Analysing TLS in the Strand Spaces Model

Allaa Kamil Gavin Lowe*

April 18, 2011

Abstract

In this paper, we analyse the Transport Layer Security (TLS) protocol (in particular, bilateral TLS in public-key mode) 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 fault-preserving 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 the public-key mode of bilateral TLS with its two main components: the Handshake and Record Layer protocols.

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, Low96, THG98, Bla09]) and used successfully to analyse a number of standard security protocols. Despite these advances, many practical security systems 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.

*Authors' address: Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford, OX1 3QD, United Kingdom, e-mail allaa.kamil, gavin.lowe@cs.ox.ac.uk.

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 a *penetrator* whose capabilities depend on the surrounding environment in which the security protocol is executed.

There are two different threat models for analysing security protocols. In both models, the penetrator is assumed to be in complete control of the network and may overhear, inject, and intercept any message sent on the network, as well as participating in protocol runs using his own identity. The difference between the models lies in how they treat cryptography.

In the Dolev-Yao model [DY83] —the model we use in this paper— the penetrator 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.

This contrasts with computational models of cryptography, which model cryptography more faithfully, e.g. [LMMS98, Can01]: messages are modelled as probability distributions over bit-strings, and the penetrator is modelled as a probabilistic polynomial-time Turing machine. This approach gives stronger results than in the Dolev-Yao model — but at the cost of extra complexity within proofs. In recent years there has been much interesting work trying to link the two approaches, e.g. [AR00, BP05, CW05].

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 transport protocols. The secure transport protocols provide a secure channel to the higher layer. In principle, such a system can be analysed by explicitly 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 to simplify the analysis [BL03, Cre04]. This approach comprises the following steps:

- Analyse the low-level secure transport protocol 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 transport protocol and just model the services it provides — that is, model it as an abstract secure channel;

- Analyse the high-level layers using the new layer of abstraction developed, and hence verify the whole security architecture.

The current paper addresses the first of these points. We analyse the Transport Layer Security (TLS) Protocol [DA99], and formalise the security services it provides. The TLS protocol has been our choice for analysis since it is an industry standard used in almost every online financial transaction. We restrict ourselves to the public-key mode (as opposed to Diffie-Hellman or Fortezza modes) of TLS in this paper. The verification method used is the strand spaces model [THG98].

In [BL03], Broadfoot and Lowe adopted the above approach to analyse layered security architectures and suggested an abstraction of TLS as a low-level secure channel. The abstraction models the security services that appear to be provided by TLS for the high-level security layer, sometimes referred to as the *Application Layer*. In the current paper, we take this abstraction as our starting point.

In more recent work, we have developed this approach further, corresponding to the second and third points above. In [KL09] we develop an abstract strand spaces model for reasoning about layered security protocols. The model abstracts away from the implementation of the secure channel and models the service it provides. It models high-level actions that a penetrator might be able to perform, such as faking messages or learning the content of messages. Properties of particular secure transport protocols are captured by restricting the penetrator actions available; for example, when modelling TLS, all faking and learning would be prevented. This abstraction is proved sound in [KL10]. Examples using this approach are described in [Kam09].

In addition to its benefits in terms of modelling simplicity and automated analysis efficiency [BL03], this layered approach has huge advantages in terms of re-usability. 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.

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 Application 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. Yasinsac and Childs [YC05] estimate that 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 Application Layer. The syntactic structure of the application protocol is not specified by the protocol. Indeed, the Application 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 fault-preserving simplifying transformations [HL01] 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 Application Layer messages; in particular, these assumptions forbid running TLS itself as the Application Layer protocol, i.e. layering TLS on top of itself.

We prove the following properties of TLS, each subject to certain assumptions:

Prefix authentication If a client c receives a sequence of messages in the Application Layer, apparently from server s, then s must have sent those messages earlier in the same order, intended for c, and vice versa.

Secrecy The penetrator does not learn the data sent in the Application Layer.

Session independence The penetrator cannot replay messages containing Application Layer data from a session between two honest agents into a sesion where he is taking part using his own identity.

Interference freedom TLS does not interfere with the Application Layer protocol; this means that when we analyse the Application Layer protocol we can abstract away the details of TLS and just consider the services it provides.

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 believe it would be reasonably straightforward to adapt our analysis to the unilateral mode. As noted above, we also do not consider the case where the shared secret is negotiated using Diffie-Hellman or Fortezza exchange [DA99].

This paper is organised in five sections. In Section 2 we give an overview of the TLS protocol and its security goals. 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 TLS: Protocol Overview

According to the TLS specification document [DA99], the primary goal of TLS is to provide confidentiality and data integrity between two communicating applications. The protocol defines two primary 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.

TLS operates on top of some reliable transport protocol (e.g. Transmission Control Protocol (TCP)) and below some higher-level application protocol which can be a general-purpose protocol, e.g. Hyper Text Transport Protocol (HTTP), or a special-purpose application security protocol such as an e-commerce protocol. In other words, TLS uses transport protocols to establish a secure channel on behalf of higher-level application protocols. TLS itself consists of four sub-protocols.

- The *Handshake Protocol* is used by the communicating parties to agree on the cryptographic keys and other cryptographic parameters that will be used in the initiated session, and to authenticate both parties.
- The *Change Cipher Spec Protocol* is used by the sending party to notify the receiving party that the subsequent messages should be protected by the negotiated cryptographic parameters. It uses a single byte with value 1 to indicate the transition.
- The *Alert Protocol* provides exception handling for TLS-secured connections.
- The *Record Layer Protocol* encapsulates messages from higher layer protocols including messages from the Handshake, Change Cipher Spec, Alert, and application protocols. The data stream is fragmented into a series of records, which are protected by the cryptographic parameters agreed upon in the Handshake protocol, and passed to a transport layer protocol for transmission.

