Analysing TLS Using the Strand Spaces Model

Allaa I. Kamil and Gavin Lowe*

April 8, 2008

Abstract

In this paper, we analyse the Transport Layer Security (TLS) protocol within the strand spaces setting. In [BL03] Broadfoot and Lowe suggested an abstraction of TLS. The abstraction models the security services that appear to be provided by the protocol to the high-level security layers. The outcome of our analysis provides a formalisation of the security services provided by TLS and proves that, under reasonable assumptions, the abstract model suggested by Broadfoot and Lowe is correct. To that end, we reduce the complexity of the protocol using safe simplifying transformations. We extend the strand spaces model in order to include the cryptographic operations used in TLS and facilitate its analysis. Finally, we use the extended strand spaces model to fully analyse TLS with its two main components: the Handshake and Record Layer Protocols.

*Authors' address: Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, OX1 3QD, UK; {allaa.kamil, gavin.lowe}@comlab.ox.ac.uk.

Contents

1 Introduction

For the last two decades, the area of formal verification of security protocols has been extensively researched. Many verification and analysis methods have been developed (e.g. [BAN89, Low01, THG98]) and used successfully to analyse a number of standard security protocols. Despite these advances, many of the practical security systems today cannot be verified using these techniques. Such systems are usually built as large security architectures and comprise many interacting components. The existing verification techniques do not scale well to handle such large architectures. In addition, the interaction between different security components has not been addressed by the majority of formalisms [Cre04].

A security protocol is a sequence of messages exchanged between two or more agents to satisfy a pre-defined security goal. The goal should be reached even in the presence of an intruder whose capabilities depend on the surrounding environment in which the security protocol is executed.

The Dolev-Yao intruder [DY83] is the de facto standard threat model for analysing security protocols. In this model, the intruder is assumed to be in complete control of the network and may overhear, inject, and intercept any message sent on the network in addition to the ability to participate in protocol runs using his own identity. However, the Dolev-Yao intruder cannot perform cryptanalysis on the cipher texts used since we assume perfect cryptography. In other words, the Dolev-Yao model abstracts away from the details of cryptographic primitives such as hashing, encryption, and signature. This layer of abstraction has facilitated the security analysis of cryptographic protocols and improved the scalability of automated analysis techniques.

Many practical security systems are built from layered security protocols, with a high-level security layer running on top of one or more lower-level secure channels. In principle, such a system can be analysed by direct modelling the whole layered architecture. However, this direct approach has clear disadvantages in terms of model complexity and analysis inefficiency. To analyse layered security architectures, we can adopt a layered approach that uses the same concept used in the Dolev-Yao model: adding a new layer of abstraction [BL03, Cre04]. This approach comprises the following steps:

- Analyse the low-level secure channel within the Dolev-Yao model and formalise the security services it provides for the high-level security

layers;

- Abstract away from the details of the implementation of the secure channel and just model the services it provides;
- Analyse the high-level layers using the new layer of abstraction developed and hence verify the whole security architecture.

In addition to its benefits in terms of modelling simplicity and automated analysis efficiency [BL03], this layered approach has huge advantages in terms of reusability. Once an abstract model of a secure channel is developed, it can be reused in the verification of different security architectures with different high-level protocols. Similarly, if we prove the correctness of a high-level protocol on a certain secure channel, the same proof applies, under certain conditions, if the protocol is run on a secure channel that provides the same, or stronger, security services but has a different implementation.

In [BL03], Broadfoot and Lowe adopted the above approach to analyse layered security architectures and suggested an abstraction of the Transport Layer Security (TLS) Protocol as an example of a low-level secure channel. The abstraction models the security services that appear to be provided by TLS for the high-level security layers, sometimes referred to as the *Transaction Layers*. In this paper we analyse the TLS Protocol, formalise the security services it provides, and prove that the abstract model suggested by Broadfoot and Lowe was indeed correct. The TLS protocol has been our choice for analysis since it is an industry standard used in almost every online financial transaction as a low-level secure channel. The verification method used is the strand spaces model [THG98].

Since its development in 1999, TLS has been analysed many times, e.g. [Pau99, DCVP04]. However, to our knowledge, no previous analysis has been able to completely verify the protocol or formally describe the security services it provides for the Transaction Layer. By examining the TLS protocol, the challenges that face its formal analysis become clear:

Size The TLS protocol is considerably larger than most of the protocols in the academic literature in terms of number and length of messages exchanged. According to Yasinsac and Childs [YC01], any form of the TLS protocol is ten to fifteen times larger than the majority of the protocols listed in the canonical security protocol collection by Clark and Jacob [CJ97].

Cryptographic complexity Many security verification techniques lack the functionality to deal appropriately with some of the cryptographic operations used in TLS, e.g. using session keys generated by pseudo-random functions. Such operations complicate the protocol model and, consequently, the analysis of the protocol.

Multi-layer interaction The TLS protocol is primarily designed to provide security services for the messages exchanged in the Transaction Layer. The syntactic structure of the transaction protocol is not specified by the protocol. Indeed, the Transaction Layer messages could leak keys used by the Handshake and Record Layer protocols, and therefore cause their failure to achieve their security goals; more subtly, multi-layer attacks may happen, where a message from one layer is replayed and interpreted as being a message from the other layer, leading to an attack. Until now, no one has verified the combination of the two layers, and the possibility of interactions between them.

In our analysis of TLS, we tackle each of these difficulties. We deal with the problem of protocol size by using safe simplifying transformations [HL99] to reduce the number and length of messages. We tackle the problem of cryptographic complexity by extending the strand spaces model to incorporate the cryptographic operations used in TLS in order to be able to reason formally about them. Finally, we consider multi-layer interaction by carefully stating some general assumptions about the syntactic structure of the Transaction Layer messages.

TLS involves authenticating the peers' identities using public key cryptography. This authentication can be unilateral, i.e. only the server is authenticated, or bilateral, i.e. the client and the server are mutually authenticated. In this paper we only consider the TLS protocol in the bilateral authentication mode. We also do not consider the case where the shared secret is negotiated using Diffie-Hellman exchange [DA99].

This paper is organised in five sections. In Section 2 we give an overview of the TLS protocol and its security goals. We also simplify the protocol by removing its redundancy and put it in a suitable form for analysis. In Section 3 we explain the strand spaces model. We also describe the extensions and modifications we made to the model to facilitate the analysis of TLS. In Section 4 we analyse the TLS protocol using the strand spaces model and formalise the security services it provides. Finally, in Section 5, we discuss

how our work compares to previous work on the analysis of TLS, and we sum up our results.

2 The Transport Layer Security (TLS) Protocol

In this section we provide a detailed description of the Transport Layer Security (TLS) protocol and its security goals. We start by giving a brief history of TLS. We then proceed to describe TLS and its two stages: the initial Handshake Protocol and the Record Layer Protocol. This is followed by a discussion of the difficulties that arise when analysing TLS. Finally, we attempt to reduce the complexity of the protocol by using safe simplifying transformations [HL99].

2.1 History of TLS

The original motivation for developing secure transport protocols was the Internet. The Secure Sockets Layer Protocol (SSL) version 1.0 was developed by Netscape Communications in June 1994 as a response to the growing security concerns on the Internet. Due to some major drawbacks, mainly missing support for credit card transactions over the Internet, the first version of SSL was never released. A few months later, Netscape shipped its first product with support for SSL version 2.0: Netscape Navigator. SSL version 2.0 offered confidential credit card transactions and server authentication with the use of digital certificates and encryption. Unfortunately, SSL version 2.0 turned out to be insecure [GW96]. In October 1995, Microsoft published Private Communication Technology (PCT) version 1.0 [BLS+95] to address some of the problems of SSL version 2.0. Many of the ideas of PCT were incorporated in SSL version 3.0 which was released by Netscape Communications in late 1995. In 1996 Microsoft released the Secure Transport Layer Protocol (STLP) [Str96], which was a modification to SSL version 3.0, providing additional features which Microsoft considered to be critical like support for datagrams (e.g. UDP) and client authentication using shared secrets.

In May 1996, the standardisation of SSL became the responsibility of the Internet Engineering Task Force (IETF). The IETF renamed SSL to Transport Layer Security (TLS) to avoid the appearance of bias toward any

particular company. The first version of TLS was released in January 1999 [DA99]. Despite the change of names, TLS is nothing more than a new version of SSL. In fact, there are far fewer differences between TLS version 1.0 and SSL version 3.0 than those between SSL version 3.0 and SSL version 2.0 [Tho00]. TLS version 1.0 is the focus of the rest of this paper.

2.2 TLS: Description

In this section we describe the basic concepts of TLS, its components, and its operation. All the cryptographic details are explained in Appendix A.

According to the TLS specification document [DA99], the primary goal of the TLS Protocol is to provide privacy and data integrity between two communicating applications. The protocol defines two different roles for the communicating parties: a *client* role and a *server* role. The client is the principal that initiates the secure communication while the server responds to the client's request.

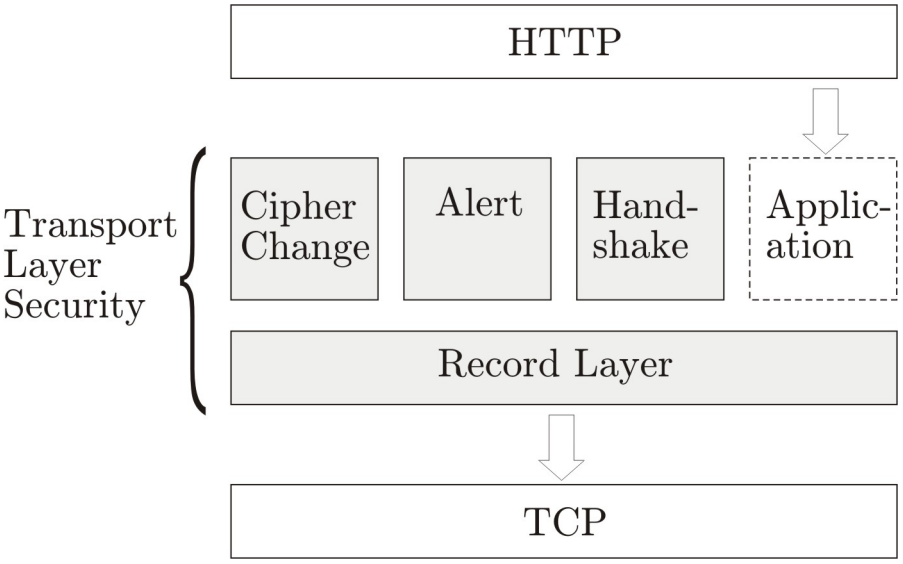


Figure 1: The composition of TLS

TLS operates on top of some reliable transport protocol (e.g. Transmission Control Protocol (TCP)) and below some higher-level transaction protocols

(e.g. Hyper Text Transport Protocol (HTTP)). In other words, TLS uses transport protocols on behalf of higher-level transaction protocols to establish a secure channel. TLS itself is not a single protocol, but consists of four sub-protocols (See Figure 1):

- The *Handshake* protocol: The Handshake Protocol is used by the communicating parties to agree on the cryptographic parameters that will be used in the initiated session.

- The *ChangeCipherSpec* protocol: The ChangeCipherSpec protocol is used by the sending party to notify the receiving party that the subsequent messages should be protected by the negotiated cryptographic parameters. The protocol uses a single byte with value 1 to indicate the transition.

- The *Alert* protocol: The Alert protocol provides exception handling for SSL secured connections. Alert messages indicate the severity of an alert (i.e. warning or fatal) and a description of the alert.

- The *Record Layer* protocol: The Record Layer Protocol encapsulates messages from the higher layer protocols including messages from the Handshake, ChangeCipherSpec, Alert, and Transaction protocols. The data stream is fragmented into a series of records. These records are protected by the cryptographic parameters agreed upon using the Handshake Protocol. The records are then passed to a transport layer protocol (such as TCP) for transmission.

It is clear from the previous description that, from a security point of view, the TLS protocol is composed of two main components. At the lowest level, layered on top of some reliable transport protocol, is the TLS Record Layer Protocol. The Record Layer Protocol is responsible for data transfer. The other component is the TLS Handshake Protocol which is responsible for establishing TLS session states between communicating peers. In the following subsections we give a detailed description of the Handshake Protocol and the Record Layer Protocol.

2.2.1 The Authentication Handshake

According to the TLS version 1.0 specification document [DA99], the Handshake Protocol provides connection security that has three basic properties:

1. The negotiation of a shared secret is secure: the negotiated secret is unavailable to eavesdroppers, and for any authenticated connection the secret cannot be obtained, even by an attacker who can place himself in the middle of the connection.

2. The negotiation is reliable: no attacker can modify the negotiation communication without being detected by the communicating parties.

3. The peer's identity can be authenticated using public key cryptography (see Appendix A for more details). This authentication can be unilateral, i.e. only the server is authenticated, or mutual, i.e. the client and the server are mutually authenticated.

As stated previously, in this paper we analyse the TLS protocol in the mutual authentication mode. Below we describe a TLS Handshake exchange between a client and a server in the mutual authentication mode.

Each handshake message contains three fields:

1. *message_type*: Indicates the type of the message, e.g. ClientHello, ServerHello, CertificateRequest, etc.

2. *message_length*: Indicates the length of the message.

3. *Content Parameters*: Refers to the content associated with each handshake message. If a content parameter is a list it is always preceded by its length.

Each TLS handshake message is presented below in the following form:

$$Message\_no.\quad Message\_type\quad A \longrightarrow B:\quad Content\ Parameters$$

$Message\_no$ refers to the order in which the message is sent in the handshake protocol. $Message\_type$ indicates the type of the message. A and B denote the sending and the receiving agents respectively. In the following message descriptions, c and s are used instead of A and B to refer to the client and the server respectively. $Content\ Pararmeters$ refers to the content parameters of the handshake message. Note that the $message\_type$ and $message\_length$ are omitted from the message body. The complete handshake exchange is included in Appendix B.

ClientHello:

1. ClientHello $c \longrightarrow s$: $client\_version, client\_nonce, sessionID\_length, sessionID, cipher\_suite\_length, cipher\_suites, compression\_length, compression\_methods$

This is the first message sent by a client in the TLS Handshake Protocol. Its main purpose is to communicate the clients connection preferences to the server. These preferences include:

- *client_version*: The highest TLS version supported by the client.
- *cipher_suites*: The list of cryptographic algorithms supported by the client (e.g. RSA, Diffie-Hellman) in decreasing order of preference.
- *compression_methods*: The list of compression algorithms supported by the client in decreasing order of preference.

The message also includes a fresh random number referred to as *client_nonce*, and an optional session identification number $sessionID$ that is used for session resumption purposes.

ServerHello:

2. ServerHello $s \longrightarrow c$: $server\_version, server\_nonce, sessionID\_length, sessionID, cipher\_suites, compression\_methods$

With this message, the server makes its choice out of the preferences offered by the client. These include the TLS version, cipher suite and compression algorithms that will be used for this connection. There is no obligation for a server to choose the client's preferred ciphers, even if supported on the server side.

The message also includes a fresh random number denoted by *server_nonce*, and returns the non-zero session identification number $sessionID$ received from the client if a previous session is to be resumed.

ServerCertificate:

3. ServerCertificate $s \longrightarrow c$: $certificate\_chain\_length, certificate\_list$

This message simply consists of a chain of X.509 certificates (See Appendix A), $certificate\_list$, presented in order, with the first one being the certificate that belongs to the server itself. This step is mandatory for the server during the handshake phase; it has to send its certificate in order to authenticate itself to the client.

ServerKeyExchange

4. ServerKeyExchange $s \longrightarrow c:$ $parameters, signed\_parameters$

This message includes the server's key exchange parameters $parameters$. These parameters contain enough data to allow the client to send a premaster secret. If the public key included in the $ServerCertificate$ message is suitable for encryption as well as verifying signatures, the field $parameters$ is empty. $parameters$ is concatenated to the $client\_nonce$ and the $server\_nonce$. The concatenated terms are hashed and signed by the signature key that corresponds to the public key provided in the $ServerCertificate$ message. The signed term is denoted by $signed\_parameters$ and has the following form:
$\{Hash(client\_nonce, server\_nonce, parameters)\}_{SK(s)}$

CertificateRequest

5. CertificateRequest $s \longrightarrow c:$ $certificate\_type\_length, certificate\_types,$
$certificate\_authorities\_length, certificate\_authorities$

This message is optional and is sent by the server to ask the client to send its certificate. The message includes the list of certificate types acceptable by the server, $certificate\_types$, and the certificate authorities recognised by the server, $certificate\_authorities$.

ServerHelloDone

6. ServerHelloDone $s \longrightarrow c:$

This message is sent by the server simply to indicate the conclusion of a handshake negotiation. The content parameters field of this message is empty.

ClientCertificate

7. ClientCertificate $c \longrightarrow s:$ $certificate\_chain\_length, certificate\_list$

This message is sent by the client after receiving a $ServerHelloDone$ message and iff the server has previously sent a $CertificateRequest$ message. The message includes the certificate chain of the client.

