1. A secure stream is maintained by a pair of secure stream states in both endpoints and the secure stream state is described by the following data structure. A secure stream can be described by the following data structures:

```
enum {outgoing, incoming, bidirectional} StreamDirection;
struct {
    uint32            stream_ID;
    SessionDirectoin  stream_direction;
    uint16            outgoing_sequence_number;
    uint48            outgoing_ROC; (rollover counter)
    uint64            outgoing_record_index;
    uint64            incoming_record_index;
    uint64            highest_received_record_index;
    uint64            unordered_delivery_highest_received_record_index;
    uint16            replay_protection_window_size;
                      (in byes, at least eight bytes large)
    opaque            replay_window [replay_protection_window_size];
} SecureStreamState;
```

**stream_ID**

The identity of a secure stream. In a secure connection where only one secure stream allowed to setup, the stream_ID for this secure stream is always zero. In the case where a secure connection is bound to a SCTP association and multiple secure stream can be setup within the secure connection, the stream_ID of each secure stream is the stream ID of the SCTP stream that this secure stream is bound to.

**stream_direction**

A secure stream can be bidirectional, outgoing, or incoming.

**outgoing_sequence_number**

Ths outgoing_sequence_number is incremented after each record is sent.

**outgoing_ROC**

The roll-over-counter (ROC) is incremented each time the outgoing_sequence_number wraps around.

**outgoing_record_index**

```
record_index = 2^16 * client_ROC + client_sequence_number;
```

When the record index has wrapped around, either a rekeying should be performed or the secure stream should be terminated. Or a replay attack can be performed after the record index is wrapped around. The record index is 64-bit long so that the lifespan of a master secret is the same as the master secret of the TLS specification.

outgoing_sequence_number, outgoing_ROC, and outgoing_record_index are used only when the stream_direction is bidirectional or outgoing.

**incoming_record_index**

This incoming_record_index is incremented after each record is received. It is only used for reliable streams. In a reliable stream, this state parameter is used to do record authentication.

**highest_received_sequence_number**

The highest sequence number an endpoint has ever received within a certain stream. It is used by unreliable, unordered streams to do replay protection.

**unordered_delivery_highest_received_record_index**

The highest sequence number an endpoint has ever received within the unordered delivery sequence space of a certain stream. It is used by reliable streams with unordered delivery or partial reliable streams with unordered delivery to do the replay protection of unordered delivery records.

**replay_protection_window_size**

The size of the sliding window for doing replay protection.

**replay_window**

The sliding window that is used to do replay protection. It is used only for unreliable, unordered streams or the unordered delivery records in reliable or partial reliable streams.

### 10.5.3.8. Data Unit (DU)

DU, which has been described earlier in section 10.5.1, is the input of the TESP record protocol in the outgoing data path and the output of the TESP record protocol in the incoming data path. A DU is delivered to the TESP through the API from either the upper layer application or the TESP alert protocol. The maximum size of the DU (MDU) is decided by the MDU discovery function of the TESP record protocol.

```
Enum {application _data, unordered_delivery, alert_message } ContentType;
struct {
    uint32       stream_ID;
    ContentType  type;
    uint16       length;
    select (type)
        case application_data:opaque data [length];
        case alert_message         Alert    alert_message;
} DataUnit;
```

**type**

The type of a DU makes it possible to multiplexing both TESP alert messages and application data from the upper layer protocol on one TESP secure connection. This parameter also differentiates normal application data from data that is sent using unordered delivery. The reason for doing so is described in section 10.5.3.16.

**length**

The length of the data or the alert message contained in the DU

### 10.5.3.9. API

The API of the TESP record protocol should be implementation specific and is not specified in this thesis. The upper layer and the TESP alert protocol should be able to deliver outgoing application data and alert messages to a specific secure stream within a specific secure connection and also receive incoming application data and alert messages from a specific secure stream within a specific secure connection through the API. The API should offer callback function to notify the upper layer application the latest update of MDU.

### 10.5.3.10. MDU Discovery Function

The TESP record protocol, like the TLS record protocol, processes data chunk by chunk instead of as a continuous byte stream. If the upper layer application sends continuous byte streams, it

has to break each byte stream into segments before it is passed down to the TESP record protocol. The reason behind this is that data needs to be authenticated in chunks. It is also possible to do data authentication over the entire data stream all at once in the end of the data transmission. However, it is much better if the application can be aware of possible data manipulation during of transmission. The data chunk being process is called a data unit (DU) in TESP.

One principle that was followed to design TESP is that when the underlying transport is unreliable, the application should either receive a complete DU or lose a DU. It should never receive only a fragment of a DU. It is assumed here incomplete DUs are useless and can not be processed.

Under this principle, the security protocol should never fragment a DU. If it does, then to obey the principle of "either receive everything, or lose everything", it has to deal with buffering and reassembling of the fragments of a DU. This complicates the design of the security protocol itself. So basically, whatever the size a DU is, TESP just wraps it into a record and sends it down to the underlying transport layer protocol. However, the size of a DU is limited by the underlying transport protocol. We call the maximum size of a deliverable DU the Maximum Data Unit (MDU). If an upper layer application delivers a DU larger than MDU to TESP, TESP will not able to handle it. It is the responsibility of TESP to discover the value of MDU for the upper layer. The TESP record protocol has a MDU discovery function that derives the MDU for a secure connection according to the information it receives from underlying transport as illustrated in Figure 21.

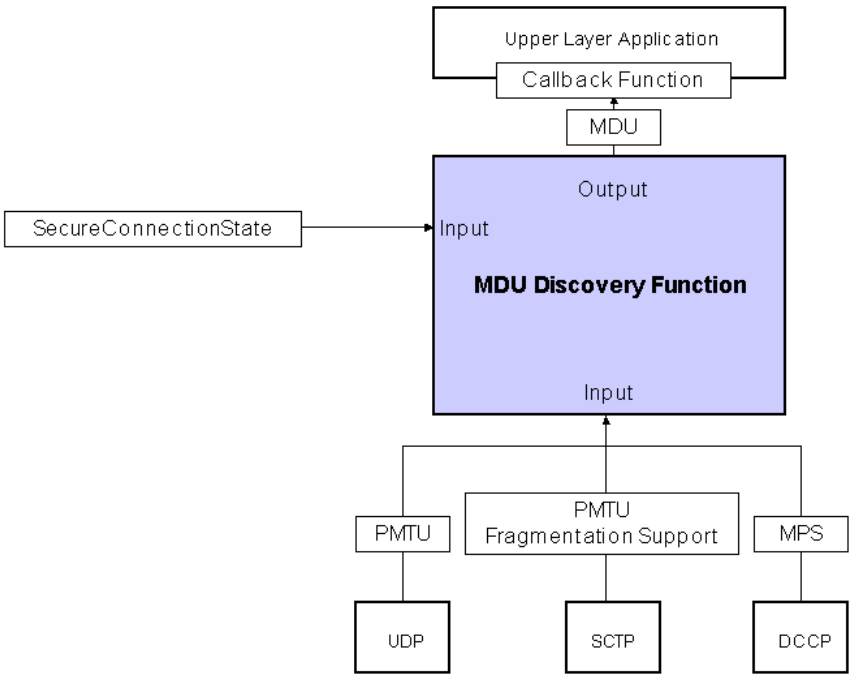



Figure 21. MDU Discovery Function

TCP has its own Path MTU discovery and can perform fragmentation/reassembly on the data records passed down by TESP if they are too large to send without IP layer fragmentation. The maximum size of a TESP data record on top of TCP is only limited by the size of the length field of the record header and, when a block cipher is used, the padding that is appended to the record payload.

```
MDU = 2^16 – (width_of_the_length_field + padding_length + 1 + authentication
_tag_length));
```

TESP needs to know more to calculate MDU when the underlying transport is a UDP session. If

TESP sends a data record that is larger than the PMTU of the transmission path down to UDP, then the IP packet that carries this data record will be fragmented on its way to the destination. This violates the 'IP layer fragmentation is considered harmful' principle. To obey this principle, when the underlying transport layer protocol is UDP, the TESP record protocol is responsible for the discovery of the PMTU. It then limits the size of DU according to the Path MTU. If the application wants to send a DU larger than MDU, TESP should return a failure to the application with MDU information.

```
MDU = PMTU – IP_header_length – UDP_header_length – (the length of the TESP
record header + padding_length + 1 + authentication _tag_length);
```

SCTP is a little bit more complex. An implementation can optionally support message fragmentation. If message fragmentation is supported by the underlying SCTP implementation, the size of a TESP data record on top of SCTP is only limited by the length field of the sctp_send( ) function of the SCTP API. Otherwise, if message fragmentation is not supported by the underlying SCTP implementation, the size of MDU is limited the PMTU which is maintained by SCTP. To summarize, if the underlying transport is SCTP, TESP is responsible for querying SCTP for information about the support of fragmentation and PMTU if fragmentation is not supported. TESP then uses this information to derive the MDU.

The DCCP protocol is a more intelligent datagram transport layer protocol than UDP. It implements Path MTU discovery and its own congestion control mechanisms. It maintains the maximum packet size (MPS), which is influenced by the PMTU, the maximum packet size allowed by the current congestion control mechanism, and the lengths of the IP and DCCP headers. If the underlying transport layer protocol is DCCP, the TESP should be able to query for the value of MPS and use it to derive the limit on the size of a DU that can be sent by the application.

```
MDU = MPS – ( the length of the TESP record header + padding_length + 1 +
authentication _tag_length);
```

### 10.5.3.11. Compression Function

The compression method used by the compression function is not yet defined. This is left as future work. The compression should take data units as input, compress them and output them in the format of a compressed data unit, as illustrated in Figure 22.