ClientKeyExchange

8. ClientKeyExchange $c \longrightarrow s:$ $\{client\_version, premaster\_secret\}_{PK(s)}$

Just as the $ServerkeyExchange$ provides the key information for the server, the $ClientKeyExchange$ tells the server the client's key information. The message contains the $premaster\_secret$ which is used later to generate the session keys. The $premaster\_secret$ is encrypted, together with the latest TLS version the client supports $client\_version$, under the public key of the server.

CertificateVerify

9. CertificateVerify $c \longrightarrow s:$ $\{Verifying\_Hash(handshake\_messages[1-8])\}_{SK(c)}$

By sending $CertificateVerify$, the client proves that it possesses the secret key that corresponds to the public key included in the $ClientCertificate$ message. The message contains a signed hash of $handshake\_messages[1-8]$ which refers to all the handshake messages exchanged up to but not including this message. The signature is then verified by the server.

ClientFinished

10. ClientFinished $c \longrightarrow s:$ $PRF(master\_secret, \text{“}clientfinished\text{”},$
$MD5(handshake\_messages[1-10]),$
$SHA(handshake\_messages[1-10]))$

The $ClientFinished$ message is sent by the client to prove that the handshake negotiation has been successful and that the negotiated cipher suite is in effect. The message includes a digest of the negotiated $master\_secret$

along with the label "$clientfinished$" and $handshake\_messages[1-9]$ hashed using the functions SHA and $MD5$. $master\_secret$ is generated from the $premaster\_secret$ using the pseudo random function PRF (See Appendix A) as follows:

$$master\_secret \;=\; PRF(premaster\_secret, \text{"}mastersecret\text{"}, \{client\_nonce, server\_nonce\})$$

$handshake\_messages[1-9]$ refers to the concatenation of all of the data from all handshake messages up to but not including this message.

ServerFinished

11. ServerFinished $s \longrightarrow c$: $PRF(master\_secret, \text{"}serverfinished\text{"}, MD5(handshake\_messages[1-10], SHA(handshake\_messages[1-10])))$

$ServerFinished$ is the last message of the handshake exchange. It is sent by the server to prove that the handshake negotiation has been successful and that the negotiated cipher suite is in effect. The message is very similar to the $ClientFinished$ message except for two things:

- $ServerFinished$ includes the label "$serverfinished$" while $ClientFinished$ includes the label "$clientfinished$"
- The $handshake\_messages[1-10]$ parameter for the $ServerFinished$ message is different from that for the $ClientFinished$ message, because the one which is sent second includes the prior one.

Since the $ClientFinished$ and $ServerFinished$ messages are sent after exchanging the $ChangeCipherSpec$ messages (See Figure 1), they are passed to the record layer to be protected using the negotiated cipher suite. Recipients of finished messages must verify that the contents are correct. Once a side has sent its Finished message and received and validated the Finished message from its peer, it may begin to send and receive transaction data over the established connection.

2.2.2 The Record Layer

As stated in [DA99], the Record Layer Protocol aims to establish a peer to peer connection that provides the followings:

- **Confidentiality**. Symmetric cryptography is used for data encryption (See Appendix A for more details about symmetric encryption). The keys for this symmetric encryption are generated uniquely for each connection and are based on the premaster secret negotiated by the TLS Handshake Protocol.

- **Message Integrity**. Message transport includes a message integrity check using a keyed MAC. Secure hash functions (e.g. SHA, MD5, etc.) are used for MAC computations.

As mentioned before, the Record Layer Protocol encapsulates all messages of the higher-layer messages. These messages go through the following operations before being transmitted via a transport layer protocol (such as TCP):

Fragmentation Fragmentation is the first operation that is performed on the higher-layer massages. Upper-layer messages are fragmented into a block of 2^{14} bytes or less of TLS plaintext. Multiple client messages of the same component type may be coalesced into a single TLS plaintext record, or a single message may be fragmented across several records. Some headers are added to the fragmented message. The format of the fragmented data become as follows:

$$\begin{aligned} fragmented\_record \;:=\; & content\_type, protocol\_version, message\_length, \\ & fragmented\_data \end{aligned}$$

Compression Compression is optionally applied on the fragmented transaction data. Messages are compressed using the compression method defined on the current session. Compression must be lossless and may not increase the content length by more than 1024 bytes. The format of the compressed message is:

$$\begin{aligned} compressed\_record \;:=\; & content\_type, protocol\_version, compressed\_length, \\ & compressed\_fragment \end{aligned}$$

Message Authentication Code Message authentication code (MAC) is computed over the compressed data. TLS uses the standard message authentication code $HMAC$. $HMAC$ requires two parameters: a secret parameter and a data parameter. The secret parameter is computed from the parameters negotiated by the Handshake Protocol. Firstly, a Key_Block is generated from the $master\_secret$, the client nonce, and the server nonce as follows:

$$Key\_Block := PRF(master\_secret, \text{“}key\ expansion\text{”}, \{server\_nonce, client\_nonce\})$$

Then the Key_Block is partitioned into four partitions. The first and second partitions are used for the $Client\_MAC$ and the $Server\_MAC$ respectively as follows:

$$\begin{aligned} Client\_MAC &:= Quarter(Key\_Block, 0) \\ Server\_MAC &:= Quarter(Key\_Block, 1) \end{aligned}$$

The third and the fourth partitions are used to generate the encryption keys as we will discuss later. The data parameter includes the compressed record after applying HMAC, the format of the message becomes:

$$\begin{aligned} client\_MAC\_record &:= HMAC(Client\_MAC, \{seq\_number, compressed\_record\}) \\ server\_MAC\_record &:= HMAC(Server\_MAC, \{seq\_number, compressed\_record\}) \end{aligned}$$

Encryption Encryption is applied using the symmetric encryption key generated from the third and fourth partitions of *Key Block*.

$$\begin{aligned} Client\_Key &:= PRF(Quarter(Key\_Block, 2), \text{“}client\ encryption\text{”}, \\ &\quad \{server\_nonce, client\_nonce\}) \\ Server\_Key &:= PRF(Quarter(Key\_Block, 3), \text{“}server\ encryption\text{”}, \\ &\quad \{server\_nonce, client\_nonce\}) \end{aligned}$$

The encrypted message takes the following form:

$$
\begin{array}{lcl}
encrypted\_client\_record & := & \{compressed\_record, Client\_MAC_record, \\
 & & padding, padding\_length\}_{Client\_Key} \\
encrypted\_server\_record & := & \{compressed\_record, Server\_MAC_record, \\
 & & padding, padding\_length\}_{Server\_Key}
\end{array}
$$

Adding a header At this final stage, a header is concatenated to each message. This header consists of:

- *content_type*: Indicates which higher-layer protocol's message is being transmitted.
- *protocol_version*: Indicates the version of TLS being used.
- *message_length*: The length in bytes of the plaintext/compresses fragment.

After performing these operations the final form of a record layer message sent by the client is:

$$
\begin{array}{lll}
\text{ClientRecord} & c \longrightarrow s: & content\_type, protocol\_version, message\_length \\
 & & encrypted\_client\_record \\
\text{ClientRecord} & s \longrightarrow c: & content\_type, protocol\_version, message\_length \\
 & & encrypted\_server\_record
\end{array}
$$

2.3 Simplifying TLS

The idea of simplifying transformations is to remove redundant information present in the protocol whilst not losing any possible attacks [HL99]. The end result should be a fault-preserved version of the original protocol, whereby any attack on the original is also an attack upon the simplified version. When the simplified version is then checked for attacks, if any are found they can be tested against the full protocol to see if they are indeed an actual attack. If the simplified protocol can be proved secure, then the original protocol is also secure.

Like most commercial protocols, the TLS protocol contains a number of fields that are included for functionality rather than security. Some messages also include more hashings than are necessary. This added complexity

makes analysis more difficult. fault preserving simplifying transformations have been used before to simplify the TLS protocol [Kar01, Aut04]. Our simplied version of TLS is similar to the one presented in [Aut04] with minor modifications. Below are the simplifying transformations which are used to reduce the complexity of the TLS protocol.

Removing atomic fields The *Removing atomic fields* transformation completely removes some fields from the protocol. We will use this transformation on all the atomic values that are in the protocol for reasons of functionality rather than security such as *message_length*, *cipher_suite_length*, *cipher_suite*, *content_type*, etc. We will also remove the session-related fields $sessionID$ and $sessionID\_length$ since we do not consider session resumption in our analysis. In addition, we assume that the public key certificates include public keys suitable for encryption. Therefore, the fields $Parameters$ and $signed\_parameters$ are not needed. Applying this transformation, some of the messages become empty and are completely removed. The Handshake Protocol becomes as follows:

1. ClientHello	$c \longrightarrow s:$	$client\_nonce$
2. ServerHello	$s \longrightarrow c:$	$server\_nonce$
3. ServerCertificate	$s \longrightarrow c:$	$Certificate\_List$
4. ClientCertificate	$c \longrightarrow s:$	$Certificate\_List$
5. ClientKeyExchange	$c \longrightarrow s:$	$premaster\_secret_{PK(s)}$
6. CertificateVerify	$c \longrightarrow s:$	$\{Verifying\_Hash(handshake\_messages[1-5])\}_{SK(c)}$
7. ClientFinished	$c \longrightarrow s:$	$PRF(master\_secret, \text{“}client finished\text{”},$ $MD5(handshake\_messages[1-6],$ $SHA(handshake\_messages[1-6])))$
8. ServerFinished	$s \longrightarrow c:$	$PRF(master\_secret, \text{“}server finished\text{”},$ $MD5(handshake\_messages[1-7],$ $SHA(handshake\_messages[1-7])))$

Please note that the *handshake_messages* field now refers to the *simplified* handshake messages exchanged up to but not including the current message.

Record layer messages become as follows:

$$
\begin{array}{lll}
\text{ServerRecord} & s \longrightarrow c: & \{transaction\_message, \\
 & & HMAC(Server\_MAC, seq\_number, transaction\_message)\}_{Server\_Key} \\
\text{ClientRecord} & c \longrightarrow s: & \{transaction\_message, \\
 & & HMAC(Client\_MAC, seq\_number, transaction\_message)\}_{Client\_Key}
\end{array}
$$

Renaming For clarity, all the fields in the protocol are now given shorter names. The certificate lists are expressed as a function $CERT$ of the certificate and the certificate issuer. Applying an injective renaming function and putting the Handshake and the Record Layer protocols together:

$$
\begin{array}{lll}
1.\ \text{ClientHello} & c \longrightarrow s: & r_c \\
2.\ \text{ServerHello} & s \longrightarrow c: & r_s \\
3.\ \text{ServerCertificate} & s \longrightarrow c: & CERT(s, v_s) \\
4.\ \text{ClientCertificate} & c \longrightarrow s: & CERT(c, v_c) \\
5.\ \text{ClientKeyExchange} & c \longrightarrow s: & \{pm\}_{PK(s)} \\
6.\ \text{CertificateVerify} & c \longrightarrow s: & \{VH(Prev_5)\}_{SK(c)} \\
7.\ \text{ClientFinished} & c \longrightarrow s: & [PRF(mk, \text{“}cf\text{”}, MD5(Prev_6), SHA(Prev_6)), 0]_{cm,ce} \\
8.\ \text{ServerFinished} & s \longrightarrow c: & [PRF(mk, \text{“}sf\text{”}, MD5(Prev_7), SHA(Prev_7)), 0]_{sm,se}
\end{array}
$$

where $[M, n]_{mac,enc}$ is a record layer message that has the form:

$$[M, n]_{mac,enc} \quad = \quad \{M, HMAC(mac, \{n, M\})\}_{enc}$$

and

$$
\begin{array}{rcl}
mk & := & PRF(pm, \text{“}ms\text{”}, \{r_c, r_s\}) \\
ke & := & PRF(mk, \text{“}ke\text{”}, \{r_c, r_s\}) \\
cm & := & Quarter(ke, 0) \\
sm & := & Quarter(ke, 1) \\
ce & := & PRF(Quarter(ke, 2), \text{“}ce\text{”}, \{r_c, r_s\}) \\
se & := & PRF(Quarter(ke, 3), \text{“}se\text{”}, \{r_c, r_s\})
\end{array}
$$

Note that we write free variables in small letters and functions in capital letters. We use $Prev_n$ to denote all previous n messages.

Removing hash functions The *Removing hash function* transformation replaces hash functions with their hashed content. We firstly replace each hash functions of the form $PRF(a, \text{“}string\text{”}, \text{“}b\text{”})$ with a hash function of the form $PRF_{string}(a, b)$ and each function of the form $Quarter(a, n)$ with $Q_n(a)$. Consequently, the $ClientFinished$ and $ServerFinished$ messages become as follows:

7. ClientFinished $c \longrightarrow s : \ [PRF_{cf}(mk, MD5(Prev_6), SHA(Prev_6)), 0]_{cm,ce}$
8. ServerFinished $s \longrightarrow c : \ [PRF_{sf}(mk, MD5(Prev_7), SHA(Prev_7)), 0]_{sm,se}$

The keying material become as follows:

$$\begin{array}{lcl} mk & := & PRF_{ms}(pm, r_c, r_s) \\ ke & := & PRF_{ke}(mk, r_c, r_s) \\ cm & := & Q_0(ke) \\ sm & := & Q_1(ke) \\ ce & := & PRF_{ce}(Q_2(ke), r_c, r_s) \\ se & := & PRF_{se}(Q_3(ke), r_c, r_s) \end{array}$$

Now we use the transformation to remove hashes that are included in the contents of other hashes in the TLS protocol. As a result, the hash functions MD5 and SHA are removed since they are included inside the pseudo random function PRF. Similarly, the keying material MK and KE are replaced with their contents since they are included in other hashes as well. The protocol becomes as follows:

1. ClientHello $c \longrightarrow s : \ r_c$
2. ServerHello $s \longrightarrow c : \ r_s$
3. ServerCertificate $s \longrightarrow c : \ CERT(s, v_s)$
4. ClientCertificate $c \longrightarrow s : \ CERT(c, v_c)$
5. ClientKeyExchange $c \longrightarrow s : \ \{pm\}_{PK(s)}$
6. CertificateVerify $c \longrightarrow s : \ \{VH(Prev_6)\}_{SK(c)}$
7. ClientFinished $c \longrightarrow s : \ [PRF_{cf}(pm, r_c, r_s, Prev_6, Prev_6), 0]_{cm,ce}$
8. ServerFinished $s \longrightarrow c : \ [PRF_{sf}(pm, R_C, R_S, Prev_7, Prev_7), 0]_{sm,se}$

where $[M, n]_{mac,enc}$ is a record layer message that has the form:

$$
\begin{aligned}
[M,n]_{mac,enc} &= \{M, HMAC(mac, \{n, M\})\}_{enc} \\
mk &:= pm, r_c, r_s \\
ke &:= mk, r_c, r_s \\
CM &:= Q_0(ke) \\
SM &:= Q_1(ke) \\
CE &:= PRF_{ce}(Q_2(ke), r_c, r_s) \\
SE &:= PRF_{se}(Q_3(ke), r_c, r_s)
\end{aligned}
$$

Coalescing fields The *Coalescingatoms* transformation coalesces pairs of fields replacing them by the first mainly to remove redundancy in the protocol. We use this transformation to remove duplicate values in the protocol. In the *ClientFinished* and *ServerFinished* messages, $Prev_7$ and $Prev_8$ are duplicated as a result of the *Removing hashes* transformation. Similarly, r_c and r_s are duplicated since they occur at the beginning of $prev_8$ and $prev_9$. Also, when substituting for mk in ke, r_c and r_s are repeated. Removing all these duplicate values, the protocol becomes as follows:

1. ClientHello	$c \longrightarrow s:$	r_c
2. ServerHello	$s \longrightarrow c:$	r_s
3. ServerCertificate	$s \longrightarrow c:$	$CERT(s, v_s)$
4. ClientCertificate	$c \longrightarrow s:$	$CERT(c, v_c)$
5. ClientKeyExchange	$c \longrightarrow s:$	$\{pm\}_{PK(s)}$
6. CertificateVerify	$c \longrightarrow s:$	$\{VH(Prev_5)\}_{SK(c)}$
7. ClientFinished	$c \longrightarrow s:$	$[PRF_{cf}(pm, Prev_6), 0]_{cm,ce}$
8. ServerFinished	$s \longrightarrow c:$	$[PRF_{sf}(pm, Prev_7), 0]_{sm,se}$

where $[M,n]_{mac,enc}$ is a record layer message that has the form:

$$
\begin{aligned}
[M,n]_{mac,enc} &= \{M, HMAC(mac, \{n, M\})\}_{enc} \\
ke &:= pm, r_c, r_s \\
cm &:= Q_0(ke) \\
sm &:= Q_1(ke) \\
ce &:= PRF_{ce}(Q_2(ke), r_c, r_s) \\
se &:= PRF_{se}(Q_3(ke), r_c, r_s)
\end{aligned}
$$

Mapping functions We use the $Mappingfunctions$ transformation to remove some of the comlexity in the assignment of the keying material. We substitute for KE and use the following functional mapping:

$$\begin{aligned} Q_0(pm, r_c, r_s) &= G_0(pm, r_c, r_s) \\ Q_1(pm, r_c, r_s) &= G_1(pm, r_c, r_s) \\ PRF_{ce}(Q_2(pm, r_c, r_s), r_c, r_s) &= G_2(pm, r_c, r_s) \\ PRF_{se}(Q_3(pm, r_c, r_s), r_c, r_s) &= G_3(pm, r_c, r_s) \end{aligned}$$

In our analysis, we assume that the functions G_0, G_1, G_2, and G_3 are hash functions, proving this formally is left to the cryptographers. We also use functional mapping to define certificates. For simplicity, a certificate $Cert(A, B)$ is defined as follows:

$$CERT(a, b) = \{a, PK(a)\}_{SK(b)}$$

■

After applying the above transformations, the final simplified version of the TLS protocol is as follows:

$$\begin{array}{lll} \text{Message 1} & c \longrightarrow s: & r_c \\ \text{Message 2} & s \longrightarrow c: & r_s \\ \text{Message 3} & s \longrightarrow c: & \{s, PK(s)\}_{SK(v_s)} \\ \text{Message 4} & c \longrightarrow s: & \{c, PK(c)\}_{SK(v_c)} \\ \text{Message 5} & c \longrightarrow s: & \{pm\}_{PK(s)} \\ \text{Message 6} & c \longrightarrow s: & \{VH(Prev_6)\}_{SK(c)} \\ \text{Message 7} & c \longrightarrow s: & [PRF_{cf}(pm, Prev_7), 0]_{cm,ce} \\ \text{Message 8} & s \longrightarrow c: & [PRF_{sf}(pm, Prev_8), 0]_{sm,se} \end{array}$$

where $[M, n]_{mac,enc}$ is a record layer message that has the form:

$$\begin{aligned} [M, n]_{mac,enc} &= \{M, HMAC(mac, \{n, M\})\}_{enc} \\ cm &:= G_0(pm, r_c, r_s) \\ sm &:= G_1(pm, r_c, r_s) \\ ce &:= G_2(pm, r_c, r_s) \\ se &:= G_3(pm, r_c, r_s) \end{aligned}$$

This simplified version will be used for analysis in the subsequent sections.

3 The Strand Spaces Model, and its use in Modelling TLS

In this section, we describe the strand spaces model [THG98] and the extensions we made to the original model to facilitate the analysis of TLS. In addition, we explain how TLS can be specified in the extended model. We also describe some of the techniques and results of the original model, and state their correctness in the extended model; in particular, we describe authentication tests [GT01, DGT07b], which are used extensively in our analysis of TLS in the next section. Finally, we discuss multi-protocol interaction within the strand spaces setting [GT00b]. Such discussion is needed to explain some of the assumptions we make in the next section to solve the problem of multi-layer interaction.

3.1 Basics of Strand Spaces

The Strand Spaces Model was developed by Thayer *et al.* [THG98] to reason about the correctness of security protocols. In this section we describe the basics of the model and the extensions we made in order to represent the cryptographic operations used in TLS. Most of the definitions of this section are taken from [THG98] and [GT01].

3.1.1 The Term Algebra

Let $\mathcal{A}$ be the set of possible messages that can be exchanged between principals in a protocol. The elements of $\mathcal{A}$ are usually referred to as *terms*. In the original Strand Spaces model [THG98], $\mathcal{A}$ is freely generated from two disjoint sets, $\mathcal{T}$ (representing *tags*, *texts*, *nonces*, and *principals*) and $\mathcal{K}$ (representing keys) by means of *concatenation* and *encryption*.

Keys can be generated using a *key generation functions* from some set kgf. Each $G \in \mathsf{kgf}$ generates keys from atoms: $G : \mathcal{T} \times \mathcal{T} \ldots \times \mathcal{T} \rightarrow \mathcal{K}$. We assume each key generation function is injective (i.e. collision-free), and distinct key generation functions have disjoint ranges. If a key $k \in \bigcup\{\mathsf{ran}(G) \mid G \in \mathsf{kgf}\}$, we say that k is *complex*; otherwise, k is *simple*. If $k = G(t_1, \ldots, t_n)$ then we say that $t_1, \ldots, t_n$ are *ingredients* of k.

To provide a mathematical model of TLS, we specialize the term algebra $\mathcal{A}$. The set $\mathcal{K}$ of keys used in TLS is partitioned into four sets: the set of public keys, $\mathcal{K}_{Pub}$; the set of secret keys, $\mathcal{K}_{Sec}$; the set of MAC keys,

$\mathcal{K}_{MAC} \subset \bigcup\{\mathsf{ran}(G) \mid G \in \mathsf{kgf}\}$; and the set of symmetric encryption keys, $\mathcal{K}_{Sym} \subset \bigcup\{\mathsf{ran}(G) \mid G \in \mathsf{kgf}\}$.

The set $\mathcal{K}$ of keys is equipped with a unary injective symmetric operator $\mathsf{inv} : \mathcal{K} \to \mathcal{K}$; $\mathsf{inv}(k)$ is usually denoted k^{-1}. We assume $k \in \mathcal{K}_{Pub}$ iff $k^{-1} \in \mathcal{K}_{Sec}$, and if $k \in \mathcal{K}_{Sym}$ then $k^{-1} = k$.

Let $\mathcal{T}_{name} \subseteq \mathcal{T}$ be the set of agent names. The functions $PK : \mathcal{T}_{name} \to \mathcal{K}_{Pub}$ and $SK : \mathcal{T}_{name} \to \mathcal{K}_{Sec}$ are injective mappings to associate each principal with a public key and a secret key respectively such that $\forall a : \mathcal{T}_{name} \bullet (PK(a))^{-1} = SK(a)$.

We adopt the following conventions on variables. Variables c, s, v_s, v_c, and ca range over $\mathcal{T}_{name}$; k (or the same letter decorated with subscripts) ranges over $\mathcal{K}_{Pub}$; ce and se range over $\mathcal{K}_{Sym}$ while cm and sm range over $\mathcal{K}_{MAC}$; h ranges over the set $Hash$ of hash functions, and G ranges over the set kgf of key generating functions; r, pm (or the same variables decorated with subscripts) range over $\mathcal{T}$ and are used as fresh values; $prev_n$ refers to the sequence of the n previous messages. Note that terms like r_c, k_c are just variables and have no trusted relation to the agent c. On the other hand, a term like $PK(c)$ is the result of applying the function PK to the argument c and therefore, it reliably refers to the public key of c.

The TLS protocol uses one-way hash functions to achieve two distinct purposes: to digest messages and to generate keys. We discussed the latter above. We model digesting as a *constructor* over the term algebra, in addition to the standard constructors of encryption and concatenation:

Definition 3.1 *Compound terms are built by three constructors:*

- $\mathsf{encr} : \mathcal{K} \times \mathcal{A} \to \mathcal{A}$ *representing encryption.*
- $\mathsf{join} : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ *representing concatenation.*
- hash : *Hash* $\times \mathcal{A} \to \mathcal{A}$ *representing hashing to digest messages, where Hash is a set of hash functions, and such that* $\mathbf{ran}\ (\mathsf{hash}) \subseteq (\mathcal{A} \setminus \mathcal{K})$.

Conventionally, $\{t\}_k$ is used to indicate that a term t is encrypted with a key K and $t_0 \hat{} t_1$ to denote the concatenation of t_0 and t_1.

Note that, while we define the hash operation as a constructor, we assume that key generation functions $G \in \mathsf{kgf}$ are functions that generate atoms (keys). This assumption seems to be justified since *complex* keys are used as atoms rather than as hashed terms after being constructed. The need to

distinguish between hashing and key generation will become clearer when we explain the Normal Form Lemma in Section 3.3.

To model TLS, we define: the hash functions $VH, PRF_{cf}, PRF_{sf}, HMAC \in Hash$; the key generation functions $G_0, G_1 \in \mathsf{kgf}$ such that their ranges are subsets of $\mathcal{K}_{MAC}$; and the key generation functions $G_2, G_3 \in \mathsf{kgf}$ such that their ranges are subsets of $\mathcal{K}_{Sym}$.

We now define some basic concepts closely related to the term algebra.

Definition 3.2 *The subterm relation $\sqsubseteq$ is defined inductively, as the least reflexive transitive relation such that:*

- $r \sqsubseteq r$;
- $r \sqsubseteq \{t\}_k$ *if* $r \sqsubseteq t$;
- $r \sqsubseteq t_0 \hat{} t_1$ *if* $r \sqsubseteq t_0$ *or* $r \sqsubseteq t_1$;
- $r \sqsubseteq \mathsf{hash}(h, t)$ *if* $r \sqsubseteq t$.

Definition 3.3 *[GT01]*

1. *If $\Re \subset \mathcal{K}$, then $t' \sqsubseteq_\Re t$ if t is in the smallest set containing t' and closed under encryption with $K \in \Re$ and concatenation with arbitrary terms.*
2. *A term t' is a component of t if t' is not of the form $t_0 \hat{} t_1$ (so is an atomic value, a hash value, or an encryption), and $t' \sqsubseteq_{\{\}} t$, i.e. t is constructed by concatenating t' with arbitrary terms.*

3.1.2 Strands, Nodes, and Bundles

A participant in a protocol can either send or receive terms. In the Strand Spaces model, a positive term is used to represent a transmission while a negative term is used to denote reception. A *strand* is a sequence of message transmissions and receptions. A *strand space* is a set of strands.

Definition 3.4 *[THG98]*

A signed term is a pair $\langle \sigma, a \rangle$ with $\sigma \in \{+, -\}$ and $a \in \mathcal{A}$. A signed term is written as $+t$ or $-t$. $(\pm\mathcal{A})^*$ is the set of finite sequences of signed terms. A typical element of $(\pm\mathcal{A})^*$ is denoted by $\langle\langle \sigma_1, a_1 \rangle, ..., \langle \sigma_n, a_n \rangle\rangle$. A strand space over $\mathcal{A}$ is a set Σ with a trace mapping $tr : \Sigma \to (\pm\mathcal{A})^*$. Fix a strand space Σ.

1. *A* node *is a pair $\langle st, i\rangle$, with $st \in \Sigma$ and i an integer satisfying $1 \leq i \leq length(tr(st))$. The set of nodes is denoted by $\mathcal{N}$. We will say the node $\langle st, i\rangle$ belongs to the strand st.*

2. *There is an edge $n_1 \rightarrow n_2$ if and only if $msg(n_1) = +a$ and $msg(n_2) = -a$ for some $a \in \mathcal{A}$. The edge means that node n_1 sends the message a, which is received by n_2, recording a potential causal link between those strands.*

3. *When $n_1 = \langle st, i\rangle$, and $n_2 = \langle st, i+1\rangle$ are members of $\mathcal{N}$, there is an edge $n_1 \Rightarrow n_2$. The edge expresses that n_1 is an immediate causal predecessor of n_2 on the strand s. $n' \Rightarrow^+ n$ is used to denote that n' precedes n (not necessarily immediately) on the same strand.*

4. *An unsigned term t* occurs in *$n \in \mathcal{N}$ iff $t \sqsubseteq msg(n)$.*

5. *Suppose $\mathcal{I}$ is a set of unsigned terms. The node $n \in \mathcal{N}$ is an* entry point *for $\mathcal{I}$ iff $msg(n) = +t$ for some $t \in \mathcal{I}$, and whenever $n' \Rightarrow^+ n$, $msg(n') \notin \mathcal{I}$.*

6. *An unsigned term t* originates *on $n \in \mathcal{N}$ iff n is an entry point for the set $\mathcal{I} = \{t' \mid t \sqsubseteq t'\}$.*

7. *An unsigned term t is* uniquely originating *in a set of nodes $S \subset \mathcal{N}$ iff there is a unique $n \in S$ such that t originates on n. (A term originating uniquely in a set of nodes can play the role of a nonce or a session key in that structure.)*

8. *An unsigned term t is* non-originating *in a set of nodes $S \subset \mathcal{N}$ iff there is no $n \in S$ such that t originates on n. (Long term keys are normally non-originating.)*

9. *A term t is new at a node n if t is a component of $msg(n)$ and for every $n_0 \Rightarrow^+ n$, t is not a component of $msg(n_0)$.*

In order to define strands for the roles of the TLS Protocol, we define a function *RECMS* that models the changes the Record Layer applies to Transaction Layer messages such as sequence numbering, MAC application, and encryption. The function takes as its inputs the MAC key of the sending agent, the encryption key of the sending agent, the MAC key of the receiving

agent, the encryption key of the receiving agent, and the sequence Sms of messages sent and received in the Transaction Layer payload. A sequence number is assigned to each message in Sms such that sent and received messages are ordered in two separate streams.

Definition 3.5 *$RECMS : (\mathcal{K}_{MAC} \times \mathcal{K}_{Sym} \times \mathcal{K}_{MAC} \times \mathcal{K}_{Sym}) \rightarrow (\pm\mathcal{A})^* \rightarrow (\pm\mathcal{A})^*$ is defined as follows:*

$$
\begin{array}{l}
RECMS\ \ (k_{s1}, k_{s2}, k_{r1}, k_{r2})\ Sms = RECMS'\ \ (k_{s1}, k_{s2}, k_{r1}, k_{r2})\ (1,1)\ Sms, \\
RECMS'\ \ (k_{s1}, k_{s2}, k_{r1}, k_{r2})\ (i,j)\ \langle\rangle = \langle\rangle, \\
RECMS'\ \ (k_{s1}, k_{s2}, k_{r1}, k_{r2})\ (i,j)\ (\langle(+m,i)\rangle \frown Sms) = \\
\qquad \langle +[m,i]_{k_{s1},k_{s2}}\rangle \frown RECMS'\ \ (k_{s1}, k_{s2}, k_{r1}, k_{r2})\ (i+1,j)\ Sms, \\
RECMS'\ \ (k_{s1}, k_{s2}, k_{r1}, k_{r2})\ (i,j)\ (\langle(-m,i)\rangle \frown Sms) = \\
\qquad \langle -[m,i]_{k_{r1},k_{r2}}\rangle \frown RECMS'\ \ (k_{s1}, k_{s2}, k_{r1}, k_{r2})\ (i,j+1)\ Sms, \\
[M,n]_{mac,enc} = \{M, HMAC(mac, \{n, M\})\}_{enc}.
\end{array}
$$

As explained in Section 2.2, the TLS protocol has two primary roles, the client c and the server s, and two secondary roles of certificate authorities v_s and v_c. The certificate authorities' strands are necessary for the origination of public key certificates. We define a strand for each of these roles.

Definition 3.6 *Let $Client[c, s, r_c, r_s, k_s, v_s, v_c, pm, Sms]$ be the set of client strands whose trace is:*

$$
\begin{array}{ll}
\langle + & r_c, \\
- & r_s, \\
- & \{k_s \hat{} s\}_{SK(v_s)}, \\
+ & \{PK(c) \hat{} c\}_{SK(v_c)}, \\
+ & \{pm\}_{k_s}, \\
+ & \{VH(prev_5)\}_{SK(c)}, \\
+ & [PRF_{cf}(pm \hat{} prev_6), 0]_{cm,ce}, \\
- & [PRF_{sf}(pm \hat{} prev_7), 0]_{sm,se}\ \rangle \\
\frown & RECMS\ \ (cm, ce, sm, se)\ Sms.
\end{array}
$$

Definition 3.7 *Let $Server[s, c, r_c, r_s, v_s, v_c, k_c, pm, Sms]$ be the set of server*

strands whose trace is:

$$\begin{array}{rl} \langle - & r_c, \\ + & r_s, \\ + & \{PK(s)\hat{}s\}_{SK(v_s)}, \\ - & \{k_c\hat{}c\}_{SK(v_c)}, \\ - & \{pm\}_{PK(s)}, \\ - & \{VH(prev_5)\}_{k_c^{-1}}, \\ - & [PRF_{cf}(pm\hat{}prev_6), 0]_{cm,ce}, \\ + & [PRF_{sf}(pm\hat{}prev_7), 0]_{sm,se} \rangle \end{array}$$
$$\hat{}RECMS\ (sm, se, cm, ce)\ Sms.$$

Definition 3.8 *Let $CA[ca, a]$ be the set of certificate authority strands whose trace is:*

$$\langle + \{PK(a)\hat{}a\}_{SK(ca)} \rangle.$$

For simplicity, we assume that the public key certificates originate in certificate authority strands and are obtained by the client and server strands before the Handshake exchange.

When talking about collections of strands, we will sometimes use an asterisk ($*$) as a wild-card; for example, $Client[c, s, r_c, r_s, k_s, v_s, v_c, *, *]$ is shorthand for $\bigcup_{pm,Sms} Client[c, s, r_c, r_s, k_s, v_s, v_c, pm, Sms]$. We shall refer to regular primary nodes according to their position in a client or a server strand; for example writing $Client_1$ for the second node in a client strand (where the strand is clear from the context).

The set $\mathcal{N}$ of nodes together with both sets of edges $n_1 \rightarrow n_2$ and $n_1 \Rightarrow n_2$ forms a directed graph $\langle \mathcal{N}, (\rightarrow \cup \Rightarrow) \rangle$. A *bundle* is a finite subgraph of $\langle \mathcal{N}, (\rightarrow \cup \Rightarrow) \rangle$ for which we can regard the edges as expressing the causal dependencies of the nodes.