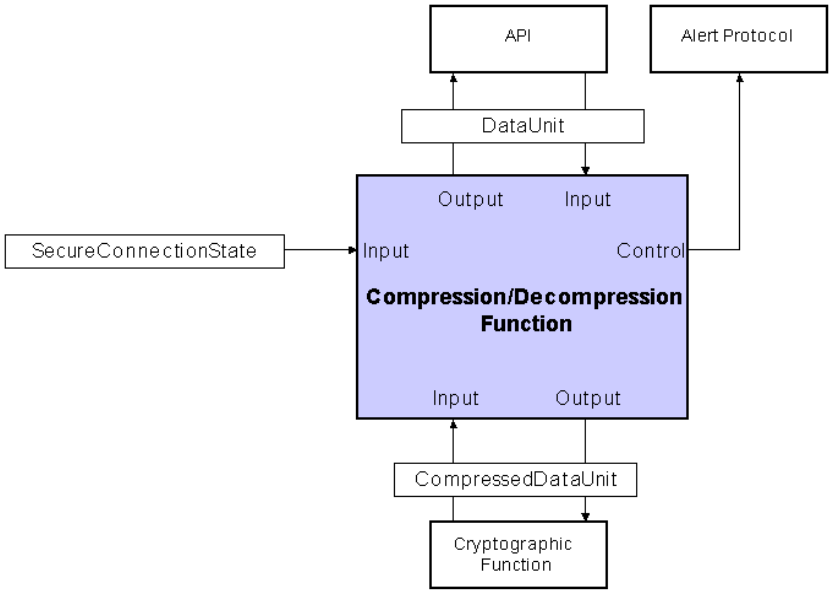



Figure 22. Compression Function

Data Unit (DU), which has been described earlier in section 10.5.1, is the input of the compression function in the outgoing data path and the output of the compression in the incoming data path. A DU is delivered to the TESP through the API from either the upper layer application or the TESP alert protocol. The maximum size of the DU (MDU) is decided by the MDU discovery function of the TESP record protocol.

```
struct {
    ContentType type;
    uint16 compressed_data_length;
    opaque compressed_data [length];
} CompressedDataUnit
```

### 10.5.3.12. Key Derivation Function

The derivation of bulk cipher state parameters and MAC function state parameters from the secure connection state parameters is done by the key derivation function of the TESP record protocol.

As illustrated in Figure 23, the TESP key derivation function takes parameters of the secure connection state of a secure connection and the stream ID of the secure stream of the DU and derives CryptoTransformParam that contains all the parameters needed by the cryptographic transformation function. The TESP key derivation function is to derive separate session keys for multiple secure streams within a secure connection from a single master secret. This feature is designed to support the multi-streaming feature of SCTP. Thus two stations can do peer-to-peer connection setup of a SCTP association via a session initiation protocol such as SIP and exchange security parameters in parallel. The two communicating parties could open multiple SCTP streams under this SCTP association. It is possible that the number of streams that is going to be opened is not predictable. Consider web page transfer over SCTP as an example. The client may open multiple streams to exchange the images within the web page. However, the number of streams the client will open depends on the number of images contained in the web page. The client only knows this number after the client receives the whole HTML file from the server. But during the phase of session setup, this information is not available. In this case, it would be convenient for the key management protocol to exchange security parameters for the SCTP association instead of for each stream within the association. The TESP record protocol then relies on the key derivation function to derive separate session keys for each SCTP stream.

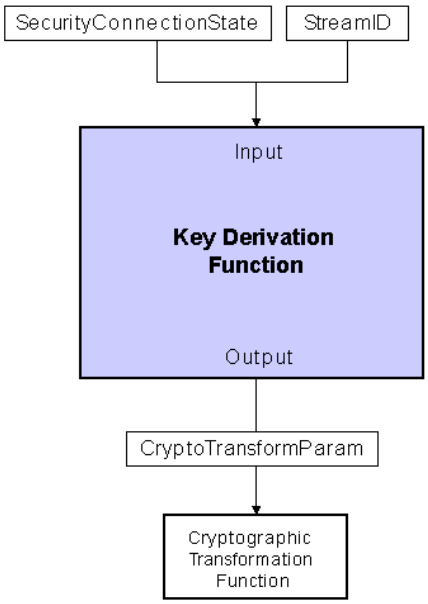



Figure 23. Key Derivation Function

When a data unit is being transformed, the key derivation function should decide which function to use (the general key derivation function or the SCTP key derivation) according to:

```
r = packet_index DIV key_derivation_rate;
key_block = PRF (SecurityParameters.master_secret, “key expansion”,
SecurityParameters.server_random + SecurityParameters.client_random + r +
secure_stream_ID);
```

The key_block is partitioned as follows:

```
entity-A_write_MAC_secret [mac_function.hash_size];
entity-A_write_key [bulk_cipher.key_size];
entity-B_write_MAC_secret [mac_function.hash_size];
entity-B_write_key [SecurityParamters.bulk_cipher.key_size];
```

If the connection type is SCTP, stream_ID is the SCTP stream ID of the SCTP stream that is used to send the data unit being transformed. If the connection type is otherwise, the stream_ID should be set to zero. The derived entity-B_write_key and the entity-B_write_MAC_secret are used as the entity-A_read_key and the entity-A_ read_MAC_secret, and vice versa.

```
struct {
    BulkCipher   bulk_cipher;
    MACFunction  mac_function;
    uint64       outgoing_record_index;
    uint64       incoming_record_index;
    uint8            MKI_size;
    opaque       MKI [MKI_size];
    opaque       entity-A_write_MAC_secret [mac_function.hash_size];
    opaque       entity-A_write_key [bulk_cipher.key_size];
    opaque       entity-B_write_MAC_secret [mac_function.hash_size];
    opaque       entity-B_write_key
[SecurityParamters.bulk_cipher.key_size];
} CryptoTransformParam
```

### 10.5.3.13. Cryptographic Transformation Function

The TESP cryptographic transformation function performs encryption and message authentication tag derivation on outgoing compressed data units. This function also performs decryption on incoming cryptographically transformed data units.

As illustrated in Figure 24, a CompressedDataUnit is the input of the cryptographic transformation function in the outgoing data path and the output of the cryptographic transformation in the incoming data path. CryptoTransformedDataUnit is the output of the cryptographic transformation function in the outgoing data path and the input of the cryptographic transformation in the incoming data path.

```
struct {
     opaque IV [BulkCipher.block_length];
     opaque data [CompressedDataUnit.length];
     opaque MAC [MACFunction.hash_size];
} StreamEncryptedData;
struct {
     opaque IV [BulkCipher.block_length];
     opaque data [CompressedDataUnit.length + padding_length + 1];
     opaque MAC [MACFunction.hash_size];
} BlockEncryptedData;
struct {
     ContentType type;
     uint16 length;
     select (BulkCipher.cipher_type)
          case stream: StreamEncryptedData data;
          case block:  BlockEncryptedData data;
} CryptoTransformedDataUnit;
```

IV generation is already described in section 10.5.3.3. The padding for block ciphers should follow the rule of padding described in TLS.

The MAC on the client side is generated as follows:

```
HMAC_hash(client_write_MAC_secret,type,packet_index,
CryptoTransformedDataUnit.data, MKI);
```

The MAC on the server side is generated as follows:

```
HMAC_hash(server_write_MAC_secret,type,packet_index,
CryptoTransformedDataUnit.data, MKI);
```

The following are the parameters that are needed for the record protocol to do the cryptographic transformation. Some of the parameters such as bulk_cipher and hash_function are directly taken from the session state. Some of them are derived from the key derivation function.

```
struct {
     BulkCipher   bulk_cipher;
     Opaque       client_write_key [bulk_cipher.key_size];
     Opaque       server_write_key [bulk_cipher.key_size];
     HashFunction mac_function;
     Opaque       client_write_MAC_secret [mac_function.hash_size];
     Opaque       server_write_MAC_secret [mac_function.hash_size];
} CryptographicTransformationParameters
```

TESP encrypts traffic of each direction with different keys just as TLS does. The reasoning would be that it is possible that a certain direction of the traffic is easier to do cryptanalysis than the other direction. These cryptographic transformation parameters are derived by the key derivation function each time a data unit is to be sent out or received.

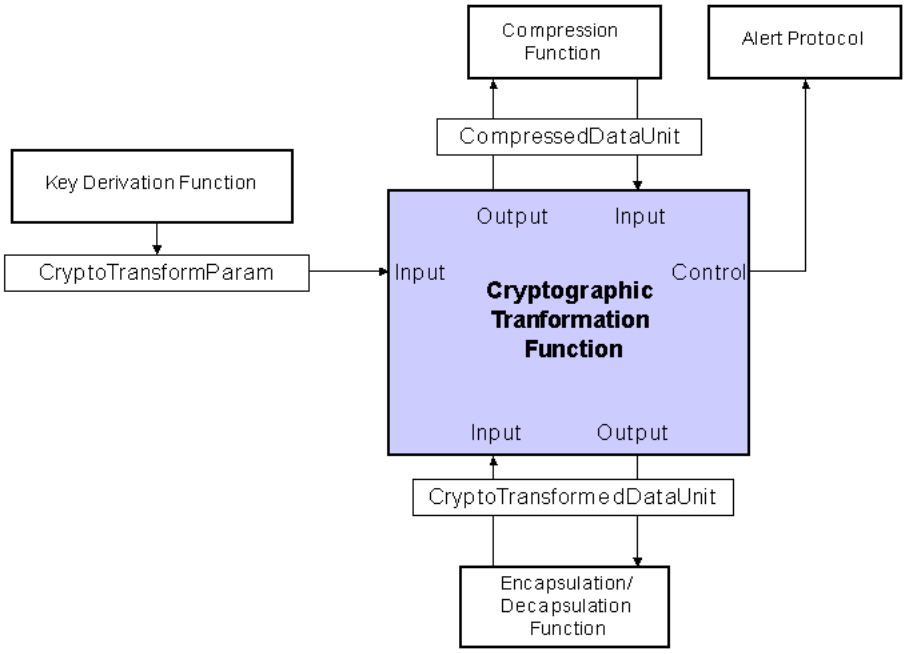



Figure 24. Cryptographic Transformation Function

**Encryption first or Authentication first**

There are two ways of performing data encryption and authentication together. The first way is to authenticate data in plaintext first, derive the authentication tag, and then encrypt both the data and the authentication tag. The second way is to encrypt data first, then perform authentication over the encrypted data and derive the authentication tag. It is not safe to authenticate data in plaintext, and then encrypt the data without encrypt the authentication tag because in this way, it is possible for an attacker to perform known-plaintext pre-computation attack on the data authentication function.

In SRTP, data authentication is done over the encrypted RTP payload after encryption. The authentication tag is not encrypted.

In TLS, data authentication is done over the plaintext before encryption. The authentication tag is encrypted. The problem of this design is that the receiving endpoint could only authenticate a data record after performing a decryption of it. Additionally, an attacker can easily waste an endpoint's resource by just sending crap data records.

In TESP, data authentication is done over encrypted data after encryption following the design of SRTP. This is because when a record is received, a record authentication check should be performed before the record is decrypted.