Definition 3.9 *[THG98] Suppose $\rightarrow_{\mathcal{B}} \subset \rightarrow$, $\Rightarrow_{\mathcal{B}} \subset \Rightarrow$, and $\mathcal{B} = \langle \mathcal{N}_{\mathcal{B}}, (\rightarrow_{\mathcal{B}} \cup \Rightarrow_{\mathcal{B}}) \rangle$ is a subgraph of $\langle \mathcal{N}, (\rightarrow \cup \Rightarrow) \rangle$. $\mathcal{B}$ is a bundle if (1) $\mathcal{N}_{\mathcal{B}}$ and $(\rightarrow_{\mathcal{B}} \cup \Rightarrow_{\mathcal{B}})$ are finite; (2) If $n_2 \in \mathcal{N}_{\mathcal{B}}$ and $msg(n_2)$ is negative, then there is a unique n_1 such that $n_1 \rightarrow_{\mathcal{B}} n_2$; (3) If $n_2 \in \mathcal{N}_{\mathcal{B}}$ and $n_1 \Rightarrow n_2$ then $n_1 \Rightarrow_{\mathcal{B}} n_2$; and (4) $\mathcal{B}$ is acyclic.*

Figure 2 illustrates a bundle representing a single run of the Handshake Protocol.

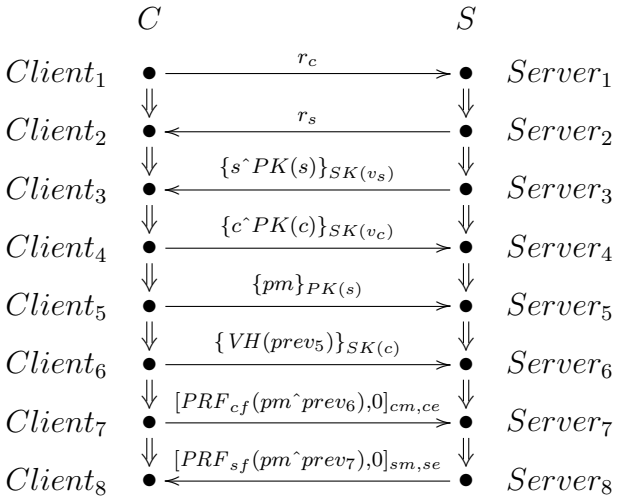


Figure 2: A bundle representing a single execution of the Handshake Protocol.

Definition 3.10 *[THG98] A node n is in a bundle $\mathcal{B} = \langle \mathcal{N}_\mathcal{B}, \rightarrow_\mathcal{B} \cup \Rightarrow_\mathcal{B} \rangle$, written $n \in \mathcal{B}$, if $n \in \mathcal{N}_\mathcal{B}$; a strand st is in $\mathcal{B}$ if all of its nodes are in $\mathcal{N}_\mathcal{B}$. The $\mathcal{B}$-height of a strand st is the largest i such that $\langle st, i \rangle \in \mathcal{B}$.*

For example, the bundle height of each strand in Figure 2 is 8.

Proposition 3.11 *[THG98] Let $\mathcal{B}$ be a bundle. Then $\preceq_\mathcal{B}$ is a partial order, i.e. a reflexive, antisymmetric, transitive relation. Every non-empty subset of the nodes in $\mathcal{B}$ has a $\preceq_\mathcal{B}$-minimal member.*

$\preceq_\mathcal{B}$ can be considered as a causal precedence relationship, because $n \prec_\mathcal{B} n'$ holds iff it is possible to get from n to n' by following zero or more $\rightarrow_\mathcal{B}$ or $\Rightarrow_\mathcal{B}$ steps, and therefore n's occurrence causally contributes to the occurrence of n' [THG98].

3.1.3 The Penetrator

We now define the powers of the penetrator in the extended algebra.

The powers of the penetrator (commonly referred to as the intruder) are characterised by two ingredients [THG98]:

- The set $\mathcal{K}_P$ of keys known initially to the penetrator.
- The set of strands that allow the penetrator to generate new messages from the atomic messages $\mathcal{T}_P$ he knows initially, and the messages he intercepts. This provides the penetrator with the powers specified by the Dolev-Yao model [DY83].

The set of penetrator strands is extended to reflect the extensions we made to the term algebra as follows:

Definition 3.12 *A penetrator trace is one of the following:*

M. *Text message:* $\langle +r \rangle$ *where* $r \in \mathcal{T}_P$.

K. *Key:* $\langle +k \rangle$ *where* $k \in \mathcal{K}_\mathcal{P}$.

C. *Concatenation:* $\langle -t_0, -t_1, +t_0\hat{}t_1 \rangle$.

S. *Separation into components:* $\langle -t_0\hat{}t_1, +t_0, +t_1 \rangle$.

E. *Encryption:* $\langle -k, -t, +\{t\}_k \rangle$ *where* $k \in \mathcal{K}$.

D. *Decryption:* $\langle -k^{-1}, -\{t\}_k, +t \rangle$ *where* $k \in \mathcal{K}$.

H. *Hashing:* $\langle -t, +\mathsf{hash}(h,t) \rangle$ *where* $h \in Hash$.

KG. *Key generation:* $\langle -r_1, \ldots, -r_n, +G(r_1, \ldots, r_n) \rangle$, *where* $r_1, \ldots, r_n \in \mathcal{T}$ *and* $G \in \mathsf{kgf}$.

A node is referred to as a penetrator node if it lies on a penetrator strand; otherwise it is called a *regular* node. An infiltrated TLS strand space contains the regular strands defined previously in addition to the above penetrator strands.

Since the behaviour of the penetrator is only specified by the above traces, it may vary arbitrarily between bundles that have the same regular strands. Such bundles are still considered equivalent. Bundle equivalence is defined formally as follows [GT01]:

Definition 3.13 *Bundles* $\mathcal{B}$, $\mathcal{B}'$ *on a strand space* Σ *are equivalent iff they have the same regular nodes.*

3.2 Paths and well behaved bundles

In previous section, we specify that the penetrator can perform several operations represented by strands. In addition, these operations can be carried out in any order. Furthermore, this is complicated by the unbounded number of protocol sessions. Consequently, the presence of the penetrator can make the problem of security protocols analysis undecidable.

In [GT01, GT00a], Thayer *et al.* restrict the order of the penetrator strands in order to assist the analysis of security protocols. There are two elements in this restriction: a normal form lemma and an efficiency condition [GT01]. In this section we restrict the order in which the penetrator performs his/her actions using the extended penetrator model developed in the previous section. We follow closely the reasoning of Thayer *et al.* in [GT01] and therefore complete proofs are only provided when necessary.

3.3 The Normal Form Lemma

Firstly, we remove some of the redundancies in the penetrator's behaviour without weakening his/her powers. With respect to the penetrator's model, there are two types of redundancies [GT01]:

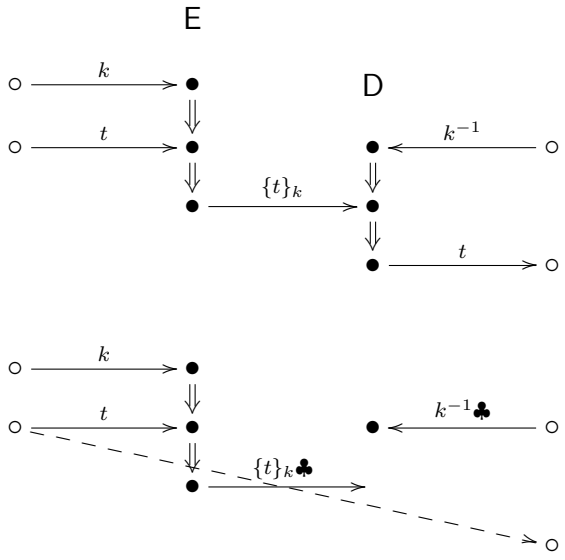
1. E-D redundancies: Here the penetrator encrypts a value h with a key K, and then decrypts with the corresponding key K^{-1}. This type of redundancy can be eliminated as shown in Figure 3.

2. C-S redundancies: Here the penetrator concatenates two values h and g to form $g\hat{}h$ and then separates the concatenated term into its subterms. This type of redundancy can be eliminated as shown in Figure 4.

By eliminating the above redundancies we end up with an equivalent bundle (Definition 3.13) since we only remove penetrator nodes. We may infer:

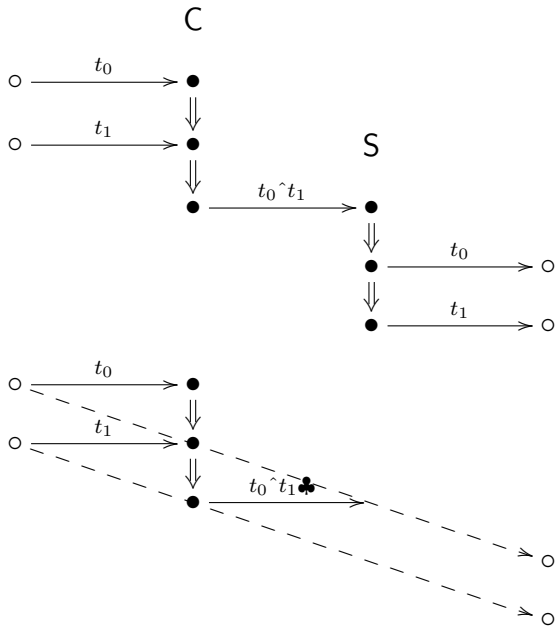
Proposition 3.14 *Every bundle is equivalent to a bundle with no redundancies of type* $\mathsf{C}-\mathsf{S}$ *and* $\mathsf{E}-\mathsf{D}$. *(As Proposition 2 in [GT01].)*

The penetrator's activity is formalised in strand spaces by the notion of a *path* [GT01]. A path p through the bundle $\mathcal{B}$ is any finite sequence of nodes and edges where the notation $m \longmapsto n$ means either $m \Rightarrow^{+} n$ with $msg(m)$ negative and $msg(n)$ positive, or else $m \rightarrow n$. The ith node of a path p is



♣ Discarded messages

Figure 3: E-D redundancies and how to eliminate them [GT01].



♣ Discarded messages

Figure 4: C-S redundancies and how to eliminate them [GT01].

referred to as p_i while $\mid p \mid$ refers to the length of p and $\ell(p)$ is used to denote $p_{|p|}$, i.e. the last node in p. A *penetrator path* is one in which all nodes other than possibly the first or the last node are penetrator nodes.

A penetrator's activity is assumed to follow a specific pattern to achieve one of two purposes [Gut01]: making a key available for a D or E strand, or constructing some message to deliver to a regular node.

In order to formalise this pattern, we use the notion of *constructive* and *destructive* edges. Our definition of constructive and destructive edges is slightly different from the one provided in [GT01] since we refer to the extended penetrator model in definition 3.12.

Definition 3.15 *$A \Rightarrow^+ -$edge is constructive if it is a part of a E, C, or H strand. It is destructive if it is part of a D, S, or KG strand. A penetrator node is initial if it is a K or M node.*

Although the strands H and KG are quite similar since they both use hash functions, we define H to be constructive and the hash-to-generate key strands, i.e. KG, to be destructive. The reason for this will become clear when we discuss the Normal Form Lemma.

Proposition 3.16 *In a bundle, a constructive edge immediately followed by a destructive edge has one of the following three forms:*

1. *Part of a $\mathsf{E}_{h,k}$ immediately followed by part of a $\mathsf{D}_{h,k}$ strand for some h, k*

2. *Part of a $\mathsf{C}_{h,k}$ immediately followed by part of a $\mathsf{S}_{h,k}$ strand for some h, k*

PROOF. This result requires the freeness of the message algebra. It also follows from the fact that hash functions are uninvertible. Note that this result will not be true if we define edges that are part of KG strands to be constructive since these strands can generate keys that can be used as key edges for D strands (See Figure 5) in which case a constructive edge should be followed by a destructive edge and the following Normal Form Lemma will not hold.

Before discussing the Normal Form Lemma, we provide the definition of *normal* bundles [GT01].

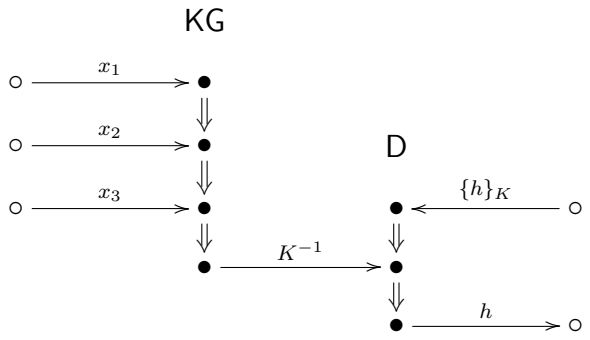


Figure 5: Considering KG strands as constructive, the Normal Form Lemma would not hold for the above bundle.

Definition 3.17 *A bundle $\mathcal{B}$ is normal if for any penetrator path of $\mathcal{B}$, every destructive edge precedes every constructive edge.*
(As Definition 5 in [GT01].)

Proposition 3.18 (**Penetrator Normal Form Lemma**) *For every bundle $\mathcal{B}$ there exists an equivalent normal bundle $\mathcal{B}'$.*

Proof. The proof is the same as the one provided in [GT01] and we just include it here to show that it still holds in the extended model. By Proposition 3.14 every bundle has an equivalent bundle that has no redundancies of type $\mathsf{C}-\mathsf{S}$ and $\mathsf{E}-\mathsf{D}$. Let us assume for a contradiction that such an equivalent bundle is not normal i.e. contains a constructive edge that precedes a destructive edge. By Proposition 3.16, this should be either a $\mathsf{C}-\mathsf{S}$ or an $\mathsf{E}-\mathsf{D}$ edge. However, this is not possible since the bundle has no redundancies. Therefore, every bundle has an equivalent normal bundle that has no redundancies of type $\mathsf{C}-\mathsf{S}$ or $\mathsf{E}-\mathsf{D}$.

The Normal Form Lemma is the first element in restricting the penetrator's order of actions. In the following section we discuss the second element of this restriction, which is the concept of *efficient* bundles.

3.4 Efficient bundles

As explained before in section 3.1.2, a component of t is a subterm t_0 which is either an atomic value, an encryption, or a hash value, and such that t_0 can be

obtained from t by repeatedly separating concatenations. Components can only be changed through cryptographic work. An efficient bundle restricts the penetrator to make the most with the available components rather than using additional regular nodes.

Definition 3.19 *A bundle is efficient iff for every node m and negative penetrator node n, if every component of n is a component of m, then there is no regular node m' such that $m \prec m' \prec n$. (As Definition 10 in [GT01].)*

The following proposition states that every bundle $\mathcal{B}$ has an equivalent efficient bundle $\mathcal{B}'$ without interfering with the Normal Form Lemma. For a detailed proof of the proposition, the reader is referred to [GT01].

Proposition 3.20 *For every bundle $\mathcal{B}$, there is an equivalent efficient bundle $\mathcal{B}'$. If a bundle is efficient, then it has an equivalent normal bundle which is also efficient. (As Proposition 14 in [GT01].)*

3.5 Specifying Authentication and Secrecy Goals

A regular strand represents a principal's local view of a protocol execution, i.e. the messages sent and received by that principal. Security protocol goals are inferences that principal can make about the strands of other principals and the behaviour of the penetrator [Gut01].

An authentication goal is an inference about what another regular principal must have done. This inference has four elements [Gut01]:

- The principal's strand: The messages sent and received by a principal are the principal's source of knowledge about what has happened so far in the protocol.
- The specification of the protocol: These describe the behaviour of the regular strands.
- The penetrator powers: These define what the penetrator can or cannot do.
- Origination assumptions: The unique origination of nonces and session keys and the non-origination of long-term secrets are vital to prove authentication goals.

From these elements, the causal laws included in the definition of bundles (Definition 3.9) are used to infer the traces of other strands. The results of authentication inferences take the form [Gut01]: for all bundles $\mathcal{B}$ and all strands st, there exists a strand st' such that

> **if** $st \in R$ has $\mathcal{B}$-height i,
> **and** some origination assumptions hold,
> **then** $st' \in R'$ and st' has $\mathcal{B}$-height j,

where R and R' are sets of roles. Such an authentication property is invariant under bundle equivalence, since it asserts that certain regular nodes must be present in bundles regardless of the presence of penetrator nodes [GT01].

To illustrate an authentication goal, consider a bundle $\mathcal{B}$ in a TLS strand space. Then the Handshake goal of authenticating the client to the server can be captured as follows: if st_c is a client strand in $Client[c, s, r_c, r_s, k_s, v_s, v_c, pm, *]$ of $\mathcal{B}$-height at least 8, pm is uniquely originating on $\langle st_c, 5\rangle$, and k_s is non-originating in $\mathcal{B}$, then there exists a server strand $st_s \in Server[s, c, r_c, r_s, PK(c), v_s, v_c, pm, *]$ of $\mathcal{B}$-height at least 8.

On the other hand, a secrecy goal is an inference about what the penetrator cannot have done. Secrecy goals are verified by proving, following some origination assumptions, that terms that are intended to remain secret are not said in public in any bundle. It follows that the penetrator could not have derived them because if he did, then there would exist a bundle in which he also utters them. Secrecy inferences take the form [Gut01]: for all bundles $\mathcal{B}$ and all regular strands st

> **if** $st \in R$ has $\mathcal{B}$-height i,
> **and** some origination assumptions hold,
> **then** there is no node $n \in \mathcal{B}$ such that $msg(n) = t$.

The term t is either a parameter of R, or a term whose ingredients are parameters of R. If the above secrecy property holds in all bundles equivalent to $\mathcal{B}$, then the value remains secret because the penetrator is unable to derive it without further cooperation from regular strands [GT01]. For example the secrecy of the premaster secret can be specified in the following form: for all bundles $\mathcal{B}$, if st_c is a client strand in $Client[c, s, r_c, r_s, k_s, v_s, v_c, pm, *]$ with $\mathcal{B}$-height at least 8, pm is uniquely originating on $\langle st_c, 5\rangle$, and $SK(c), SK(s)$ are non-originating in $\mathcal{B}$, then for all nodes $n \in \mathcal{B}$, $msg(n) \neq pm$.

3.6 Proving Secrecy and Authentication

In this section, we discuss tests to prove secrecy and authentication within the strand spaces setting. We start by defining the set of *safe* keys that are not available to the penetrator. We then present authentication tests.

3.6.1 Safe and Penetrable Keys

We can distinguish between two sets of keys: the set of *penetrable* keys that may become known to the penetrator, and the set of *safe* keys whose secrecy is preserved.

We begin by defining some concepts that we will use. We say that a term t_0 *occurs only within* a set of terms R in t if in the abstract syntax tree of t, every path from the root to an occurrence of t_0 traverses some $t_1 \in \mathsf{R}$ before reaching t_0 [DGT07a]; note that this includes the possibility that t_0 does not occur at all in t. More formally:

Definition 3.21 *We say that t_0 occurs only within R in t, where R is a set of terms, if:*

1. *$t_0 \not\sqsubseteq t$; or*
2. *$t \in \mathsf{R}$; or*
3. *$t \neq t_0$ and either (a) $t = \{t_1\}_k$ and t_0 occurs only within R in t_1; or (b) $t = \mathsf{hash}(h, t_1)$ and t_0 occurs only within R in t_1; or (c) $t = t_1 \hat{} t_2$ and t_0 occurs only within R in t_1 and t_2.*

We say that t_0 occurs only within R in $\mathcal{B}$, if for every positive regular node $n \in \mathcal{B}$, and for every new component t of n, t_0 occurs only within R in t.

On the other hand, t_0 *occurs outside* R in t if t_0 does not occur only within R in t. This means that $t_0 \sqsubseteq t$ and there is a path from the root to an occurrence of t_0 as a subterm of t that traverses no $t_1 \in R$ [DGT07a].

We now inductively define the set $\mathcal{P}(\mathcal{B})$ of penetrable keys, as in [Gut01]. We expand the original definition to include the complex keys that can be generated from messages known initially to the penetrator ($X_0(\mathcal{B})$, below). Further, it includes the keys that can be derived from encrypted terms using penetrable decryption keys ($Y_{i+1}(\mathcal{B})$, below); it also includes the complex keys whose ingredients are derivable from encrypted terms using penetrable decryption keys, and consequently can be derived by the penetrator using KG-strands ($X_{i+1}(\mathcal{B})$, below).

Definition 3.22 *Let $\mathcal{B}$ be a bundle, and let $P_i^{-1}(\mathcal{B}) = \{k^{-1} \mid k \in P_i(\mathcal{B})\}$.*

Let $\mathcal{P}_0(\mathcal{B}) = \mathcal{K}_p \cup X_0(\mathcal{B})$ where $k \in X_0(\mathcal{B})$ iff k is complex and for every ingredient r of k, $r \in \mathcal{T}_p$.

Let $\mathcal{P}_{i+1}(\mathcal{B}) = \mathcal{P}_i(\mathcal{B}) \cup X_{i+1}(\mathcal{B}) \cup Y_{i+1}(\mathcal{B})$, where:

- *$k \in X_{i+1}(\mathcal{B})$ iff k is complex and for every ingredient r of k, there exists a positive regular node $n \in \mathcal{B}$ and a term t such that t is a new component of n, $r \sqsubseteq_{P_i^{-1}(\mathcal{B})} t$;*
- *$k \in Y_{i+1}(\mathcal{B})$ iff there exists a positive regular node $n \in \mathcal{B}$ and a term t such that t is a new component of n, and $k \sqsubseteq_{P_i^{-1}(\mathcal{B})} t$.*

We define $\mathcal{P}(\mathcal{B}) = \bigcup_i \mathcal{P}_i(\mathcal{B})$. When $k \in \mathcal{P}(\mathcal{B})$, we say that k is penetrable.

As in [Gut01], Definition 3.22 is justified by proving that any penetrable key that becomes available to the penetrator in any bundle $\mathcal{B}$ is in fact a member of $\mathcal{P}(\mathcal{B})$:

Proposition 3.23 *Let $\mathcal{B}$ be a bundle with $n \in \mathcal{B}$ and $msg(n) = K$. Then $K \in \mathcal{P}(\mathcal{B})$.*

Proof. By Propositions 3.18 and 3.20, we assume that $\mathcal{B}$ is normal and efficient. Our induction hypothesis is that, for all $n' \preceq_{\mathcal{B}} n$, $msg(n') \in \mathcal{K} \Rightarrow msg(n') \in \mathcal{P}(\mathcal{B})$.

Suppose K is a simple key; Let p be a path such that $\ell(p) = n$, K is simple, K originates at p_1, and $K \sqsubset msg(p_j)$ for $1 \leq j \leq \mid p \mid$ and therefore p does not traverse any key edges. We can distinguish between the following cases:

1. p_1 is a penetrator node. Hence, by definition, n is a K node and $\mid p \mid = 1$, in which case $K \in \mathcal{K}_p$ and by Definition 3.22, $K \in \mathcal{P}_0(\mathcal{B})$.
2. p_1 is a regular node. Therefore, there can be other regular nodes on p. Let p_λ be the last regular node on p and p' be the penetrator path from p_λ to $\ell(p)$. Since $msg(\ell(p))$ is an simple key and $\mathcal{B}$ is normal, p' can only traverse destructive edges. Since $K \sqsubset msg(p'_j)$ for $1 \leq j \leq \mid p' \mid$, p' can only traverse D-strands and S-strands. Moreover, since we assume that p traverses no key edges, $msg(p_j + 1) \sqsubseteq msg(p_j)$. Therefore, $K \sqsubseteq_{\aleph} msg(p'_1)$ where $\aleph$ contains K_1 whenever K_1^{-1} was used in a D-strand along p'. In other words $\aleph$ is a collection of keys previously penetrable, in which case $K \in \mathcal{P}_{i+1}(\mathcal{B})$ for some i.

For complex keys define p to be a path such that $\ell(p) = n$, K is complex and p does not traverse any key edges. We can distinguish between the following cases:

1. K originates at $\ell(p)$. Consequently, $\ell(p)$ can only be the last node of a KG-strand. By the definition of a KG-strand, every node n' immediately preceding $\ell(p)$ has an atomic term r such that r is an ingredient of K. Define a path p' for each such n' such that $\ell(p') = n'$, r originates at p'_1, and $r \sqsubset msg(p'_m)$ for $1 \leq m \leq \mid p' \mid$ and therefore each p' does not traverse any key edges. Applying the same argument for simple keys above, we conclude that each r is either known initially to the penetrator or else if p'_1 is a regular node $r \sqsubseteq_{\aleph} msg(n'')$ where n'' is the last regular node on p' and $\aleph$ contains K_1 whenever K_1^{-1} was used in a D-strand along p', in which case $K \in \mathcal{P}_{i+1}(\mathcal{B})$ for some i.

2. K originates at p_1. Consequently, we can define a path p similar to the path p we defined for simple keys and the same proof applies except that p_1 cannot be a penetrator node since K is complex.

□

We can now define the set $\mathcal{S}$ of safe keys, i.e., keys that the penetrator cannot obtain; note that safe keys and penetrable keys are disjoint. Again our definition differs from that provided in [Gut01] since it identifies when a complex key is safe. We first define the set of safe atoms (which includes complex keys), i.e., those atoms that the penetrator cannot obtain; we then restrict this to keys to find the safe keys. A safe atom: (1) is not initially known by the penetrator; (2) occurs only hashed or encrypted with the inverse of a safe key; and (3) if it is complex, every ingredient is safe.

Definition 3.24 *Let $\mathcal{M}(\mathcal{B})$ be the set of safe atoms, defined by $\mathcal{M}(\mathcal{B}) = \bigcup_i \mathcal{M}_i(\mathcal{B})$, where $\mathcal{M}_i(\mathcal{B})$ is defined inductively as follows:*

- *$a \in \mathcal{M}_0(\mathcal{B})$ iff: (1) $a \notin \mathcal{K}_p \cup \mathcal{T}_p$; (2) a occurs only within the set of terms $\{\mathsf{hash}(h, t) \mid t \in \mathcal{A}\}$ in $\mathcal{B}$ (recall that this could mean that a does not occur at all); and (3) a is not complex.*

- *$\mathcal{M}_{i+1}(\mathcal{B}) = \mathcal{M}_i(\mathcal{B}) \cup X_{i+1}(\mathcal{B})$, where $a \in X_{i+1}(\mathcal{B})$ iff: (1) $a \notin \mathcal{K}_p \cup \mathcal{T}_p$; (2) a occurs only within the set of terms $\{\{t\}_k \mid k^{-1} \in \mathcal{M}_i(\mathcal{B})\}$ in $\mathcal{B}$; and (3) if a is complex, at least one ingredient of a is in $\mathcal{M}_i(\mathcal{B})$.*

Define the set $\mathcal{S}(\mathcal{B})$ of safe keys by $\mathcal{S}(\mathcal{B}) = \mathcal{M}(\mathcal{B}) \cap \mathcal{K}$.

We say that atom *a occurs safely* if it is a member of $\mathcal{M}(\mathcal{B})$. To prove that a occurs safely, we show that it is not initially known by the penetrator, and that its occurrence in every new component is protected by a hash or an encryption with some key k such that $k^{-1} \in \mathcal{M}_i(\mathcal{B})$; i is typically 0 or 1 in most security protocols.

Definition 3.25 *A term t is said to occur safely in a bundle $\mathcal{B}$, iff for every positive regular node $n \in \mathcal{B}$ and a new component t_1 of n, t occurs only within the set of terms R in t_1 such that $\mathsf{R} \subseteq \{\mathsf{hash}(h), \{h\}_K : K^{-1} \in \mathcal{S}(\mathcal{B})\}$.*

Proposition 3.26 *Let $\mathcal{B}$ be a bundle and r an atom such that $r \notin \mathcal{K}_p \cup \mathcal{T}_p$, and r occurs safely in $\mathcal{B}$; then for every equivalent bundle $\mathcal{B}'$ there is no node $n \in \mathcal{B}'$ such that $msg(n) = r$.*

Proof. Let's assume for a contradiction that there is such a bundle $\mathcal{B}'$ with a node n such that $msg(n) = r$. By the definition of bundle equivalence, n is not a regular node since r occurs safely in $\mathcal{B}$. In adidtion, r cannot be originating in n since n is a penetrator node and $r \notin \mathcal{K}_p \cup \mathcal{T}_p$. Therefore, n must be traversed by a penetrator path p starting at a regular node n' such that $r \sqsubset msg(n')$. By Definition 3.25, there are two possibilities: (1) r occurs within a hashed term in $msg(n')$: In this case p cannot exist by the definition of hash functions. (2) r occurs within an encrypted term $\{h\}_k$ in $msg(n')$: p must traverse a decryption edge in a **D** strand. It follows from the structure of a **D** strand that there must exist a key edge with a node n'' such that $msg(n'') = k^{-1}$. However, such node cannot exist since k^{-1} is a safe key By Definition 3.25. Therefore, there is no node $n \in \mathcal{B}'$ such that $msg(n) = r$.

3.6.2 Authentication Tests

In [GT00a], Thayer and Guttman introduced the concept of *Authentication Tests* to prove the authentication goals of security protocols. In Section 4.3 we will apply these tests to the Handshake Protocol. The Authentication Tests are based on the fact that security protocols follow specific challenge-response patterns to achieve authentication. Informally, a principal creates and sends a message t_1 containing a uniquely originating value r and then receives r back in a transformed form t_2. If a safe key is necessary to transform

t_1 to t_2, then we can conclude that some regular participant possessing the relevant key has transformed t_1. Authentication tests can be used in two different ways:

- An outgoing test: r is sent in an encrypted form and the challenge is to decrypt it;
- An incoming test: The challenge is to create an encrypted value containing r using a safe key to prove that the encrypted term has not originated on a penetrator node.

The Authentication Tests theorems [GT00a, DGT07b, DGT07a] specify the conditions under which we can infer that certain regular nodes exist in the bundle. They proved to be one of the most powerful tools of the strand space model because of their simplicity. We extend these tests to include hashed values and prove their soundness in the extended model.

Authentication Test 1 *The Outgoing Authentication Test [DGT07a] Let $\mathcal{B}$ be a bundle. Suppose that $n_0, n_1 \in \mathcal{B}$, and*

$$R \subset \{\mathsf{hash}(h,t),\ \{t\}_K \mid t \in \mathcal{A},\ K^{-1} \in \mathcal{S}(\mathcal{B})\}.$$

Suppose that r originates uniquely in $\mathcal{B}$ on node n_0 and occurs only within R in $msg(n_0)$, but r occurs outside R in $msg(n_1)$. Then there is an integer i and a regular strand s such that $m_1 = \langle s, i\rangle \in \mathcal{B}$ is positive, and i is the least integer k such that r occurs outside R in $msg(\langle s, k\rangle)$. Moreover, there is a node m_0 such that $r \sqsubset msg(m_0)$, and $n_0 \preceq_{\mathcal{B}} m_0 \Rightarrow^+ m_1 \preceq_{\mathcal{B}} n_1$.

Proof. Consider the set $\Phi = \{m \in \mathcal{B} : r \textit{ occurs outside } (R) \textit{ in } msg(m)\}$. We know that Φ is non empty because $n_1 \in \Phi$ and . By bundle induction (Proposition 3.11), Φ has a $\preceq$-minimal positive member denoted as m_1. Therefore, $m_1 \preceq_{\mathcal{B}} n_1$.

Since r uniquely originates at n_0, there must exist an edge $n_0 \preceq_{\mathcal{B}} m_0 \Rightarrow^+ m_1$ in $\mathcal{B}$ such that $r \sqsubseteq msg(m_0)$.

Since, m_1 is defined as a minimal member of Φ and therefore $m_0 \notin \Phi$. Consequently, we can deduce that r occurs only within R in $msg(m_0)$ and *occurs outside* R in $msg(m_1)$. Now we want to prove that $m_0 \Rightarrow^+ m_1$ lies on a regular strand. Let us assume for a contradiction that $m_0 \Rightarrow^+ m_1$ lies on a penetrator strand. Such strand extracts r from R. However, this is not

possible since r *occurs safely* within R and therefore, cannot be extracted by the penetrator. Consequently, $m_0 \Rightarrow^+ m_1$ lies on a regular strand. $\square$

In outgoing authentication tests, the edge $m_0 \Rightarrow^+ m_1$ is called an *outgoing transforming edge* [DGT07b]. The nodes n_0 and n_1 are called an *outgoing authentication test*, and the edge $m_0 \Rightarrow^+ m_1$ is called an *outgoing transforming edge* [DGT07b]. In Lemma 4.11, we show that the TLS nodes $Client_5$ and $Client_8$ form an outgoing authentication test, and use this to establish authentication guarantees for the client.

Authentication Test 2 *The Incoming Authentication Test [DGT07a] Suppose that $n_1 \in \mathcal{B}$ is negative, $t = \{t_0\}_K \sqsubseteq msg(n_1)$, and $K \in \mathcal{S}(\mathcal{B})$. Then there exists a regular $m_1 \prec_{\mathcal{B}} n_1$ such that t originates on m_1.*

Proof. Consider the set $\Phi = \{m \in \mathcal{B} : t_1 \sqsubseteq msg(m)\}$. We know that Φ is non empty because $n_1 \in \Phi$. By bundle induction (Proposition 3.11), Φ has at least one $\preceq$-minimal member denoted by m_1. Let us assume for a contradiction that m_1 is a penetrator node. Since $t \sqsubseteq msg(m_1)$, we can deduce than m_1 can only be a positive node on an E-strand with a key edge K. But this contradicts the assumption that $K \in \mathcal{S}(\mathcal{B})$. Therefore, m_1 must be on a regular node. $\square$

In incoming authentication tests, the node m_1 is called an *incoming transforming node* and n_1 as an *incoming test node* [DGT07b].

The node m_1 is called an *incoming transforming node* and n_1 as an *incoming test node* [DGT07b]. In Lemma 4.12 we show that $Server_6$ is an incoming test node, and use this to establish authentication guarantees for the server.