### 10.5.3.14. Encapsulation/Decapsulation Function

The TESP encapsulation/decapsulation function, as illustrated in Figure 25, encapsulates CryptoTransformedDataUnit into a TESPRecord in the outgoing data path and decapsulates a TESPRecord into a CryptoTransformedDataUnit in the incoming data path. The encapsulation format is adaptive according to the type of the underlying transport session to reduce overhead, optimizing bandwidth utilization.

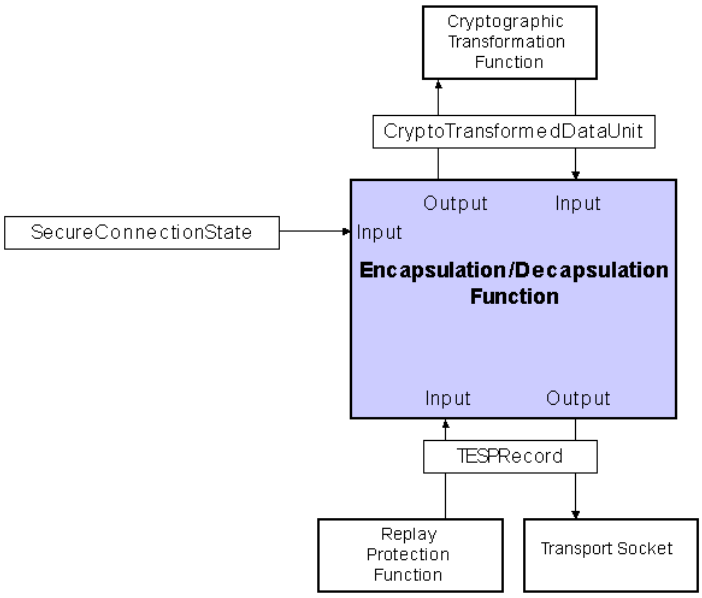



Figure 25. Encapsulation/Decapsulation Function

**Record Format**

The format of a TESPRecord, was illustrated in Figure 26. It varies according to the type of the underlying transport layer session. A TCPRecord is the encapsulation format for all types of DU in a secure connection over a TCP connection. An SCTPRecord is the encapsulation format for DUs that carry alert messages or application data in a secure connection over a SCTP association. URUORecord (URUO refer to 'Unreliable, Unordered') is the encapsulation format for all types of DU in a secure connection over a UDP session or a DCCP session. It is also the encapsulation format for DUs that carry unordered delivery application in a secure connection over a SCTP association. All these formats contain at least the ContentType field. TCPRecord contains the length field to carry information about the length of a record because TCP connections are byte streams. URUORecord contains the sequence_number field because the sequence number of a record should be explicit if the underlying transport session is unreliable or unordered. The MKI field is only needed when rekeying is enabled on and MKI is used to synchronize rekeying of a secure connection.

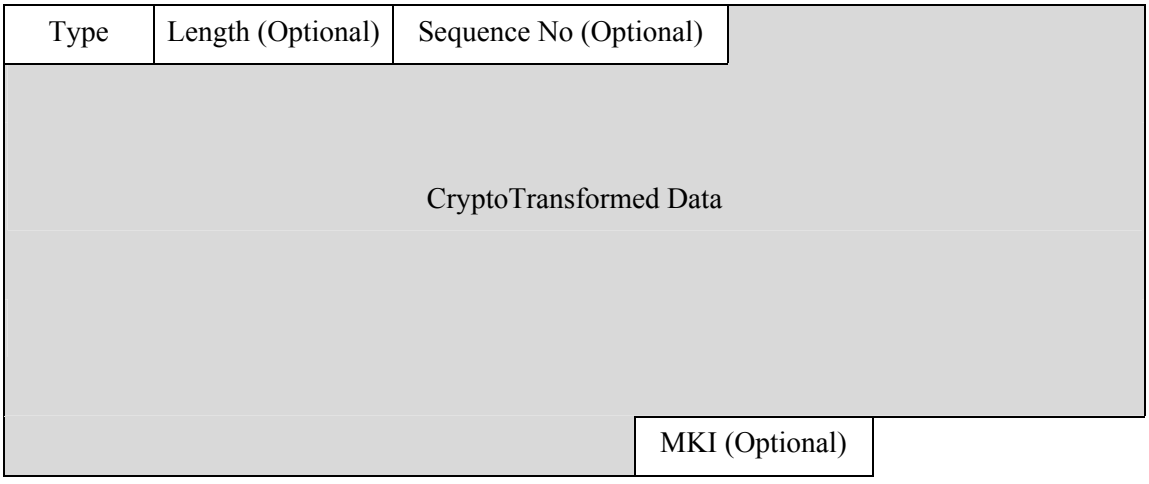



Figure 26. Record Encapsulation Format

```
Struct {
```

```
    Select (SecureConnectionState.security_parameters.transport_type)
        case TCP:                          TCPRecord;
        case SCTP:
            select(DU.type)
                case unordered_delivery:   URUORecord (unreliable,
            unodered record);
                case application _data:
                case alert_message:        SCTPRecord
        case UDP:
        case PR-SCTP:
        case DCCP:                    URUORecord;  (unreliable,  unodered
    record);
} TESPRecord;
struct {
    ContentType   type;
    uint16        length;
    opaque        crypto_transformed_data
[CryptoTransformedDataUnit.length];
    opaque        MKI [MasterKeyInfo.MKI_size]; (optional)
} TCPRecord;
struct {
    ContentType   type;
    opaque        crypto_transformed_data
[CryptoTransformedDataUnit.length];
    opaque        MKI [MasterKeyInfo.MKI_size]; (optional)
} SCTPRecord;
struct {
    ContentType   type;
    uint16        sequence_number;
    opaque        crypto_transformed_data
[CryptoTransformedDataUnit.length];
    opaque        MKI [MasterKeyInfo.MKI_size]; (optional)
} URUORecord;
```

### 10.5.3.15. Record Authentication Function

When TESP receives a record, a record authentication, as illustrated in Figure 27, must first be performed to make sure that the content of the record is not manipulated by an attacker before any further processing could be done. The default MAC function of TESP is HMAC SHA-1.

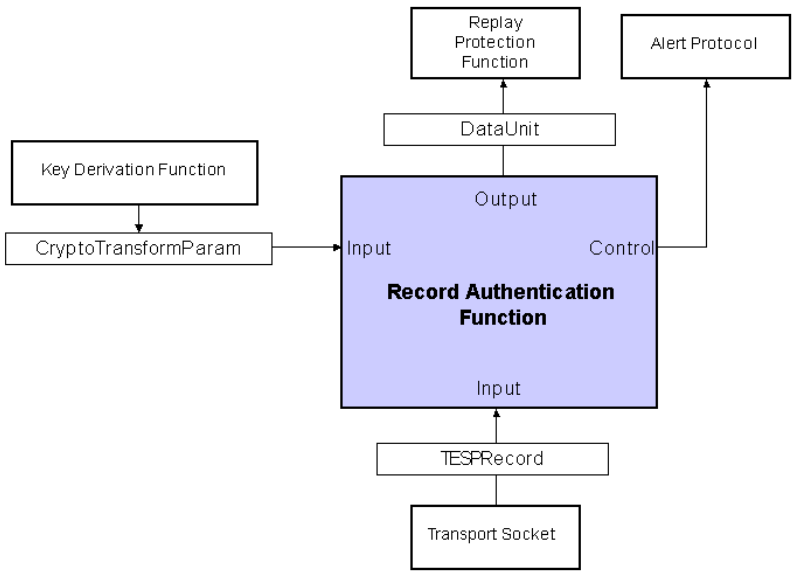



Figure 27. Replay Protection Function

Unlike TLS, in which the MAC is encrypted and record authentication is performed **after** the received record is decrypted, TESP performs record authentication **before** a record is decrypted. The reason is that TESP supports weak transport sessions such as UDP sessions, where even a blind attacker can easily perform a resource attack by sending a large amount of records with arbitrary content if decryption must be done before authentication.

The record authentication function performs the following camputation:

```
HMAC_hash (client_write_MAC_secret, type, packet_index,
CryptoTransformedDataUnit.data, MKI)
```

This computation is compared with the authentication tag of the received data record. If the underlying transport session is a TCP connection or a SCTP association, the packet index is the incoming_record_index of the SecureStreamState for the stream the received record belongs to. If the underlying transport session is a UDP session, a PR-SCTP association, or a DCCP session, the packet_index is the derived from the value of the sequence number field of the received record header and the incoming ROC of the SecureStreamState for the stream the received record belongs to. A record that fails to pass data authentication, it should be dropped and an alert message should be sent to the other endpoint.

**Optional Record Authentication Check**

The UDP checksum is optional. DCCP supports checksum coverage selection as a feature. This is to support some applications that can tolerate data corruption and would like to trade the correctness of data against the improvement in performance over noisy links. If record authentication is performed, these features will provide no benefit to the upper layer protocols. However, if the record authentication check is not performed for a secure connection, an attacker can easily perform false data injection attacks, data manipulation, and replay attacks.

In TESP, for secure connections bound to DCCP and UDP sessions, one can choose record authentication coverage. This coverage can *optionally* include application data.

### 10.5.3.16. Replay Protection Function

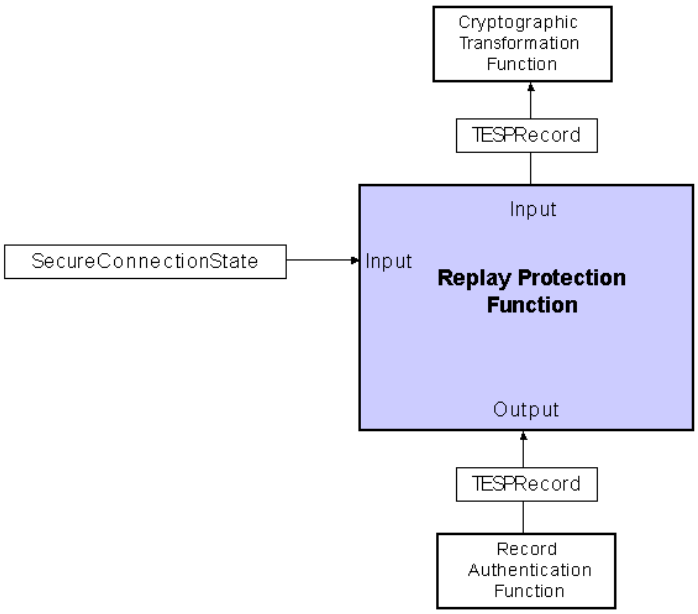



Figure 28. Replay Protection Function

Replay protection is done within each secure stream and, as stated earlier, depends on the type of the underlying transport session.