3.7 Protocol Independence via Disjoint Encryption

Most formal verifications methods can only model and analyse cryptographic protocols the assumption that the participants in the protocol under analysis can only behave according to what is specified in the protocol. This assumption is clearly unrealistic since agents can usually engage in concurrent runs of several protocols. When multiple protocols are combined, the penetrator can use messages of a protocol to compose messages that (s)he could have not, otherwise, created. This problem is often referred to as the problem of *compositionality of security protocols*.

If parallel protocols are designed such that they share no common keys, the security of each protocol is independent of the others. However, in practice, it is not conceivable that, if an agent engaged in many protocols that use PKI, (s)he would have a different public key certificate for each protocol. Therefore, less restrictive conditions are required for concurrent protocols.

In [THG99, GT00b], Thayer *et al.* addressed the problem of protocol composition within the Strand Spaces framework by introducing the *Mixed Strand Spaces model* [THG99]. In this section, we outline some of the important concepts of the mixed strand spaces model. For a detailed discussion, the reader is referred to [GT00b].

To represent multiple protocols, some regular strands are selected as being runs of the protocol under analysis, which is referred to as the *primary* protocol. The regular strands of other protocols are referred to as *secondary*.

Definition 3.27 *A multiprotocol strand space is a strand space* (Σ, tr) *together with a distinguished subset of the regular strands* $\Sigma_1 \subset \Sigma \backslash \mathcal{P}_\Sigma$ *called the set of primary strands.*
(As Definition 3.1 in [GT00b].)

The concept of bundle equivalence is redefined within the mixed strand spaces as follows.

Definition 3.28 *Bundles* $\mathcal{B}$, $\mathcal{B}'$ *in the multiprotocol strand space* (Σ, tr, Σ_1) *are equivalent iff*

1. *they have the same primary nodes, meaning* $\mathcal{B} \cap \Sigma_1 = \mathcal{B}' \cap \Sigma_1$,
2. *for all* r, r *originates uniquely on a primary node in* $\mathcal{B}$ *iff* r *originates uniquely and on a primary node in* $\mathcal{B}'$,
3. *for all* r, r *is non-originating in* $\mathcal{B}$ *iff* r *is non-originating in* $\mathcal{B}'$.

(As Definition 16 in [Gut01].)

This is a wider notion of equivalence since it requires fewer nodes and unique origination facts to be unchanged. Therefore, any existence assertion about equivalent bundles remains true [Gut01].

As mentioned before, the simplest way to prevent parallel protocol harmful interactions would be to require that the two protocols not use the same ciphertext as a part of any message, i.e. if $n1 \in \Sigma_1$ and $n2 \in \Sigma_2$, and if

$\{h\}_K \sqsubseteq msg(n_1)$, then $\{h\}_K \not\sqsubseteq msg(n_2)$. However, such a condition would prevent using public key certificates between parallel protocols. Such shared encryptions are harmless because they contain public values. As long as the secondary protocol does not extract private values from within shared encryptions, or repackage their private contents, potentially insecurely, they should be allowed. The concept of a *shared encryption is defined as follows.*

Definition 3.29 *$\{h\}_K$ is a shared encryption if there exist $n_1 \in \Sigma_1$ and $n_2 \in \Sigma_2$ such that $\{h\}_K \sqsubseteq msg(n_1)$ and $\{h\}_K \sqsubseteq msg(n_2)$. It is an outbound shared encryption if this holds with n_1 positive and n_2 negative. It is an inbound shared encryption if this holds with n_1 negative and n_2 positive.*
(As Definition 22 in [Gut01].)

The mixed strand spaces model restricts but does not prevent shared encryptions. Outbound and inbound shared encryptions are treated differently.

Definition 3.30 (**Disjoint Outbound Encryption**) *Σ has disjoint outbound encryption iff for every outbound shared encryption $\{h\}_K$, for every atom $r \sqsubset \{h\}_K$, and for every $n_2 \Rightarrow^+ n_2' \in \Sigma_2$,*
if *n_2 is negative and $\{h\}_K \sqsubseteq msg(n_2)$,*
and *n_2' is positive and t_0 is a new component of n_2',*
then *$r \not\sqsubseteq t_0$.*
(As Definition 23 in [Gut01].)

The definition states that no secondary strand manipulates r into a new component. This definition has the important property that values originating uniquely on primary nodes cannot 'zigzag' to a secondary node, before being disclosed to the penetrator [Gut01].

The condition on inbound shared encryptions is that they should never occur in new components created on secondary nodes.

Definition 3.31 (**Disjoint Encryption**) *Σ has disjoint inbound encryption if for every inbound shared encryption $\{h\}_K$, and $n_2 \Rightarrow^+ n_2' \in \Sigma_2$ if $t_0 \sqsubseteq msg(n_2')$ is a new component, then $\{h\}_K \not\sqsubseteq t_0$).*

Σ has disjoint encryption if it has both disjoint inbound encryption and disjoint outbound encryption.
(As Definition 6.3 in [GT00b].)

Definition 3.32 *Σ_1 is independent of Σ_2 if for every bundle $\mathcal{B}$ in Σ, there is an equivalent bundle $\mathcal{B}'$ in Σ such that $\mathcal{B}'$ is disjoint from Σ_2.*
(As Definition 7.1 in [GT00b].)

The proposition of disjoint encryption is stated in terms of independent strand spaces (disjoint inbound and outbound encryption) as shown below. Proof of the proposition is provided in [GT00b].

Proposition 3.33 (**Protocol Independence**) *If Σ has disjoint encryption, then Σ_1 is independent of Σ_2. (As Proposition 27 in [Gut01].)*

□ In the next section, we will use some of the notions outlined above and use them to address the problem of multi-layer interaction in TLS.

4 Security Analysis of TLS

In this section we analyse TLS, formalise the security services it provides, and prove that the abstract model suggested by Broadfoot and Lowe [BL03] for TLS is correct. We use the simplified version of TLS provided in Section 2.3 and the extended strand spaces framework developed in Section 3 to analyse TLS.

In Section 4.1 we review Broadfoot and Lowe's abstraction of the security services provided by TLS. In Section 4.2 we state the assumptions we use in the analysis: these assumptions concern origination of terms, and the independence of the Handshake and Transaction Layer protocols. In Section 4.3 we analyse the Handshake protocol, obtaining guarantees for both the client and the server. Finally, in Section 4.4 we analyse the Record Layer, verifying authentication, secrecy and session independence properties.

Our proof is modular in the following sense:

- The analysis of the Handshake protocol is independent of the Record Layer protocol, other than as captured by the assumptions stated in Section 4.2; this analysis could, therefore, be re-used if the Handshake protocol were combined with an alternative Record Layer protocol.

- The analysis of the Record Layer protocol is independent of the Handshake protocol, other than the assumptions stated in Section 4.2, and that it assumes the secrecy and authentication properties we prove of

the Handshake protocol; this analysis could, therefore, be re-used if the Record Layer protocol were combined with an alternative Handshake protocol that achieves the same secrecy and authentication properties.

4.1 Broadfoot-Lowe Abstract Model for TLS

In [BL03], Broadfoot and Lowe describe the security services they believe TLS provides, as a trace-based specification, i.e. a specification in terms of a property that all traces of TLS should satisfy.

Let *Agent* be the set of all agents, partitioned into two sets: $Honest$ of honest agents, and $Dishonest$ of dishonest agents. The communications are assumed to be grouped in sessions, of type *Session*. Communications are described in terms of two channels: $send.A.B.s.m$ represents the Transaction Layer at A passing the message m to TLS to be sent to B as part of session s; and $receive.B.A.s.m$ represents TLS passing the message m, (apparently) received from A, to the Transaction Layer at B as part of session s.

Authentication and integrity requirements are expressed as follows:

$$\forall A, B : Honest; s : Session \bullet tr \downarrow receive.B.A.s \leq send.A.B.s.$$

Within each session between A and B, the messages accepted by the Record Layer of B as being from A are guaranteed to be sent by A, intended for B, and sent in that particular order. This property is called *stream* or *prefix* authentication.

The confidentiality requirement is expressed as follows:

$$\forall A : Honest; P : Dishonest; s : Session; m : Message; tr : Trace \bullet \\ tr^\frown\langle receive.A.P.s.m\rangle \leq tr \Rightarrow PIK \cup sentToPenetrator(tr') \vdash m.$$

$Ms \vdash M$ represents the ability of the penetrator to deduce message M from the set of messages Ms; PIK represents the set of messages initially known to the penetrator; and $sentToPenetrator(tr)$ is the set of messages deliberately sent to the penetrator (when he engages in TLS using one of his identities).

The predicate states that an agent A can receive message m from the penetrator only if that message can be produced from the penetrator's initial knowledge and those messages that have previously been deliberately sent to him. This predicate implies two properties:

- *Secrecy*: The intruder cannot deduce anything from the transaction layer messages observed in TLS sessions between honest agents.

- *Session independence* (*Non-hijackability*): If a TLS session is initiated with the penetrator using one of his identities, he cannot utilize (hijack) any Transaction Layer messages observed in another TLS session between honest agents in his own session; i.e., the two sessions are independent.

In the following sections, we use the strand spaces model to prove that TLS provides stream authentication, secrecy, and session independence for Transaction Layer messages.

4.2 Assumptions

In Section 1 we explained that one of the main problems that complicates the analysis of TLS is multi-layer interaction. We also stated that, since the syntactic structure of Transaction Layer is not specified by TLS, the Transaction Layer messages could, in principle, leak keys used by the Handshake and Record Layer Protocols. Furthermore, multi-layer attacks may happen, where a message from one layer is replayed and interpreted as being a message of the other layer, leading to an attack.

It is clear from the previous discussion that we should place sufficient conditions upon the syntactic structure of the Transaction Layer messages such that the correctness of TLS is independent of the Transaction Layer payload. Some of these conditions are merely origination assumptions which are necessary for proving authentication and secrecy, and others are mostly adapted from the concept of *disjoint encryption* [THG99, GT00b], which addresses the problem of protocol composition.

4.2.1 Origination Assumptions

Firstly, we lift the concepts of *nodes*, *terms*, *origination*, and *new component* to the Transaction Layer level.

Definition 4.1 *Let Σ be a TLS space and $\mathcal{N}$ be the regular nodes in Σ.*

- *a node n is a Record Layer node iff $n \in \mathcal{N}$ and there is an integer $i > 0$ and a term $t \in \mathcal{A}$ such that $msg(n) = \pm[t, i]_{mac,enc}$ where $mac \in \mathcal{K}_{MAC}$ and $enc \in \mathcal{K}_{Sym}$. The Transaction Layer term of n is then given by: $transmsg(n) = t$.*

- *An unsigned term t originates in the Transaction Layer iff there exists a positive Record Layer node n such that $t \sqsubseteq transmsg(n)$, and for every Record Layer node n' such that $n' \Rightarrow^+ n$, $t \not\sqsubseteq transmsg(n')$.*

- *A component t_1 of $transmsg(n_1)$ is new in the Transaction Layer at Record Layer node n_1 iff, for every Record Layer node n_0 such that $n_0 \Rightarrow^+ n_1$, t_1 is not a component of $transmsg(n_0)$.*

We now state our assumptions using the concepts defined above. First, secret keys and the premaster secret are not originated on regular Record Layer nodes:

Assumption 4.2 *None of the following originates in the Transaction Layer:*

- *secret keys from $\mathcal{K}_{Sec}$;*

- *terms of the form $G_0(*,*,*)$, $G_1(*,*,*)$, $G_2(*,*,*)$ or $G_3(*,*,*)$;*

- *the premaster secret pm.*

The fresh values used in TLS are uniquely originating.

Assumption 4.3 *For every regular client strand $st_c \in Client[c,*,r_c,*,*,*,*,pm,*]$ such that $SK(c) \notin \mathcal{K}_p$, r_c originates uniquely on $\langle st_c, 1\rangle$, and pm originates uniquely on $\langle st_c, 5\rangle$.*

For every server strand $st_s \in Server[s,*,*,r_s,*,*,*,*,*]$ such that $SK(s) \notin \mathcal{K}_p$, r_s originates uniquely on node $\langle st_s, 2\rangle$.

4.2.2 Disjoint Encryption Assumptions

We now adapt disjoint encryption assumptions from [GT00b, GT01], to prevent interactions between the two layers of TLS. To see why such assumptions are necessary, suppose the Transaction Layer protocol contains a simple nonce challenge of the following form:

$$\begin{array}{lll} \text{Message 1.} & a \longrightarrow b: & \{x\}_{PK(b)} \\ \text{Message 2.} & b \longrightarrow a: & x \end{array}$$

The penetrator can simply intercept message 5 of the Handshake protocol and send it to the server as message 1 of the Transaction protocol; the variable x gets bound to the value of the premaster secret, and so the premaster secret is revealed in message 2.

In what follows, we write Σ_H for the Handshake protocol nodes, and Σ_{RL} for the Record Layer nodes.

Definition 4.4 $\{t\}_K$ *is a* shared encryption *in a TLS space if there exist a Handshake node* $n_1 \in \Sigma_H$ *and a Record Layer node* $n_2 \in \Sigma_{RL}$ *such that* $\{t\}_K \sqsubseteq msg(n_1)$ *and* $\{t\}_K \sqsubseteq transmsg(n_2)$.

In [GT00b], Guttman and Thayer state that the simplest way to prevent multi-protocol harmful interactions would be to require that the parallel protocols do not use the same ciphertext as a part of any message. However, they argue that such a condition would prevent using public key certificates, for example, between different protocols. Such shared encryptions are harmless because they contain public values. This also applies to multi-layer interaction. On the other hand, layering protocols that extract private values from within shared encryptions, or repackage their private contents are potentially insecure. The following assumption captures the limitations we place upon shared encryptions.

Assumption 4.5 *Let* Σ *be a TLS space. We assume the following:*

1. *The Transaction Layer protocol does not remove pm from the protection of the encryption with the server's key: for every* $n \Rightarrow^+ n' \in \Sigma_{RL}$*, if* n *is negative,* n' *is positive,* $transmsg(n)$ *contains a subterm of the form* $\{pm\}_{k_s}$*, and* t_0 *is a new component of* $transmsg(n')$*, then* $pm \not\sqsubseteq t_0$.
2. *For every node* $n \in \Sigma_{RL}$*, no subterm of* $transmsg(n)$ *is of the form* $\{VH(prev_5)\}_{k_c}$.
3. *For every node* $n \in \Sigma_{RL}$*, no subterm of* $transmsg(n)$ *is of the form* $[M, n]_{mac,enc}$.

4.3 Security Analysis of the Handshake Protocol

We fix a TLS strand space Σ satisfying the above assumptions.

4.3.1 Public Key Infrastructure

We start by establishing the correctness of the public key infrastructure used to bootstrap the communication, i.e. each public key is reliably associated with its owner, and hence the corresponding secret key is not compromised.

In the following lemma, we prove that if a long term secret key is not initially known by the penetrator, then it is permanently safe.

Lemma 4.6 *Let $\mathcal{B}$ be a bundle in Σ. Then for all $n \in \mathcal{B}$, if $msg(n) \in \mathcal{K}_{sec}$ then $msg(n) \in \mathcal{K}_P$.*

Proof: Let k be a secret key. Examining the Handshake protocol, k does not originate on a regular Handshake node. Using Assumption 4.2, there is no regular node n_2 such that k originates in the Transaction Layer on n_2. Therefore, k originates on no regular node. It follows that, if $k \sqsubseteq msg(n)$, then k originates on a penetrator node, and consequently, $k \in \mathcal{K}_P$. □

We now present a couple of lemmas to prove that the certificates used by the participants in the protocol are valid and therefore each public key is correctly associated with its owner.

Lemma 4.7 *Let $\mathcal{B}$ be a bundle in Σ. For every public key certificate $\{pk\hat{}a\}_{sk} \sqsubseteq msg(n)$ where $n \in \mathcal{B}$, either $pk = PK(a)$ or $sk \in \mathcal{K}_P$.*

Proof: A certificate $\{pk\hat{}a\}_{sk}$ can originate in one of the following strands:

1. A regular CA strand $st \in CA[ca, a]$. In this case the certificate reliably associates each principal with its correct public key, i.e. $pk = PK(a)$.
2. A penetrator E strand. By Lemma 4.6, a penetrator can only use an initially known secret key to sign the certificate and therefore $sk \in \mathcal{K}_P$.

□

Lemma 4.8 *Let $\mathcal{B}$ be a bundle in Σ.*

1. *For every client strand $st_c \in Client[c, s, *, *, v_s, k_s, *, *, *]$ in $\mathcal{B}$, if $SK(c) \notin \mathcal{K}_P$, and $SK(v_s) \notin \mathcal{K}_P$, then $k_s = PK(s)$.*
2. *For every server strand $st_s \in Server[s, c, *, *, *, k_c, v_c, *]$ in $\mathcal{B}$, if $SK(s) \notin \mathcal{K}_P$, and $SK(v_c) \notin \mathcal{K}_P$, then $k_c = PK(c)$.*

Proof: Each regular strand specified above can only accept a public key certificate signed by a secret key $SK(v_s)$ or $SK(v_c)$ that is not in $\mathcal{K}_P$. By Lemma 4.7, if the certificate is signed by an uncompromised key then the public key included in the certificate is reliably associated with its owner. □

4.3.2 The Client's Guarantees

Firstly, we prove the secrecy of the premaster secret.