**TCP Connections**

For TCP connections, since each data record is guaranteed to arrive in the same order as it is sent into the network, replay protection is already done by the record authentication function as the implicit record index is covered by the data authentication.

**UDP Sessions, DCCP Sessions**

When the replay protection function is used to protect a UDP session or a DCCP session, since the data records can arrive in arbitrary order, a replay list is used to protect the session against replay attack. The replay list of TESP uses the same sliding window approach as described in the SRTP specification. The size of the sliding window is a receiver-side parameter that is implementation specific. As with SRTP, only data records with an index ahead of the window, or, inside the window but not yet received are accepted. Other packets are seen as replayed packets and are dropped. It is possible that in extreme cases where packet delivery is highly unordered late packets could be treated as replayed packets. For example, if the size of the replay list sliding window is 32, and the sequence of the received data records are as follows:

1, 2, 3, 4, 50, 6, 7,…, 48, 49, 51,…

Then after the data record with packet index number 50 is received and authenticated, the data records with packet index number ranging from 6 to 18 are regarded as replayed packets and will be thrown away. However, since the application that uses unreliable transport should be able to handle data loss, as long as this scenario does not happen a lot, this kind of misjudgment will not hurt the performance of the application severely.

An alert message should be sent to the sender when a replay happens. It is the application that decides if the secure connection should be closed or not. Unlike the case for TCP connections or SCTP reliable, ordered streams where when a data replay is detected, the secure connection should be closed because of possible data loss in the middle of a connection, in the case of UDP or DCCP sessions, as long as replayed records are detected and dropped, the transmission could still continue without any effect. This is to prevent a DoS attack by an attacker sending replayed records if a secure connection is forced to be terminated upon receiving replayed any record.

**SCTP Associations**

The replay protection for secure connections over SCTP Associations is more complex than the case for TCP connections because of the unordered delivery feature of SCTP.

In the SCTP specification, it is stated that when an end point receives a data chunk marked as unordered delivery, it must bypass all the data chunks in front of it in the receiving queue and be delivered to the upper layer. It is also recommended that an unordered data chunk should be put at the head of the outbound transmission queue if possible at the sending end. In this case, a data record marked as unordered is very likely to arrive at the receiving end out of the order of the record index.

The scenario for packet arrival in this case is similar to the packet arrival of UDP and DCCP sessions, i.e., possibly unordered arrival. However, since in a SCTP association the transport of unordered delivered data records is still reliable, the sliding window approach that is used in UDP and DCCP to prevent replay attack cannot be applied to SCTP streams. With the sliding window approach, arriving records that are out of the range of the sliding window's coverage are dropped because there is no way of telling if they are replay records or not. Additionally, it is very likely that unordered delivered records can be very far out of sequence in some extreme environments thus resulting in later received records being dropped. An example of this is as follows:

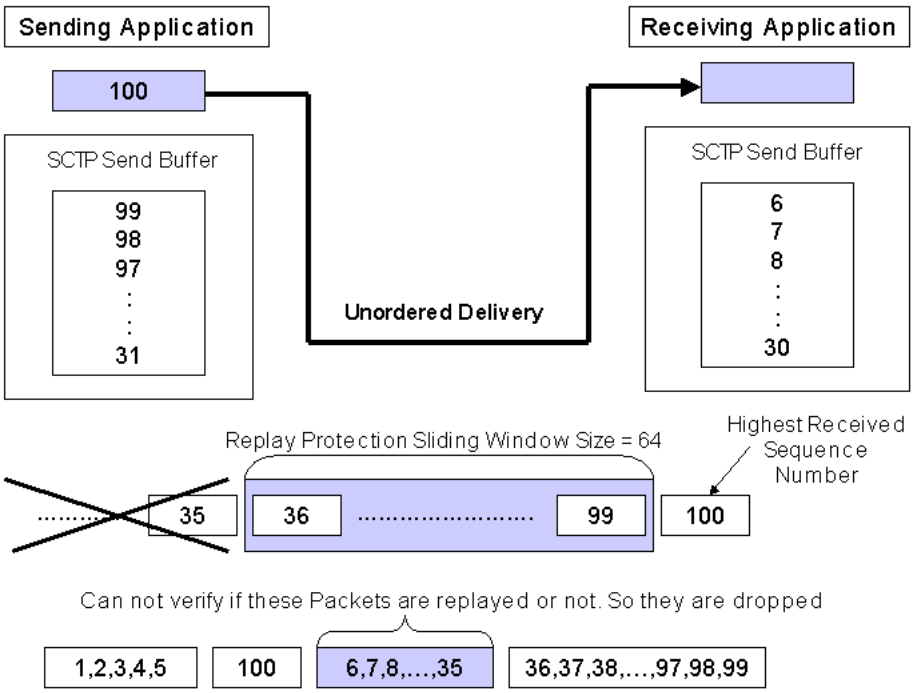



Figure 29 SCTP unordered delivery with the sliding window replay protection mechanism

At a certain moment, the transmission queue of the SCTP stack in the sending end holds data records with the following record indexes, as illustrated in Figure 29:

31, 32, 33, …97, 98, 99

The receiving queue of the SCTP stack in the receiving end holds data records with the following record index:

6, 7, 8,…, 28, 29, 30

At the receiver the size of the sliding window to 64. If, the application at the sender sends a data record with record sequence number 100 marked as unordered, it will bypass the data records that are still in the transmission queue (31 to 99) and be sent into the network and when received by the receiver it again bypasses all the data records that are still in the receiving queue (6 to 30) and will be received by TESP. If TESP used a sliding window approach, it would verify the data integrity of data record number 100 and then set the largest received record number to 100, since the size of the sliding window size is only 64, data records from 6 to 35 will be dropped.

A solution to this issue is as follows: within a secure stream, separate the ordered delivered records and the unordered delivered in two packet sequence spaces, then perform replay protection separately in these two different spaces. The separation of sequence space is done by having one additional record type for unordered delivered records and additional state parameters for the replay protection for unordered delivered records.

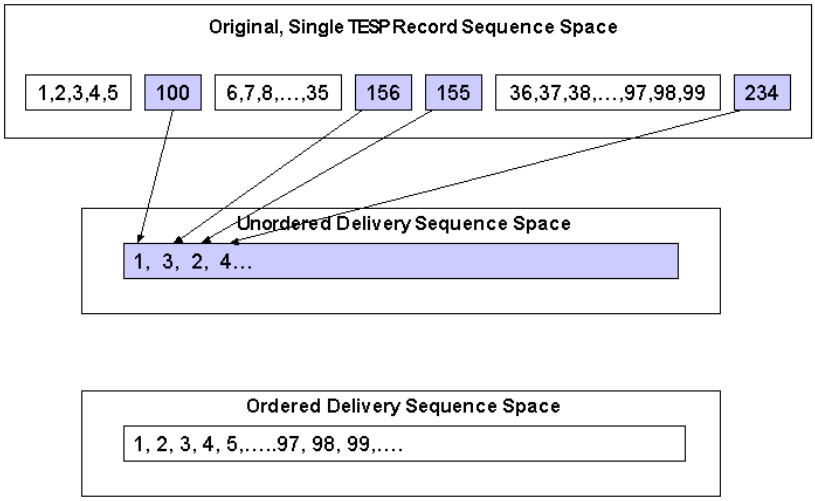



Figure 30 Separate Sequence Space for Unordered Delivered Records

In the ordered delivery sequence, record arrival is reliable and ordered. Thus the method for doing replay protection is the same as for a TCP connection.

In the unordered delivery sequence, record arrival is reliable. However, it is likely, but not necessary, that the unordered records arrive at the same ordered as their sequence number. To perform replay protection over unordered delivered records, a parameter that contains the value of the largest received packet index should be maintained. There should also be a link list of all the gaps in received records that have not been received. These states can also be used to detect record dropping as described in section 10.5.3.17.

**PR-SCTP Associations**

In PR-SCTP Associations, replay protection should also be done in two separate sequence spaces. One for ordered delivery, one for unordered delivery.

For ordered delivered records, data records arrive in the correct order. However, some data records might be lost during transmission. Replay protection has to check if the record index of the received data record is larger than the record index of the previously received data record. If any data record arrives out of order, then an alert message should be sent. For unordered delivered record, the sliding window approach could be used without a problem because the data delivery is partially reliable and record drop is tolerable.

## 10.5.3.17. Record Dropping Detection for Reliable Transport Sessions

An attacker that is sitting in the middle of the path between the communicating endpoints can, not only replay records, but could also cause drop records. There is no need for the security connection running on top of unreliable transport layer sessions such as UDP sessions, DCCP sessions, and PR-SCTP associations to care about the dropping of data because the service provided by this kind of transport session to the upper layer protocols is intrinsically unreliable and data loss is tolerable. Even if there is no attacker dropping packets deliberately, packets are like to be dropped by an overloaded router from time to time. It is not possible for a security protocol to tell if a packet is dropped by an evil attacker or by a busy router when the service provided to the security protocol by the underlying transport session is unreliable.

But for secure connections running on top of reliable transport sessions such as TCP connections and SCTP association, since the service provided by the underlying transport sessions is anticipated to be reliable, it is the responsibility of the security protocol running over

them to detect dropping of data if there is any and to give warnings to the upper layer protocols when it occurs.

One might argue that now that the data delivery service provided a transport session is reliable, how could it be possible that an attacker in the middle can drop data? The reliable transport layer protocol such as TCP will retransmit data anyway. The fact is that, if the security protocol is running on top of transport layer protocol, there is no integrity protection of the transport layer protocol header. An attacker sitting in the middle of the path of a transport session can easily do a 'proxy in the middle attack', in which an attacker manipulates the sequence number and acknowledgement number so that even if a segment of data is lost, the transport layer protocol stacks of the endpoints still think that all the data was reliably delivered.

It is possible to prevent an attacker from manipulating the session state variables carried by the transport layer headers by moving the security down 'into' the transport layer or even further downward into the IP layer. However, the compatibility with intermediary-based transport services and performance optimization mechanisms will be lost (as stated earlier in the introductory section of this thesis). However, a security protocol running on top of a reliable transport session can detect dropping of data if there is any.

For secure connections running on top of TCP connections, packet dropping can easily be detected by the discontinuity of record index number.

For secure connections running on top of SCTP associations, the same 'record index number discontinuity' detection can be used for the ordered SCTP message index space. For unordered delivered messages, there is no way to tell if a message is dropped or late until the end of transmission of a certain secure stream. In TESP, before each secure stream is terminated, a closure alert message from the alert protocol must be sent in either direction (depending on if the secure stream is a unidirectional one or a bidirectional one). This alert message is to prevent the truncation of a secure stream. In each closure alert message for secure streams in a secure connection over a SCTP association, there is a parameter called 'the number of unordered delivery records sent'. The receiving endpoint can then use this parameter in the closure alert message to check if there is any missing unordered record. If one or more records missing, a TESP implementation will give a warning to the upper layer.

#### 10.5.3.18. Key Refreshment

Similar to SRTP, key refreshment is implicitly performed within the key derivation function and synchronized by the key derivation rate.

#### 10.5.3.19. Rekeying

Similar to SRTP, rekeying is done by manually replacing the master secret in the secure connection state with a new one. Rekeying is synchronized by the MKI field of a TESP record or by the <from, to> information of a master key.

### 10.5.4. The TESP Alert Protocol

The TESP alert protocol is responsible for sending alert messages indicating important events and errors. The sending of these alert messages can be initiated by the function blocks in the record protocol.

```
enum {warning, fatal} AlertLevel;
enum {
     close_notify (0);
     bad_record_mad (1);
     decryption_failed (21);
     record_overflow(22);
     decompression_failure (30);
     internal_error (80);
} AlertDescription
```

```
struct {
    AlertLevel level;
    AlertDescription description;
} Alert;
```

## 10.6. Security Analysis of TESP

Before doing a security analysis of TESP, the potential attackers were analyzed and categorized based on their relative location to the endpoints that are under attack, in order to show which attacks are the most dangerous ones. Attackers sitting in different locations have different levels of capability to launch a given type of attack. Attacks that can easily be launched by even blind attackers are of the biggest concern.

After the analysis of attackers, different areas of security were considered. First, the confidentiality of data protected by TESP was analyzed. Possible attacks against confidentiality and the immunity of TESP from these attacks were studied. Secondly, the confidentiality of other information (besides data) that could reveal information to help attackers to perform crypto analysis was studied. The level of data integrity was also examined. Additionally, the protection provided by TESP against other possible attacks such as replay attacks and packet dropping attacks were considered.

Although it is not TESP's responsibility, the vulnerabilities of the underlying transport sessions are described to remind the reader of the compromise of transport session security when moving the implementation of security from the network layer up to the transport layer.

### 10.6.1. Analysis of Attackers

An Attacker is an entity that is trying to break the security of the communication between two endpoints. Attackers can be categorized into the following kinds:

- Blind attackers: this kind of attacker represents the majority of attackers. The only information known by attackers of this kind about the endpoints under attack are IP addresses and sometimes the port information of the communicating endpoints that are under attack. The most common blind attacks are the denial-of-service attacks.
- Attackers sitting in the middle: attackers of this kind usually control at least one of the routers or switches on the path between the communicating endpoints that are under attack. These attackers can access any transport session running between the endpoints that are under attack and they are able to analyze, drop, or manipulate the packets that belong to any transport session between the endpoints.
- Attackers with access to the secure connection under attack: attackers of this kind have some control over at least one of the communication endpoints that are under attack and can get hold of a secure connection for a period. They can send arbitrary plaintext over a secure connection and can easily perform chosen plaintext attacks over the secure connection.

### 10.6.2. Confidentiality of Data

An attacker sitting in the middle can launch the following attacks on TESP protected sessions by only passively eavesdropping the traffic:

- Traffic analysis: when a block cipher is used for encryption, TESP can support random-length padding to hide the occurrence, the frequency, and even the volume of traffic flow from an attacker doing traffic analysis.
- Brute force attack: the difficulty of launching a brute force attack depends directly on the strength (i.e. key length) of the ciphers. The default ciphers provided by TESP are assumed to have a key length that is long enough to keep the data confidential for a few

months from even governmental intelligence agencies with around $300M budget for buying machines to do brute-force attacks [63]. The strength of other ciphers that are going to be added to TESP should be carefully studied.

- Time-memory tradeoff attack: the session key derivation function, the data authentication function, and the default cipher are salted with large-enough random numbers. This should protect TESP secure connections against time-memory tradeoff attacks [65], [66].
- Occurrence of Two-time-pad: when TESP is used, before the record index of any secure stream within a secure connection wraps, the master secret must be renewed to avoid the occurrence of a two-time-pad when a stream cipher is used for encryption.

### Chosen Plaintext Attack

One possible active attack that can be launched upon WTLS and TLS to compromise data confidentiality is a chosen plaintext attack on CBC-based ciphersuites when the attacker controls a secure connection. TESP is immune from the chosen plaintext attack because the IV of a record is not predictable.

### 10.6.3. Confidentiality of Other Information

Record Type: as the record type is not encrypted, thus it is information available to attackers (for example revealing the traffic pattern of unordered delivery in a secure connection bound to a SCTP association).

Alert Messages: the presence of alert message and the length of alert messages is information available to attackers

MKI: although covered by data authenticated, it is sent in plaintext. This reveals nothing about the master secret itself. However, it does reveal the occurrence of rekeying, which can be useful information for a crypto analysis, to an eavesdropper.

Padding: TESP does not have the problems related to padding with the CBC-based ciphersuites in TLS mentioned in [25]. The data authentication of TESP covers the padding and padding length of a record when a block cipher is used for encryption. It is not possible for a record with the padding and the padding length field manipulated by an attacker to pass the record authentication of the receiving end. Additionally, at the receiving endpoint, unlike in TLS where decryption is done before data authentication is performed, decryption is only done if a record first passes the record authentication check. This makes Vaudenay's attack [25] impossible to perform. The difficulty of launching a brute force attack also depends on the availability of the known plaintext. In TLS and WTLS, the padding is filled with the padding length value. Therefore, even if the data transmitted looks totally random, an attacker doing brute-force attack can still search for the pattern of padding as a hint of plaintext recognition.

### 10.6.4. Data Integrity

MD5 is not considered a secure hash function any longer, thus it is not used in TESP. The data integrity of a secure connection relies on the security of SHA-1.

### 10.6.5. Replay Attack

The replay function of TESP should be able to detect replay attacks on secure connections on top of TCP connections, SCTP association, UDP sessions, and DCCP sessions.

### 10.6.6. Packet Dropping Attack

TESP provides mechanisms to protect a secure connection from data dropping or data truncation attacks.

For a secure connection on top of a SCTP association, it is possible to have records dropped by

an attacker in the middle. But at the end of a stream, the receiving endpoint of the secure stream will know if any record is missing. However, this information may be too late to be useful.

### 10.6.7. The Vulnerability of Transport Sessions

TESP is a security protocol running on top of transport layer protocols. A security protocol can provide the confidentiality and authenticity of the data transmitted over the underlying transport session. It can also detect fraudulent data, replayed data, and dropping and truncation of data *provided* the underlying transport is a reliable one. However, it is not possible for a security protocol to protect the underlying transport session against any attack on the state of a transport session. The best thing a security protocol can do is to detect the abnormal termination of a transport session so that the upper layer protocols are aware of a DoS attack when it occurs. If a transport session is compromised and the data delivery becomes unavailable, the secure connection running on top of the session is also interrupted.

In this thesis, the security analysis focuses on a peer-to-peer traffic model. The vulnerability of transport layer protocols that are related to client-server traffic model such as TCP's vulnerability to SYN flood attacks are not considered.

**Security Analysis of TCP Connections**

An attacker sitting in the middle of the path of the communication, he can easily perform DoS attacks by simply dropping packets. An attacker of this kind can also setup a proxy that can manipulate state parameters in TCP headers and insert some fraudulent data, truncate data, or drop data in the middle.

For a blind attacker, performing an attack on a transport session much more difficult, but still possible. In [67], two kinds of blind reset attacks and a blind data injection attack were described. Solutions are provided by the same document, but a change of the TCP protocol should be made. Adding an MD5 signature option to TCP headers [68] could also be a solution. But this solution has problems (such as breaking the compatibility with intermediary-based transport services and performance optimization mechanisms). The limited TCP header length causes this option to very likely be unable to fit into the TCP header if other TCP options are used.

**Security Analysis of SCTP Associations**

For an attacker sitting in the middle of the communication path, DoS attack can easily be performed by simply dropping packets. An attacker of this kind can also setup a proxy that can manipulate state parameters in SCTP headers and insert some fraudulent data, truncate data, or drop data in the middle. SCTP is also vulnerable to denial of service attack launched by a blind attacker when the dynamic address reconfiguration SCTP extension is used [69].

**Security Analysis of UDP Sessions**

An attacker sitting in the middle can easily perform all kinds of DoS attacks on UDP sessions. He can also insert false datagrams or replay datagrams. Truncation of data or dropping of some datagrams done by an attacker sitting in the middle will not fatally damage the communication because a UDP session is intrinsically providing a unreliable service and the upper layer that uses this service should be able to deal with loss of datagrams.

A blind attacker can also easily insert false datagrams into a UDP session. However since there is no state for a UDP session, it is impossible for a blind attacker to perform any DoS attack over a UDP session.

**Security Analysis of DCCP Sessions**

An attacker sitting in the middle can easily perform all kinds of DoS attacks on DCCP sessions. He can also insert false datagrams or replay datagrams. Truncation of data or dropping of some datagrams done by an attacker sitting in the middle will not fatally damage the communication because a DCCP session provides an intrinsically unreliable service and the upper layer

protocols that use this service should be able to handle loss of datagrams.

It is more difficult for a blind attacker to insert false data into a DCCP session because of the sequence window feature that only allows datagrams with a valid sequence number.

However, since DCCP is still under development, there is currently no research on how easy is it for a blind attacker to hijack a DCCP session (to perform a blind reset attack or a blind data injection attack).

**Conclusion**

When security is done on top of the transport layer protocol, an attacker sitting in the middle can easily break a transport session by performing DoS attacks. However, this kind attack can also occur if network layer security such as IPSec is used for security, because the attacker can simply drop packets if he controls one of the links in the communication path.

It is also possible for an attacker sitting in the middle to spoof the communicating endpoints by manipulating state parameters carried by the transport layer headers and replay, insert, drop, or truncate a segment of data. This kind of attack could be stopped if the security protocol is in the transport layer or in the network layer, however this sacrifices compatibility with intermediary services. This kind of attack, although cannot be stopped, can also be passively detected by a properly designed security protocol running on top of the transport layer.