Lemma 4.9 *Let $\mathcal{B}$ be a bundle in Σ, and let $st_c \in Client[c, s, r_c, r_s, k_s, v_s, v_c, pm, Sms]$ be a regular client strand in $\mathcal{B}$ such that $SK(v_s), SK(s) \notin \mathcal{K}_P$. Then for all nodes $n \in \mathcal{B}$, $msg(n) \neq pm$.*

Proof: If we prove that premaster secret only *occurs safely* in Σ then we have proved its secrecy by Proposition 3.26.

Let R be the set of terms $\mathsf{R} = \{\mathsf{hash}(h), \{h\}_K \mid K^{-1} \textit{ is safe}\}$. Let S be the set of regular nodes such that $S = \{n \mid pm \textit{ occurs outside } \mathsf{R} \textit{ in } msg(n)\}$. Suppose, for a contradiction, that S is non-empty. Then by bundle induction (Proposition 3.11), it has a $\preceq$-minimal element m, which is positive. By Assumption 4.3, pm originates uniquely on $Client_5$. Therefore, $Client_5 \preceq m$. We perform a case analysis over m.

- Case $m = Client_5$. Then $msg(m) = \{pm\}_{k_s}$. By Lemma 4.8, $k_s = PK(s)$. Given that $SK(s) \notin \mathcal{K}_P$, $m \notin S$.
- Case m is some other regular Handshake node. By inspection of the Handshake protocol, no such node transforms a message to send pm outside of R, so $m \notin S$.
- Case m is a positive Record Layer node. pm does not originate in the Transaction Layer, by Assumption 4.2. Further, by clause 1 of Assumption 4.5, no Transaction Layer edge transforms a shared encryption so that pm occurs outside R. Consequently, $m \notin S$.

Hence S is empty and pm occurs safely in Σ. □

We now show that the session keys remain secret.

Lemma 4.10 *Let $\mathcal{B}$ be a bundle in Σ, and let $st \in Client[c, s, r_c, r_s, k_s, v_s, v_c, pm, Sms]$ be a regular client strand such that $SK(v_s), SK(s) \notin \mathcal{K}_p$. Then for all nodes $n \in \mathcal{B}$, $msg(n) \notin \{G_0(pm, r_c, r_s), G_1(pm, r_c, r_s), G_2(pm, r_c, r_s), G_3(pm, r_c, r_s)\}$.*

Proof: Examining the protocol, terms of the form $G_*(*, *, *)$ are only uttered as subterms of the hash function *HMAC* in the initial Handshake. Also, by Assumption 4.2 no such term originates in the Transaction Layer. It follows from the secrecy of the premaster secret (Lemma 4.9) and the definition of safe keys (Definition 3.24) that the session keys $G_0(pm, r_c, r_s)$, $G_1(pm, r_c, r_s)$, $G_2(pm, r_c, r_s)$, and $G_3(pm, r_c, r_s)$ are safe. □

We now prove that the server is authenticated to the client, and they agree on their identities, the nonces and the premaster secret.

Lemma 4.11 *Let $\mathcal{B}$ be a bundle in Σ, and $st_c \in Client[c, s, r_c, r_s, k_s, v_s, v_c, pm, *]$ be a client strand of $\mathcal{B}$-height at least 8, such that $SK(v_s), SK(s) \notin \mathcal{K}_p$. Then there exists a unique server strand $st_s \in Server[s, c, r_c, r_s, PK(c), v_s, v_c, pm, *]$ of $\mathcal{B}$-height at least 8.*

Proof: We show that the fifth and the eighth nodes on st_c form an outgoing authentication test for pm. Define the set of terms $\mathsf{R} = \{\{pm\}_{k_s}, \{VH(prev_5)\}_{SK(c)}, [PRF_{cf}(pm\hat{}prev_6), 0]_{cm,ce}\}$. By Assumption 4.3, pm uniquely originates on $Client_5$. By Lemma 4.8, $k_s = PK(s)$ and therefore pm occurs only within R in $msg(Client_5)$. In addition, pm occurs outside R at node $Client_8$. It follows that $Client_5 \Rightarrow^+ Client_8$ is an outgoing test for pm.

Using the Outgoing Authentication Test, there exists an outgoing transforming edge $m_0 \Rightarrow^+ m_1$ that lies on some regular strand. By clause 1 of Assumption 4.5, this edge does not involve Record Layer nodes. Hence, by inspection of the Handshake protocol, the transforming edge can only be $Server_5 \Rightarrow^+ Server_8$ in some server strand $st_s = Server[s', c', r'_c, r'_s, k'_c, v'_c, v'_s, pm', *]$. Consequently, the $\mathcal{B}$-height of st_s is at least 8. We can also deduce that $pm' = pm$. Since the server strand is using the corresponding secret key and PK is injective, it must be the case that $s' = s$.

Using the Incoming Authentication Test, we can prove that $msg(Client_8) = [PRF_{sf}(pm\hat{}prev_7), 0]_{sm,se}$ is an incoming authentication test. By Lemma 4.10, $sm, se \notin \mathcal{P}(\mathcal{B})$. Therefore, the term $[PRF_{sf}(pm\hat{}prev_7), 0]_{sm,se}$ must have originated in some regular node n. By clause 3 of Assumption 4.5, n is not a Record Layer node. Hence, by inspection of the Handshake protocol, n can only be $Server_8$ in some server strand st_s. We can deduce that: $r'_c = r_c$, since r_c is the first message of $prev_7$; $r'_s = r_s$, since r_s is the second message of $prev_7$; $v'_s = v_s$, since $\{s\hat{}PK(s)\}_{SK(v_s)}$ is the third message of $prev_7$; $c' = c$, $k'_c = PK(c)$, and $v'_c = v_c$ since $\{c\hat{}PK(c)\}_{SK(v_c)}$ is the fifth message of $prev_7$. Therefore, $st_s \in Server[s, c, r_c, r_s, PK(c), v_s, v_c, pm, *]$. Now we want to prove that such st_s is unique. By Assumption 4.3, r_s originates uniquely in Σ in a server strand. Hence, there can be at most one such st_s. $\square$

4.3.3 The Server's Guarantees

Having established the client's guarantees, we now prove the server's guarantees: the authentication of the client to the server, and the secrecy of the session keys used by the server.

Lemma 4.12 *Let $\mathcal{B}$ be a bundle in Σ, and $st_s \in Server[s, c, r_c, r_s, k_c, v_s, v_c, pm, *]$ be a server strand of $\mathcal{B}$-height at least 6, such that $SK(v_c), SK(c) \notin \mathcal{K}_p$. Then there exists a unique client strand $st_c \in Client[c, s, r_c, r_s, PK(s), v_s, v_c, pm, *]$ of $\mathcal{B}$-height at least 6.*

Proof: As in Lemma 4.11, we show that $Server_6$ forms an incoming test node. By Lemma 4.8, $k_c = PK(c)$, and so $\{VH(prev_5)\}_{k_c^{-1}}$ is a test component in $Server_6$. Using the Incoming Authentication Test, there exists a positive regular node $m_1 \in \mathcal{B}$ such that $\{VH(prev_5)\}_{k_c}$ originates on m_1.

By clause 2 of Assumption 4.5, m_1 cannot be a Record Layer node. Hence, by inspection of the Handshake protocol, $m_1 = Client_6$ for some client strand $st_c \in Client[c', s', r'_c, r'_s, k'_s, v'_s, v'_c, pm', *]$. As in Lemma 4.11 it is easy to prove that $k_c = PK(c)$, $c' = c$, $r'_c = r_c$, $r'_s = r_s$, $s' = s$, $k'_s = PK(s)$, $v'_s = v_s$, $v'_c = v_c$, and $pm' = pm$. Therefore, $st_c \in Client[c, s, r_c, r_s, PK(s), v_s, v_c, pm, *]$.

Now we want to prove that such st_c is unique. By Assumption 4.3, pm originates uniquely in Σ in st_c; hence, there can be at most one such st_c. (We could, alternatively, have used the unique origination of the client's nonce (Assumption 4.3) to establish the uniqueness of the client strand.)

□

Lemma 4.13 *Let $\mathcal{B}$ be a bundle in Σ, and $st \in Server[s, c, r_c, r_s, k_c, v_s, v_c, pm, Sms']$ a server strand such that $SK(v_s), SK(s), SK(v_c), SK(c) \notin \mathcal{K}_p$. Then for all nodes $n \in \mathcal{B}$, $msg(n) \notin \{G_0(pm, r_c, r_s), G_1(pm, r_c, r_s), G_2(pm, r_c, r_s), G_3(pm, r_c, r_s)\}$.*

Proof: The proof here is very similar to the proof of Lemma 4.10. The secrecy of the premaster secret pm, used to construct the keys, follows from the fact that pm is secret from the client's point of view by Lemma 4.9, and that the server and the client agree on the value of pm by Lemma 4.12. □

Note the difference between the conditions required by the client's guarantees (Lemmas 4.10 and 4.11) and the server's guarantees (Lemmas 4.12 and 4.13). The client only requires the server's secret key to be uncompromised, while the server requires the client's secret key and his own secret key to be uncompromised.

4.4 Security Analysis of the Record Layer

In the previous section we proved that the initial Handshake results in four secret authenticated session keys. In this section we formalise and prove the security services provided by the Record Layer.

4.4.1 Prefix Authentication

We prove that the Record Layer provides an authenticated stream for each participant, i.e. if the client receives a sequence of messages in the Transaction Layer, then the server must have sent these messages earlier in the same order, and vice versa.

We start by proving that any two sets of session keys used by two different strands for sending messages in the Record Layer are completely disjoint. Define $keys(st)$ to be the set of session keys for out-going messages for the regular strand st.

Definition 4.14 *Let $\mathcal{B}$ be a bundle in Σ, and st be a primary regular strand in $\mathcal{B}$:*

- *If $st \in Client[*, *, r_c, r_s, *, *, *, pm, *]$, then $keys(st) = \{G_0(pm, r_c, r_s), G_2(pm, r_c, r_s)\}$,*
- *If $st \in Server[*, *, r_c, r_s, *, *, *, pm, *]$, then $keys(st) = \{G_1(pm, r_c, r_s), G_3(pm, r_c, r_s)\}$.*

Lemma 4.15 *Let $\mathcal{B}$ be a bundle in Σ, and st_1 and st_2 be primary regular strands of $\mathcal{B}$-height at least 8. Then $st_1 \neq st_2 \Rightarrow keys(st_1) \cap keys(st_2) = \{\}$.*

Proof: Let $st_1 \in Client[c_1, s_1, r_{c1}, r_{s1}, *, *, *, pm_1, *]$ and $st_2 \in Client[c_2, s_2, r_{c2}, r_{s2}, *, *, *, pm_2, *]$ be distinct regular client strands. Then

$$\begin{aligned} keys(st_1) &= \{G_0(pm_1, r_{c1}, r_{s1}), G_2(pm_1, r_{c1}, r_{s1})\}, \\ keys(st_2) &= \{G_0(pm_2, r_{c2}, r_{s2}), G_2(pm_2, r_{c2}, r_{s2})\}. \end{aligned}$$

By Definition 3.1, the ranges of G_0 and G_2 are disjoint. Therefore, keys constructed using G_0 are distinct from keys constructed using G_2. By Assumptions 4.3, $pm_1 \neq pm_2$ since $st_1 \neq st_2$. By Definition 3.1, key generator functions are collision free, and hence $G_0(pm_1, r_{c1}, r_{s1}) \neq G_0(pm_2, r_{c2}, r_{s2})$ and $G_2(pm_1, r_{c1}, r_{s1}) \neq G_2(pm_2, r_{c2}, r_{s2})$. Hence $keys(st_1) \cap keys(st_2) = \{\}$.

The result can be proved in a similar way for other combinations of TLS primary strands, i.e. a server strand and a client strand, and two server strands. □

We now show that each message received by a principal in the Transaction Layer is authenticated, i.e. has been sent earlier by the expected sender and was intended for that principal.

Lemma 4.16 *Let $\mathcal{B}$ be a bundle in Σ, and $st_c \in Client[c, s, r_c, r_s, k_s, v_s, v_c, pm, Sms]$ a client strand such that $SK(v_s), SK(s) \notin \mathcal{K}_P$. Let n be a node in st_c such that $msg(n) = -[t, i]_{ce,cm}$ for $i > 0$. Then there exists a node n' in the unique corresponding server strand $st_s \in Server[s, c, r_c, r_s, k_s, v_s, v_c, pm, Sms']$ such that $msg(n') = +[t, i]_{se,sm}$.*

Proof: By Lemma 4.11, the server strand $st_s \in Server[s, c, r_c, r_s, k_s, v_s, v_c, pm, *]$ exists and is unique. Recall that a message received by a client in the Record Layer is of the form:

$$-[t, i]_{se,sm} = -\{t, Hmac(sm, \{i, t\})\}_{se}.$$

By Lemma 4.10 the server session keys se and sm are safe, i.e. only known to the client and the server. It follows from the Incoming Authentication Test that a term in the form $\{t\}_{se}$ can only originate in a regular strand. In particular, it can originate only in the strand st_s by Lemma 4.15. □

Lemma 4.17 *Let $\mathcal{B}$ be a bundle in Σ, and $st_s \in Server[s, c, r_c, r_s, k_s, v_s, v_c, pm, Sms]$ a server strand such that $SK(v_c), SK(c), SK(v_s), SK(s) \notin \mathcal{K}_P$. Let n be a node in st_s such that $msg(n) = -[t, i]_{se,sm}$ for $i > 0$. Then there exists a node n' in the unique corresponding client strand $st_c \in Client[c, s, r_c, r_s, k_s, v_s, v_c, pm, Sms']$ such that $msg(n') = +[t, i]_{ce,cm}$.*

Proof: The proof is similar to the previous lemma, and uses Lemmas 4.12, 4.13, and 4.15. □

We now prove that if a principal receives a stream of messages in the Transaction Layer, then the corresponding principal must have sent the same messages in the same order earlier. We write $sent(Sms)$ and $received(Sms)$ for the sent and received messages of Sms:

$$\begin{aligned} sent(Sms) &:= \langle m \mid +m \longleftarrow Sms \rangle, \\ received(Sms) &:= \langle m \mid -m \longleftarrow Sms \rangle. \end{aligned}$$

We write $\#Sms$ for the length of Sms.

Theorem 1 *Let $\mathcal{B}$ be a bundle in Σ.*

1. *For each client strand* $st_c \in Client[c, s, r_c, r_s, k_s, v_s, v_c, pm, Sms]$ *of* $\mathcal{B}$*-height* $8 + \#Sms$*, and such that* $SK(v_s), SK(s) \notin \mathcal{K}_P$*, there is a unique server strand* $st_s \in Server[s, c, r_c, r_s, k_c, v_s, v_c, pm, Sms']$ *such that* $received(Sms) \leq sent(Sms')$ *and* $received(Sms') \leq sent(Sms)$*.*

2. *For each server strand* $st_s \in Server[s, c, r_c, r_s, k_c, v_s, v_c, pm, Sms]$ *of* $\mathcal{B}$*-height* $8 + \#Sms$*, and such that* $SK(v_c), SK(c), SK(v_s), SK(s) \notin \mathcal{K}_P$*, there is a unique client strand* $st_c \in Client[c, s, r_c, r_s, k_s, v_s, v_c, pm, Sms']$ *such that* $received(Sms) \leq sent(Sms')$ *and* $received(Sms') \leq sent(Sms)$*.*

Proof: We prove the first part of the theorem; the second part can be proved in a similar way.

We have chosen the $\mathcal{B}$*-height* of the client strand such that the whole strand is included in $\mathcal{B}$. By Lemma 4.16, there is a unique server strand $st_s \in Server[s, c, r_c, r_s, k_s, v_s, v_c, pm, Sms']$ such that for each message $-[t, i]_{ce,cm}$ received by the client strand, the server sent $+[t, i]_{se,sm}$. Further, the i determine the order in which the messages are sent and received. It follows that the messages in $received(Sms)$ must be in $sent(Sms')$, and t must have the same index i in both sequences. It follows that $received(Sms) \leq sent(Sms')$. □

4.4.2 Secrecy

We now prove that the Record Layer provides secrecy for the Transaction Layer. Since Transaction Layer messages may contain terms that are known to the penetrator before starting the Transaction Layer exchange, such as identities, certificates, etc., the Record Layer cannot guarantee that the penetrator does not know any of the contents of the Transaction Layer messages. The secrecy provided by the Record Layer guarantees that the penetrator cannot learn anything "new" from messages exchanged in the Transaction Layer between two regular strands.

Theorem 2 *Let* $\mathcal{B}$ *be a normal bundle in* Σ*.*

1. *For each client strand* $st_c \in Client[c, s, r_c, r_s, k_s, v_s, v_c, pm, Sms]$ *such that* $SK(v_s), SK(s) \notin \mathcal{K}_P$*, there is no penetrator path that starts at a Record Layer node* $n' \in st_c$ *and includes a penetrator node* n *such that* $msg(n) \sqsubseteq transmsg(n')$*.*

2. *For each server strand* $st_s \in Server[s, c, r_c, r_s, k_c, v_s, v_c, pm, Sms]$ *such that* $SK(v_c), SK(c), SK(v_s), SK(s) \notin \mathcal{K}_P$, *there is no penetrator path that starts at a Record Layer node* $n' \in st_s$ *and includes a penetrator node* n *such that* $msg(n) \sqsubseteq transmsg(n')$.