When the underlying transport is reliable, TESP can protect the integrity of application data (i.e. by passively detecting the occurrence of the replay, insertion, dropping, and truncation of data done by any attacker). When the underlying transport is unreliable, the best TESP can do is to passively detect the occurrence of replay and insertion of data done by an attacker. Data dropping or truncation attacks cannot be differentiated from packet loss caused by the network, but these kinds of attacks are considered relatively harmless for unreliable transport sessions.

## 10.7. TESP-MIKEY Relationship

For a security protocol such as TESP to function, a key management protocol must be executed prior to the exchange of the master secret and other necessary parameters for the crypto sessions. These parameters are altogether called a security policy. After key management is done, the master secret and the security policy are correctly exchanged. It is then said that the communicating endpoints have setup a ‘security association’ and communication endpoints can start sending data over the crypto sessions protected by the security protocol. One of the tasks of this thesis was to define an interface between the new security protocol designed in this thesis and external key management protocols. The interface of TESP to external key management protocols is the security parameters and has already been specified in section 10.5.3.5. MIKEY is a promising solution for key management of peer-to-peer communication using session setup protocols such as SIP. MIKEY is designed to have the extensibility to support multiple security protocols. An interface of MIKEY to SRTP has been included as part of the MIKEY specification. In this section, the interface to TESP of MIKEY is demonstrated.

Since the terminology used in TESP and MIKEY is not consistent, a table mapping this terminology is given in Table 2.

Table 2 Mapping of Terminology between MIKEY, SRTP, and TESP

| MIKEY | TESP | SRTP |
|---|---|---|
| Crypto Session | Secure Connection | SRTP stream |
| Secure Association | Security Parameters | Input to SRTP's crypto context |
| TEK | Master Secret | SRTP Master key |

**New Security Policy for TESP**

The parameters that need to be exchanged by the key management protocol for a security association (i.e. the security policy) vary from one security protocol to another. The security policy payload of MIKEY is able to carry different formats of security policy for different security protocols. MIKEY has defined a format of security policy for SRTP. To enable MIKEY to work with TESP, a new format of security policy for TESP should be defined.

**Mapping CS ID to TESP Secure Connection**

One of the advantages of using MIKEY as the key management protocol is that MIKEY is able to setup multiple security associations for multiple crypto sessions sharing one TGK (i.e. multiple crypto sessions in one crypto session bundle). Each crypto session should be given a unique CS ID and this CS ID is used to point to the correct security association that is setup by a MIKEY handshake. A CS ID map is provided as information to map the CS IDs to crypto sessions.

When MIKEY is used to setup security associations for SRTP streams (i.e. crypto sessions), the SSRC is used to identify which CS ID is assigned to which SRTP stream. An SRTP-ID map is specified in MIKEY to provide information of the mapping of CS ID to SRTP streams. To enable MIKEY to support TESP, a new CS ID map should be defined for TESP. This new CS ID map should have a new CS ID map type. The port information and transport layer protocol should be used to identify which CS ID is assigned to which TESP secure connection.

The current design of the common header payload limits MIKEY to having only one CS ID map per CSB. If between two endpoints there are more than one SRTP sessions and more than one TESP secure connections to be setup simultaneously, two separate CSB should be created. If the intention is to have multiple SRTP streams and TESP secure connections within the same CSB (i.e. to share one TGK), a new map type that can contains more than one map (e.g. one map for SRTP streams and another map for TESP secure connections) must be defined.

# 11. Conclusions

The goal of this thesis was to find a suitable solution for securing peer-to-peer communication in heterogeneous networks on top of the transport layer. Because of the additional delay caused by inband handshake and the poor compatibility with some transport protocols, it was determined that existing security protocols such as TLS and DTLS were not suitable in such a usage scenario and a new security protocol should be designed.

During this process, a new security protocol called 'Raw TLS' was first designed by simply removing the handshake sub-protocol and the changecipherspec sub-protocol of TLS and DTLS. According to the performance evaluation of 'Raw TLS', the removal of inband handshake achieved by 'Raw TLS' does shorten the initialization delay noticeably. However, 'Raw TLS' still has some shortcomings including compatibility issues with some of the transport protocols and the lack of rekeying and key refreshment mechanisms. Therefore, another security protocol called TESP was proposed. TESP not only retains the advantage of 'Raw TLS' (i.e. the low initialization delay) but also resolves all the incompatibility issues with transport protocols. It also provides mechanisms for rekeying and key refreshment. A security analysis of TESP was carried out and no security flaws were found.

# 12. Future Work

## 12.1. Implementation and Performance Evaluation of TESP

An analysis and modeling of TESP with formal methods and tools will help discover errors of the security protocol if there is any. An implementation of TESP could be done as a proof-of-concept. A performance evaluation and comparison with other security protocols could also provide information about the feasibility of TESP. It is expected that the saving of the initialization delay of TESP will be the same as that of 'Raw TLS' shown in section 10.3.3; and the throughput performance of TESP will be slightly better than TLS because TESP improves bandwidth utilization with the adaptive encapsulation function.

## 12.2. Specifying Transport over TESP in SDP

Since TESP is designed for peer-to-peer communication, it should be specified how to establish secure connections over TESP using SDP. New protocol identifiers such as 'TCP/TESP', 'SCTP/TESP', 'UDP/TESP', 'DCCP/TESP' should be defined.

# References


[1] C. A. Kent and J. C. Mogul, “Fragmentation Considered Harmful”, in the proceedings of SIGCOMM 1987, vol. 17, no. 5, October 1987

[2] P. Srisuresh and M. Holdrege, “IP Network Address Translator (NAT) Terminology and Considerations”, IETF, RFC 2663, August 1999.

[3] U. Blumenthal, et al., “Securely Enabling Intermediary-based Transport Services”, Internet Draft, IETF, Expired, June, 2003

[4] R. Stewart, et al., “Stream Control Transmission Protocol”, RFC 2960, IETF, October 2000.

[5] R. Stewart, et al., “SCTP Partial Reliability Extension”, RFC 3758, IETF, May 2004.

[6] R. Stewart, et al., “Stream control transmission protocol (SCTP) dynamic address reconfiguration”, Internet Draft, IETF Transport Area Working Group, Work in progress February 2003.

[7] E. Kohler, M. Handley, and S. Floyd, “Datagram Congestion Control Protocol (DCCP)”, Internet Draft, IETF, November 2004.

[8] A. J. Menezes, et al., ”Handbook of Applied Cryptography”, Published by CRC Press, October 1996.

[9] H. Krawczyk, et al., ”HMAC: Keyed Hashing for Message Authentication”, RFC 2104, IETF, February 1997.

[10] R. Rivest, “The MD5 Message Digest Algorithm”, RFC 1321, IETF, April 1992.

[11] D. Eastlake, 3rd and P. Jones , “US Secure Hash Algorithm 1 (SHA1)”, RFC 3174, IETF, September 2001.

[12] R. Thayer and K. Kaukonen, “A Stream Cipher Encryption Algorithm”, Internet Draft, IETF, Expired, July 1999.

[13] S. Casner, et al., ”RTP: A Transport Protocol for Real-Time Applications”, RFC 3550, IETF, July 2003.

[14] J. Rosenberg, et al., “SIP: Session Initiation Protocol”, RFC 3261, IETF, June 2002.

[15] M. Handley and V. Jacobson, “SDP: Session Description Protocol”, RFC 2327, IETF, April 1998.

[16] F. Andreasen, et al., “Session Description Protocol Security Descriptions”, Internet Draft, IETF, Work in Progress, July 2004.

[17] B. Ramsdell (Editor), “S/MIME Version 3 Message Specification”, RFC 2633, IETF, June 1999.

[18] J. Arrko, et al., "MIKEY: Multimedia Internet KEYing", RFC 3830, IETF, August 2004.

[19] M. Baugher, et al., “MSEC Group Key Management Architecture”, Internet Draft, IETF, Work in Progress, June 2004.

[20] M. Johansson, “Practical Evaluation of Revocation Schemes”, M.Sc. Thesis, KTH/NADA, 2004.

[21] M. Baugher, et al., "The Secure Real Time Transport Protocol", RFC 3711, IETF, March 2004.

[22] P. Karlton and P. C. Kocher, ”The SSL Protocol”, Netscape Communications Corp.,

Febuary 1995.

[23] T. Dierks and C. Allen, "The TLS Protocol - Version 1.0", RFC 2246, IETF, January 1999.

[24] T. Dierks and E. Rescorla, “The TLS Protocol - Version 1.1”, Internet Draft, IETF, Work in Progress, August 2004.

[25] D. Wagner and B. Schneier, “Analysis of the SSL 3.0 protocol”, The Second USENIX Workshop on Electronic Commerce Proceedings, USENIX Press, November 1996, pp. 29-40.

[26] B. Moeller, "Security of CBC Ciphersuites in SSL/TLS: Problems and Countermeasures", [Referred 24.09.2004], <http://www.openssl.org/~bodo/tls-cbc.txt>.

[27] P. Eronen and H. Tschofenig, “Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)”, Internet Draft, IETF, Work in Progress, September 2004.

[28] S. Blake-Wilson, et al., “Transport Layer Security (TLS) Extensions”, RFC 3546, IETF, June 2003.

[29] M. Mayers, et al., “X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP”, RFC 2560, IETF, June 1999.

[30] A. Jungmaier, et al., “Transport Layer Security over Stream Control Transmission Protocol, RFC 3436, IETF, December 2002.

[31] WAP Forum, “WAP TLS Profile and Tunneling Specification, 11-April-2001”, [Referred 24.09.2004], <http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html>

[32] R. Khare and S. Lawrence, “Upgrading to TLS Within HTTP/1.1”, RFC 2817, IETF, May 2000.

[33] WAP Forum, “Wireless Transport Layer Security Specification Version 06-Apr -2001”, [Referred to on 24.09.2004], <http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html>

[34] Markku-Juhani Saarinen, “Attacks Against the WAP WTLS Protocol”, University of Jyväskylä, Finland, [Referred to on 30.09.2004], <www.jyu.fi/~mjos/wtls.pdf>

[35] WAP Forum, “Wireless Application Protocol Architecture Specification Version 12-July-2001”, [Referred to on 24.09.2004] <http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html>

[36] WAP Forum, “WAP Certificate and CRL Profiles”, [Referred to on 24.09.2004], <http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html>

[37] E. Rescorla and N. Modadugu, “Datagram Transport Layer Security”, Internet Draft, IETF, Work in Progress, February 2004.

[38] H. Inamura, et al., ”TCP over Second (2.5G) and Third (3G) Generation Wireless Networks”, RFC 3481, IETF, February 2003.