Proof: We prove the first part; the second part is very similar.

Let us assume, for a contradiction, that there is such a penetrator path from n' to n. Let $transmsg(n') = t$. From Definition 3.5,

$$msg(n') = +\{t, Hmac(cm, \{i, t\})\}_{ce}.$$

Since $\mathcal{B}$ is normal and $msg(n) \sqsubseteq t$, the first $\Rightarrow$ edge in the path from n' to n must be a decryption edge in a **D** strand. It follows that the key node of this **D** strand has message ce. But this contradicts Lemma 4.10. □

4.4.3 Session Independence

In Theorem 1 we proved that the penetrator cannot replay messages from one session between regular strands into another session between regular strands. But can he replay messages from a session between regular strands into a session where he is taking part using his own identities? In this section we show that he cannot. More precisely, we show that a penetrator path that starts at a Record Layer node in a session between regular strands can lead to a regular node n' only if n' lies on the other strand in the same session. Hence different sessions are independent.

Theorem 3 *Let* $\mathcal{B}$ *be a normal bundle in* Σ.

1. *Suppose* $st_c \in Client[c, s, r_c, r_s, k_s, v_s, v_c, pm, Sms]$ *is a regular client strand such that* $SK(v_s), SK(s) \notin \mathcal{K}_P$. *Suppose there is a penetrator path* p *that starts at a Record Layer node* $n \in st_c$ *and ends at another regular node* n'. *Then* n' *is on the corresponding server strand* $st_s \in Server[s, c, r_c, r_s, PK(c), v_s, v_c, pm, *]$, *as in Theorem 1.*

2. *Suppose* $st_s \in Server[s, c, r_c, r_s, k_c, v_s, v_c, pm, Sms]$ *is a regular server strand such that* $SK(v_c), SK(c), SK(v_s), SK(s) \notin \mathcal{K}_P$. *Suppose there is a penetrator path* p *that starts at a Record Layer node* $n \in st_s$ *and ends at another regular node* n'. *Then* n' *is on the corresponding client strand* $st_c \in Client[c, s, r_c, r_s, PK(s), v_s, v_c, pm, *]$, *as in Theorem 1.*

Proof: We prove the first part of the theorem; the second part is similar.

Consider the form of the penetrator path starting at n and ending at n' (recalling that $\mathcal{B}$ is normal). The path cannot contains a node n'' such that $msg(n'')$ is a proper subterm of $msg(n)$, since such paths cannot exist by Theorem 2. Further, $msg(n)$ cannot be a proper subterm of $msg(n')$: examining the protocol, no Handshake node contains a Record Layer message as a proper subterm; and by Assumption 3, no Record Layer node contains a Record Layer message as a proper subterm. Hence $msg(n) = msg(n')$, and so n' is a Record Layer node.

Let $st_s \in Server[s, c, r_c, r_s, k_s, v_s, v_c, pm, *]$ be the corresponding server strand, as given in Theorem 1. Clearly n' must be on a strand that uses the same session keys as st_c; and hence this strand must be st_s, by the unique origination of the premaster secret and the server's nonce. □

5 Conclusions and Related Work

In this paper we employed the strand spaces model to analyse and verify the TLS protocol. To enable this analysis, we simplified the TLS protocol using safe simplifying transformations [HL99]. In addition, we extended the term algebra and the penetrator's model in the strand spaces framework to include the operation of generating complex keys using hash functions. Finally, we analysed the TLS protocol using the adapted strand spaces model. We started the analysis by placing some syntactic assumptions on the application protocols. We then adopted a modular verification approach starting with the initial Handshake protocol and then proceeding to the Record Layer protocol. We concluded our verification by formalising the security services provided by TLS: mutual authentication, stream authentication, confidentiality, and session independence. Consequently, we verified that the abstract model suggested by Broadfoot and Lowe in [BL03] was correct under the stated assumptions of the analysis. Adopting a modular analysis approach has reduced the complexity of the analysis, provided a clearer and better understanding of the TLS protocol, and potentially allows for proof re-use.

Our analysis of TLS has improved on previous formal verifications of the protocol: we know of no previous security proof of TLS that examines the Record Layer protocol. Consequently, the abstract security services provided by TLS for the Transaction Layer have not been verified previously.

TLS has been analysed before using model checking techniques, for exam-

ple in [DCVP04]. However, these techniques are constrained by the intrinsic limitations of model checking such as state explosion and incompleteness of results.

Many direct proof techniques have been used to verify TLS. Examples include Protocol Composition Logic (PCL) [HSD+05] and equational reasoning [OF05]. Perhaps the most known attempt is the verification carried out by Paulson [Pau99] using the inductive approach [Bel00] to analyse a simplified version of TLS. The analysis took a moderate six man-weeks effort, to model the protocol in HOL as inductive definitions, and just under three minutes to generate the proofs in Isabelle. The abstract message exchange was obtained by *reverse engineering* the TLS specification; unlike our use of fault-preserving transformations, this does not guarantee that no attacks are lost. Further, although the inductive proofs assume that "application data does not contain secrets associated with TLS sessions, such as keys and master-secrets", they do not impose clear restrictions on the syntactic structure of the Transaction protocol to ensure this.

Although the main focus of our analysis is the TLS protocol version 1.0, the techniques used in this paper can be applied with minor modifications to many variants of TLS that achieve different sets of security goals, for example, unilateral TLS.

Our current research focuses on utilizing the rich framework of strand spaces to model a wide variety of secure channels. The model abstracts away from the implementation details of the secure channels, and just models the security services they provide. The aim is to facilitate the layered analysis approach described in the introduction, and to enable its application to a wide variety of layered security architectures.

Acknowledgements

We would like to thank Chris Dilloway for his helpful comments on this work. We would also like to thank Joshua Guttman for many useful discussions on the strand space model, over several years. This work is partially funded by a research studentship from the UK Engineering and Physical Sciences Research Council (EPSRC).

References

[Aut04] Mike Auty. Simplifying TLS, March 2004.

[BAN89] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. In *Proceedings of the Royal Society of London*, volume 426, pages 233–271, 1989.

[Bel00] G. Bella. *Inductive Verification of Cryptographic Protocols.* PhD thesis, University of Cambridge, March 2000.

[BL03] Philippa Broadfoot and Gavin Lowe. On distributed security transactions that use secure transport protocols. *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW)*, 00(2):141, 2003.

[BLS+95] J. Benaloh, B. Lampson, D. Simon, T. Spies, and B. Yee. The private communication technology protocol, 1995.

[CJ97] John A. Clark and Jeremy L. Jacob. A survey of authentication protocol literature. Technical Report 1.0, 1997.

[Cre04] C.J.F. Cremers. Compositionality of security protocols: A research agenda. In F. Gadducci and M. ter Beek, editors, *Proceedings of the 1st VODCA Workshop*, volume 142 of *Electronic Notes in Theoretical Computer Science*, pages 99–110. Elsevier ScienceDirect, 2004.

[DA99] T. Dierks and C. Allen. The TLS protocol: Version 1.0. request for comments: 2246, available at http://www.ietf.org/rfc/rfc2246.txt, 1999.

[DCVP04] Gregorio Diáz, Fernando Cuartero, Valentiń Valero, and Fernando Pelayo. Automatic verification of the TLS handshake protocol. In *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC'04)*, pages 789–794, New York, NY, USA, 2004. ACM Press.

[DGT07a] Shaddin F. Doghmi, Joshua D. Guttman, and F. Javier Thayer. Completeness of the authentication tests. In *Proceedings of the*

12th European Symposium On Research In Computer Security (ESORICS), pages 106–121, 2007.

[DGT07b] Shaddin F. Doghmi, Joshua D. Guttman, and F. Javier Thayer. Searching for shapes in cryptographic protocols. In *Proceedings of the 13th International Conference, Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 523–537, 2007.

[DY83] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(2):198–208, March 1983.

[GT00a] Joshua D. Guttman and F. Javier Thayer. Authentication tests. In *IEEE Symposium on Security and Privacy*, pages 96–109, 2000.

[GT00b] Joshua D. Guttman and F. Javier Thayer. Protocol independence through disjoint encryption. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW)*, Washington, DC, USA, 2000. IEEE Computer Society.

[GT01] Joshua D. Guttman and F. Javier Thayer. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 2001.

[Gut01] Joshua D. Guttman. Security goals: Packet trajectories and strand spaces. *Lecture Notes in Computer Science*, 2171:197–263, 2001.

[GW96] I. Goldberg and D. Wagner. Randomness and the netscape browser. *Dr. Dobbs Journal*, January 1996.

[HL99] Mei Lin Hui and Gavin Lowe. Safe simplifying transformations for security protocols. In *Proceedings of The 12th Computer Security Foundations Workshop (CSFW)*. IEEE Computer Society Press, 1999.

[HSD+05] Changhua He, Mukund Sundararajan, Anupam Datta, Ante Derek, and John C. Mitchell. A modular correctness proof of IEEE 802.11i and TLS. In *Proceedings of the 12th ACM conference on Computer and Communications Security (CCS)*, pages 2–15, New York, NY, USA, 2005. ACM Press.

[Kar01] Kanita Karaduzovic. Analysis of the transport layer security protocol. Master's thesis, University of Oxford, September 2001.

[Low01] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. *Lecture Notes in Computer Science*, 1055:147–166, 2001.

[OF05] Kazuhiro Ogata and Kokichi Futatsugi. Equational approach to formal analysis of TLS. In *ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 795–804, Washington, DC, USA, 2005. IEEE Computer Society.

[Pau99] Lawrence C. Paulson. Inductive analysis of the internet protocol TLS. *ACM Transactions on Information and System Security*, 2(3):332–351, 1999.

[Str96] Strawman. Secure transport layer protocol (stlp), discussion draft, 1996.

[THG98] F. Javier Thayer, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Why is a security protocol correct?. In *IEEE Symposium on Research in Security and Privacy*, pages 160–171. IEEE Computer Society Press, 1998.

[THG99] F. Javier Thayer, Jonathan C. Herzog, and Joshua D. Guttman. Mixed strand spaces. In *Proceedings of the 1999 IEEE Computer Security Foundations Workshop (CSFW)*, Washington, DC, USA, 1999. IEEE Computer Society.

[Tho00] Stephen Thomas. *SSL and TLS: Securing the Web.* Wiley, 2000.

[YC01] A. Yasinsac and J. Childs. Analyzing internet security protocols. In *The 6th IEEE International Symposium on High-Assurance Systems Engineering (HASE)*, page 0149, Washington, DC, USA, 2001. IEEE Computer Society.

A The public key infrastructure (PKI)

Public key certificates are used to bind an identity with a public key usually under the signature of a trusted certificate authorities (CA)s to prevent the fake use of public keys. X.509 is the international standard certificate that is widely accepted as the appropriate format for public key certificates. The format of a X.509 certificate can be represented as follows:

$$certificate = (Version, ID, \langle ID\ Related\ Info\rangle, Validity, Algorithms, PK_{ID}, PK_{issuer})_{SK_{issuer}}$$

where:

- $Version$ denotes the version of the certificate,
- ID denotes the identity of the holder of the certificate,
- $\langle ID\ Related\ Info\rangle$ holds information related to the identity holder: e.g. a friendly name, a serial number, etc,
- $Validity$ is a pair of from-to dates expressed as UDT,
- $Algorithms$ refers to the algorithms used to generate the key pair,
- PK_{ID} denotes the public key associated with ID,
- PK_{issuer} denotes the public key of the issuer of the certificate, and
- SK_{issuer} refers to the private key of the issuer of the certificate.

X.509 certificates can be chained. A certificate chain is a sequence of certificates issued by the successive issuers. Each certificate is followed by the certificate of its issuer. The signature can be verified with the pubic key in the issuer's certificate, which is the next certificate in the chain. The certificate chain should lead to a certificate that is signed by a trusted party.

B TLS Message Flow

This section includes the complete description of the TLS protocol.

B.1 Hanshake Protocol Messages

The message flow of the Handshake protocol iis given below. Please note that messages 10 and 12 are CipherSpecChange messages and are not part of the Hanshake protocol.

1. ClientHello	$C \longrightarrow S:$	$message\_type, message\_length, client\_version,$ $client\_nonce, sessionID\_length, sessionID,$ $cipher\_suite\_length, Cipher\_Suites,$ $compression\_length, Compression\_Methods$
2. ServerHello	$S \longrightarrow C:$	$message\_type, message\_length, server\_version,$ $server\_nonce, sessionID\_length, sessionID,$ $cipher\_suites, compression\_methods$
3. ServerCertificate	$S \longrightarrow C:$	$message\_type, message\_length, certificate\_chain\_length,$ $Certificate\_List$
4. ServerKeyExchange	$S \longrightarrow C:$	$message\_type, message\_length, Parameters,$ $Signed_Parameters$
5. CertificateRequest	$S \longrightarrow C:$	$message\_type, message\_length, certificate\_type\_length,$ $Certificate\_Types, certificate\_authorities\_length,$ $Certificate\_Authorities$
6. ServerHelloDone	$S \longrightarrow C:$	$message\_type, message\_length$
7. ClientCertificate	$C \longrightarrow S:$	$message\_type, message\_length, certificate\_chain\_length,$ $Certificate\_List$
8. ClientKeyExchange	$C \longrightarrow S:$	$message\_type, message\_length, Premaster\_secret$
9. CertificateVerify	$C \longrightarrow S:$	$message\_type, message\_length,$ $(Hash(handshake\_messages))_{SK(C)}$
10. ChangeCipherSpec	$C \longrightarrow S:$	1

11. ClientFinished	$C \longrightarrow S:$	$message\_type, message\_length, PRF(master\_secret,$ "$clientfinished$", $MD5(handshake\_messages,$ $SHA(handshake\_messages)))$
12. ChangeCipherSpec	$S \longrightarrow C:$	1
13. ServerFinished	$S \longrightarrow C:$	$message\_type, message\_length, PRF(master\_secret,$ "$serverfinished$", $MD5(handshake\_messages,$ $SHA(handshake\_messages)))$

where

$RANDOM$	$=$	$4\_bye\_gmt\_unix\_time + 28\_byte\_random\_bytes$
$Cipher\_Suites$	$=$	$cipher\_suite_1, cipher\_suit_2, \ldots, cipher\_suite_n$
where		
$cipher\_suite_i$	$=$	$key\_exchange\_algorithm, cipher\_algorithm,$ $MAC\_algorithm, cipher\_type, is\_exportable, hash\_size,$ $key\_material\_info, IV\_size$
$Compression\_Methods$	$=$	$compression_1, compression_2, \ldots, compression_n$
$Certificate\_List$	$=$	$certificate_1\_length, certificate_1,$ $certificate_2\_length, certificate_2, \ldots,$ $certificate_n\_length, certificate_n$
$Certificate\_Types$	$=$	$certificate_1, certificate_2, \ldots, certificate_n$
$Certificate\_Authorities$	$=$	$certificate\_authority_1, certificate\_authority_2,$ $\ldots, certificate\_authority_n$

where $certificate_i$ is defined in A

The fields $Parameters$, $Signed\_Parameters$, and $premaster\_secret$ are defined as follows:

if $(key\_exchange\_algorithm = DH)$

$Parameters = DH\_length, DH\_value$

else

$Parameters = RSA\_length, RSA\_value$

$$Signed\_Parameters = \{Hash(Client\_random + Server\_random + Parameters\}_{SK(B)}$$

if	$(key\_exchange\_algorithm = RSA)$		
	$premaster\_secret$	$=$	$(client\_version, premaster\_secret)_{RSA/DH}$
else			
if	$(key\_exchange\_algorithm = FixedDH)$		
	$premaster\_secret$	$=$	$null$
else			
	$premaster\_secret$	$=$	DH_length, DH_value
where			
	$premaster\_secret$	$=$	$client\_version + random$

B.2 Record Layer Protocol Messages

The message flow of the Record Layer protocol is given below.

$$\begin{array}{lll} \text{ServerRecord} & S \longrightarrow C: & \{content\_type, version, message\_length, \\ & & Stream\_Cipher/Block\_Cipher\}_{Server\_Key} \\ \text{ClientRecord} & C \longrightarrow S: & \{content\_type, version, message\_length, \\ & & Stream\_Cipher/Block\_Cipher\}_{Client\_Key} \end{array}$$

where

$$\begin{array}{lcl} Stream\_Cipher & = & Application\_Data + \\ & & HMAC(Server\_MAC\_Secret/Client\_MAC\_Secret, HMAC\_Data) \\ Block\_Cipher & = & Application\_Data + padding\_length + \\ & & HMAC(Server\_MAC\_Secret/Client\_MAC\_Secret, HMAC\_Data) \\ HMAC\_Data & = & sequence\_number, TLS\_protocol\_message, \\ & & TLS\_version, message\_length, message\_content \end{array}$$