[39] L. Westberg and M. Lindqvist, "Real-time Traffic over Cellular Access Networks", Internet Draft, IETF, Expired, May 2000.

[40] K. Svanbro, et al., ”The challenges of voice-over-IP-over-wireless”, Ericsson Review, Issue no. 01/2001

[41] Stephan Rein, et al., “Voice Quality Evaluation for Wireless Transmission with ROHC”, in the Proceedings of the 7th IASTED International Conference on Internet and Multimedia Systems and Applications (IMSA), pages 461-466, Honolulu, HI, August 2003.

[42] V. Jacobson, “Compressing TCP/IP Headers for Low-Speed Serial Links”, RFC 1144, IETF, February 1990.

[43] M. Degermark, et al., ”IP Header Compression”, RFC 2507, IETF, February 1999.

[44] S. Casner and V. Jacobson, “Compressing IP/UDP/RTP Headers for Low-Speed Serial Links”, RFC 2508, IETF, February 1999.

[45] C. Bormann, et al., “RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed”, RFC 3095, IETF, July 2001.

[46] G. Pelletier, et al., ”RObust Header Compression (ROHC): A profile for TCP/IP (ROHC-TCP)”, Internet Draft, IETF, Work in Progress, July 2004.

[47] 3GPP TS 25.323: “Technical Specification Group Radio Access Network; Packet Data Convergence Protocol (PDCP) Specification (Release 5)”, September 2002.

[48] S. Kent and R. Atkinson, “Security Architecture for the Internet Protocol”, RFC 2401, IETF, November 1998.

[49] S. Kent and R. Atkinson, “IP Authentication Header”, RFC 2402, IETF, November 1998.

[50] S. Kent and R. Atkinson, “IP Encapsulating Security Payload (ESP)”, RFC 2406, IETF, November 1998.

[51] D. Piper, “The Internet IP Security Domain of Interpretation for ISAKMP “, RFC 2407, IETF, November 1998.

[52] D. Maughan, “Internet Security Association and Key Management Protocol (ISAKMP)“, RFC 2408, IETF, November 1998.

[53] D. Harkins and D. Carrel, “The Internet Key Exchange (IKE)”, RFC 2409, IETF, November 1998.

[54] AbdelNasir Alshamsi and Takamichi Saito, “A technical comparison of IPSec and SSL”, Tokyo, [Referred to on 18.02.2005], < http://eprint.iacr.org/2004/314.pdf>

[55] ITU-T Recommendation Q.700, “Introduction To ITU-T Signaling System No. 7 (SS7)”.

[56] P. Srisuresh and M. Holdrege, “IP Network Address Translator (NAT) Terminology and Considerations”, RFC 2663, IETF, August 1999.

[57] J. Border, et al., ”Performance Enhancing Proxies Intended to Mitigate Link-related Degradations”, RFC 3135, IETF, June 2001.

[58] P. Srisuresh, et al., “Middlebox communication architecture and framework”, RFC 3303, IETF, August 2002.

[59] I. Hajjeh and A. Serhrouchni, "ISAKMP Handshake for SSL/TLS", IEEE GLOBECOM '03, Vol. 3, San Francisco, USA, 1-5 December 2003, Pages: 1481-1485.

[60] D. Yon and G. Camarillo, “Connection-Oriented Media Transport in the Session Description Protocol (SDP)”, Internet Draft, IETF, Work in Progress, September 2004.

[61] R. Fairlie-Cuninghame, “Guidelines for specifying SCTP-based media transport using SDP”, Internet Draft, IETF, Expired, May 2001.

[62] G. Parsons, “Real-time Facsimile (T.38) - image/t38 MIME Sub-type Registration”, RFC 3362, IETF, August 2002.

[63] “ECRYPT Yearly Report on Algorithms and Key sizes (Rev. 2004)”, to appear in ECRYPT (www.ecrypt.eu.org).

[64] I. Mironov, “Random Shuffles of RC4”, In Proceedings of Crypto, August 2002

[65] A. Biryukov and A. Shamir, “Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers”, Proceedings, ASIACRYPT 2000, LNCS 1976, pp. 1-13, Springer Verlag.

[66] D. Mcgrew and S. Fluhrer, “Attacks on Encryption of Redudant Plaintext and Implications on Internet Security”, the Proceedings of the Seventh Annual Workshop on Selected Areas in Cryptography (SAC 2000), Springer-Verlag.

[67] M. Dalal, “Transmission Control Protocol security consideration”, Internet-Draft, IETF; June 2004.

[68] A. Heffernan, “Protection of BGP Sessions via the TCP MD5 Signature Option”, RFC 2385, August 1998.

[69] T. Aura, P. Nikander, and G. Camarillo, "Effects of Mobility and Multihoming on Transport-Protocol Security", In Proc. 2004 IEEE Symosium on Security and Privacy (SSP'04), May 2004. IEEE Computer Society.

[70] X. Wang, D. Feng, X. Lai, and H. Yu, "Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD", rump session, CRYPTO 2004, August 2004.

[71] Eric Rescorla, “An Introduction to OpenSSL Programming, Part I and Part II”, Linux Journal, the September 2001 issue.

[72] John Viega, et. al., “Network Security with OpenSSL”, June 2002, O’reilly.

[73] Dummynet Home Page, <http://info.iet.unipi.it/~luigi/ip_dummynet/>, [Referred to on 14.03.2005]

# A. Appendix

## A.1. Performance Figures

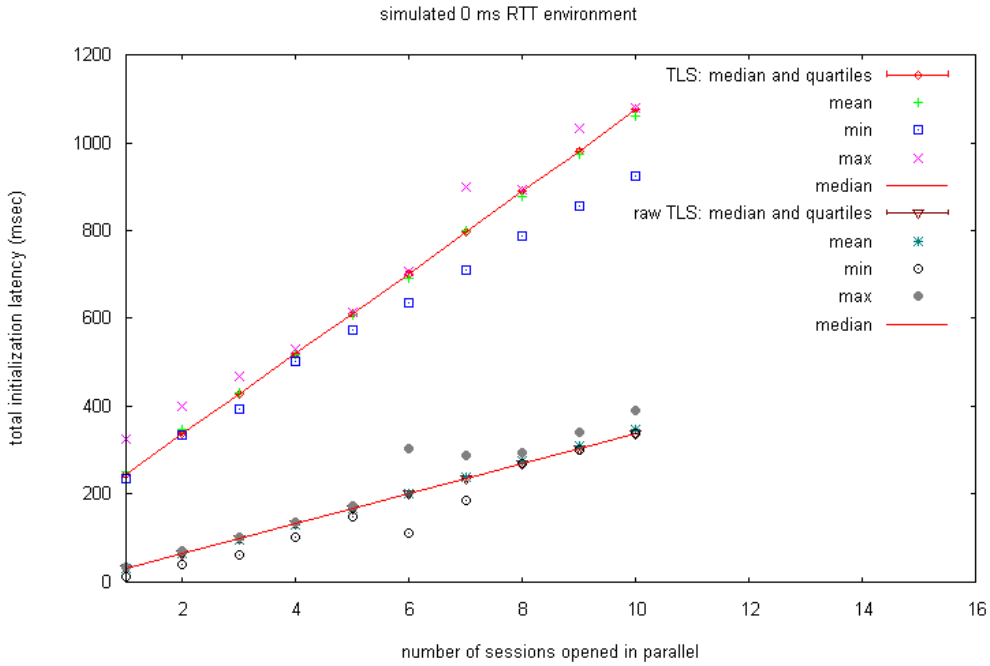



Figure A. 1. Performance of TLS and 'raw TLS' in simulated 0 msec RTT environment

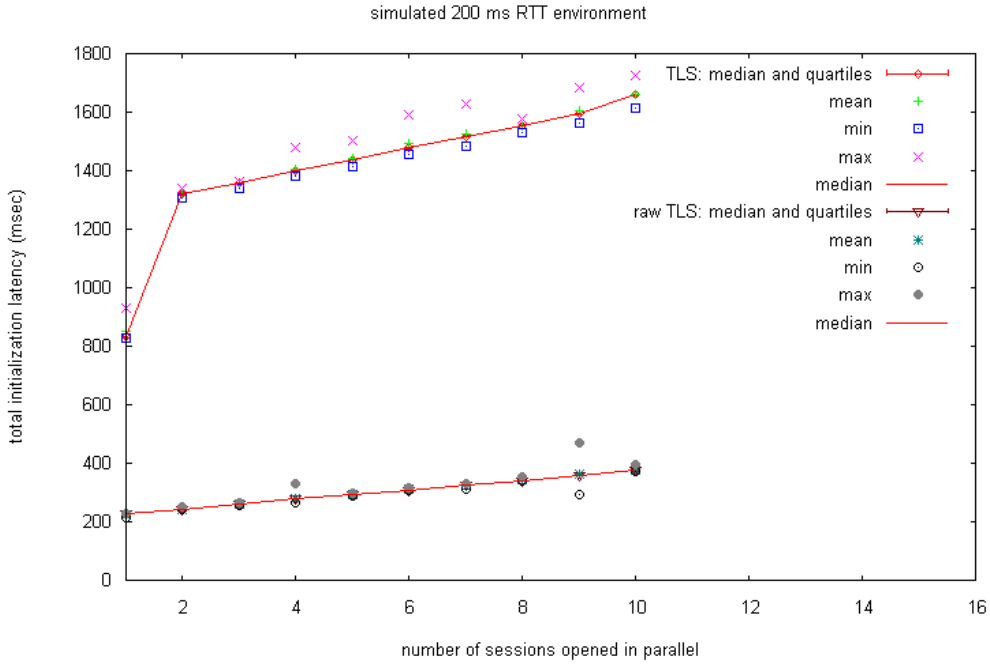



Figure A. 2. Performance of TLS and 'raw TLS' in simulated 200 msec RTT environment

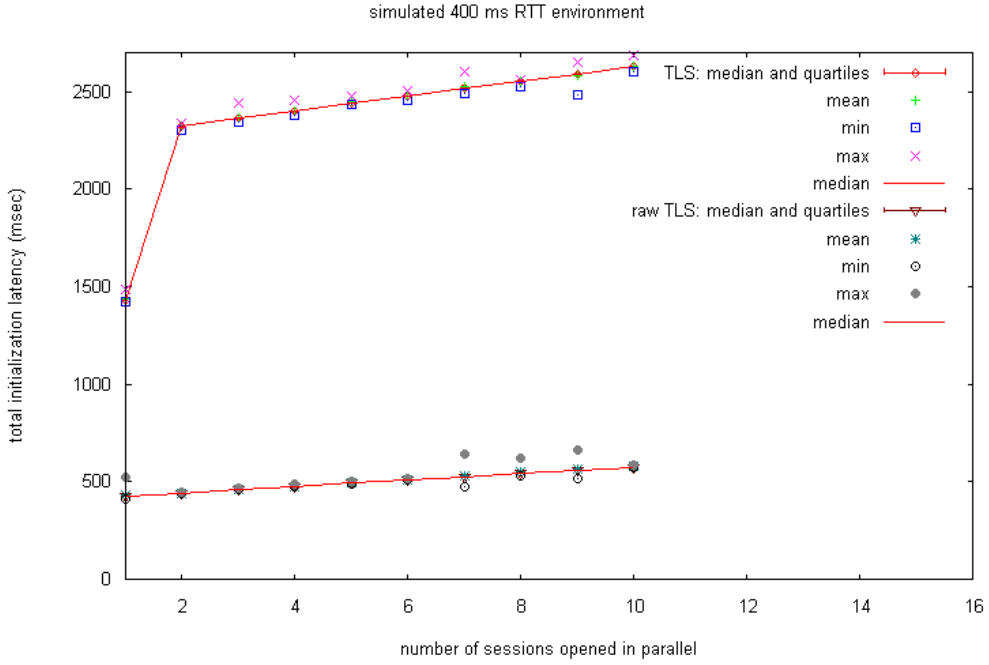



Figure A. 3. Performance of TLS and 'raw TLS' in simulated 400 msec RTT environment

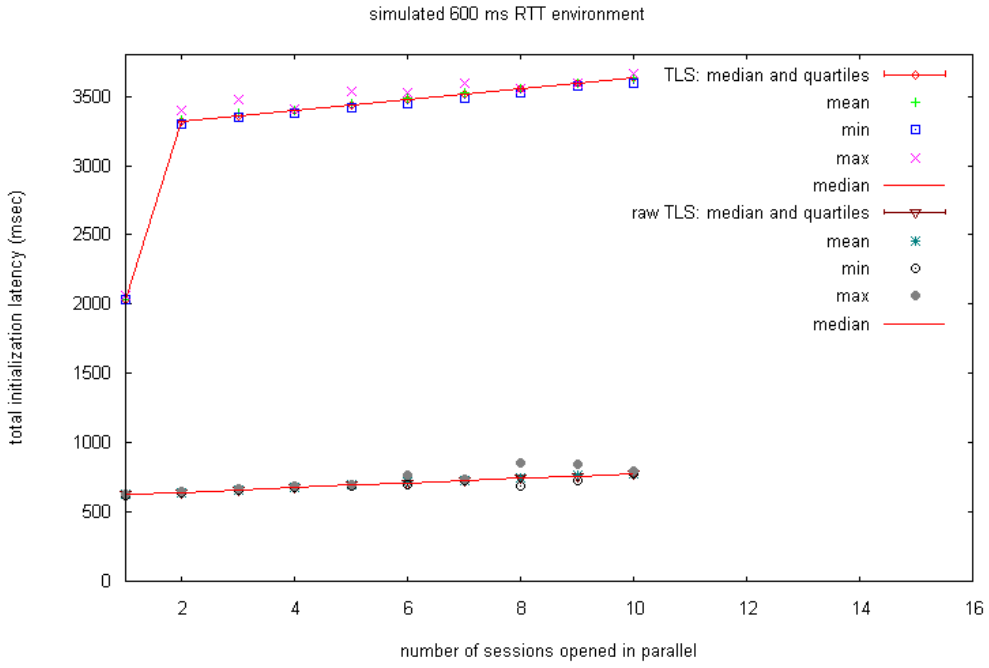



Figure A. 4. Performance of TLS and 'raw TLS' in simulated 600 msec RTT environment

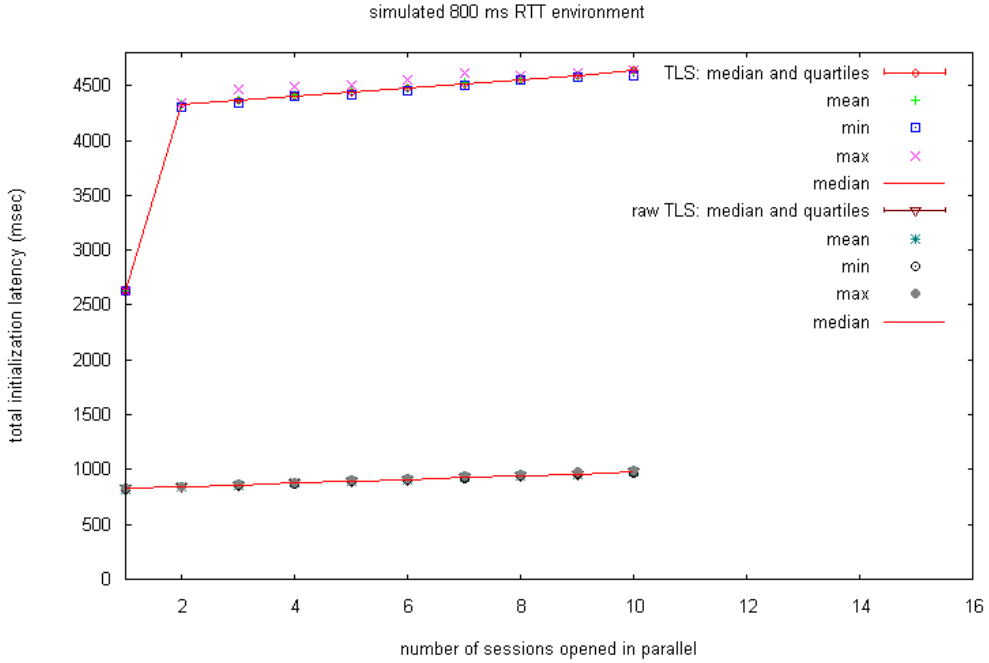



Figure A. 5. Performance of TLS and 'raw TLS' in simulated 800 msec RTT environment

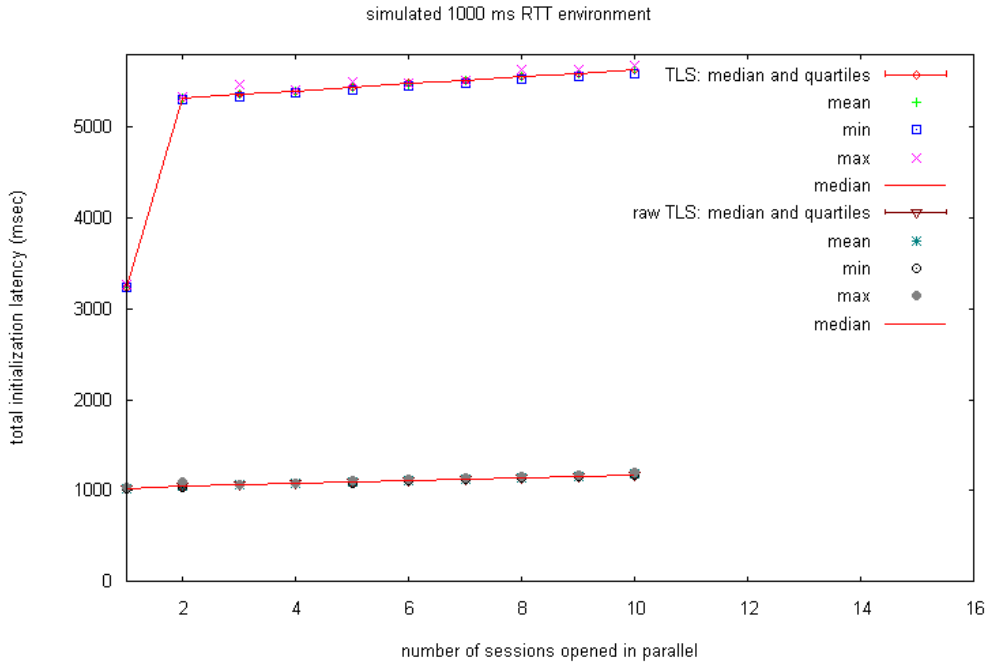



Figure A. 6. Performance of TLS and 'raw TLS' in simulated 1000 msec RTT environment

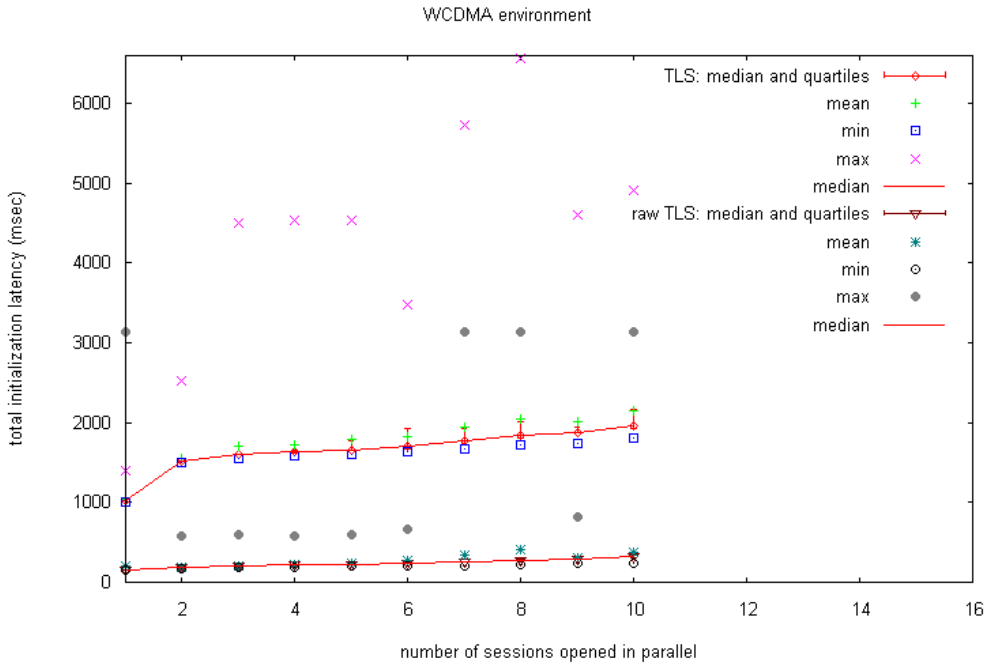



Figure A. 7 Performance of TLS and ‘raw TLS’ in real WCDMA environment

IMIT/LCN 2005-04

www.kth.se