From a security point of view, the TLS Protocol is composed of two main components, the Record Layer Protocol and the Handshake Protocol.

We do not consider session re-negotiation in this paper. Recently, an attack was found on re-negotiation [Ext09], which exploits that there is no cryptographic binding between the two sessions. A recent Internet Engineering Task Force RFC [RRDO10] seeks to fix this.

In the following subsections we describe the Record Layer and Handshake Protocols. We then perform various simplifications on the protocols, which will simplify our subsequent analysis. We will formalise the relationship between the full and simplified protocols in Section 3.3, once we have presented the relevant background material on strand spaces.

2.1 The Record Layer

As mentioned before, the Record Layer Protocol encapsulates all messages of the higher-layers. These messages are fragmented into records and compressed. If the records are exchanged after a run of the Change Cipher Spec Protocol, then they go through the following cryptographic operations before being transmitted via a transport layer protocol:

- Message authentication codes MACs are computed over the messages. TLS uses the standard message authentication code $HMAC$, which requires two parameters: a secret parameter and a data parameter. During the Handshake Protocol, two MAC keys, denoted cm and sm, are

generated for the client and server, respectively, to be used as secret parameters. The format of the application message after applying the MACs becomes:

$$MAC\_record \;:=\; HMAC(mk, \{i, t\}),$$

where mk is a MAC key (cm or sm), i is a sequence number that orders the messages sent by the server and the client into two separate streams, and t is a Application Layer message.

- Encryption is applied using two symmetric encryption keys for the client and server, denoted by ce and se, respectively, which are generated during the Handshake protocol.

The final Record Layer messages take the form:

$$[t, i]_{mk,ek} \;=\; \{t, HMAC(mk, \{i, t\})\}_{ek},$$

where mk is a MAC key (cm or sm, respectively) and ek is a symmetric encryption key (ce or se, respectively).

2.2 The Handshake Protocol

According to the TLS version 1.0 specification document [DA99], the Handshake Protocol provides a connection that reliably negotiates a shared secret and authenticates the peers' identities using public key cryptography. This authentication can be unilateral or bilateral. We only consider the TLS Protocol in the bilateral mode.

We now describe the steps of the Handshake Protocol.

ClientHello This is the first message sent by a client in the TLS Handshake Protocol. It communicates the client's 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; and *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 the *client_nonce*, and an optional session identification number *sessionID* that is used for session resumption purposes.

ClientHello. $c \longrightarrow s$: *client_version*, *client_nonce*, *sessionID_length*, *sessionID*, *cipher_suite_length*, *cipher_suites*, *compression_length*, *compression_methods*

ServerHello With this message, the server makes its choice out of the preferences offered by the client, including the cipher suite and compression algorithm that will be used for this connection. The message also includes a fresh random number referred to as the *server_nonce*, and returns the non-zero session identification number *sessionID* received from the client if a previous session is to be resumed.

ServerHello. $s \longrightarrow c$: *server_version*, *server_nonce*, *sessionID_length*, *sessionID*, *cipher_suite*, *compression_method*

ServerCertificate This message contains a chain of X.509 certificates, *server_cert_chain*, 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.

ServerCertificate. $s \longrightarrow c$: *cert_chain_length*, *server_cert_chain*

ServerKeyExchange This message includes the server's key exchange parameters. Recall that we are considering only the public-key mode of TLS in this paper, so the key exchange parameters are simply the server's public key. (Alternative modes pass Diffie-Hellman or Fortezza parameters.) This public key is concatenated with the *client_nonce* and *server_nonce*. The concatenated terms are hashed and signed by the signature key $SK(s)$ that corresponds to the public key provided in the **ServerCertificate** message. The signed term is denoted by *signed_parameters* and has the form: $\{Hash(client\_nonce, server\_nonce, PK(s))\}_{SK(s)}$.

ServerKeyExchange. $s \longrightarrow c$: $PK(s)$, *signed_parameters*

CertificateRequest 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, *cert_types*, and the certificate authorities recognised by the server, *cert_auths*.

CertificateReq. $s \longrightarrow c$: *cert_type_length*, *cert_types*, *cert_auths_length*, *cert_auths*

ServerHelloDone 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.

ServerHelloDone. $s \longrightarrow c$:

ClientCertificate This message is sent by the client after receiving a **ServerHelloDone** message if the server has previously sent a **CertificateRequest** message. The message includes the certificate chain of the client.

ClientCertificate. $c \longrightarrow s$: $cert\_chain\_length, client\_cert\_chain$

ClientKeyExchange Just as the **ServerkeyExchange** message provides the key information for the server, the **ClientKeyExchange** tells the server the client's key information. In public-key mode, 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. (Alternative modes pass Diffie-Hellman or Fortezza parameters.)

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

CertificateVerify 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 all the handshake messages exchanged prior to this message, denoted $handshake\_messages[1-8]$. The signature is then verified by the server.

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

Key generation The previous messages have the aim of establishing the premaster secret as a shared secret, together with agreement on the client's and server's nonces. The keys subsequently used in the Record Layer are formed from these three values. In slightly more detail, the *master secret* is calculated from the premaster secret and the two nonces using a pseudo-random function *PRF*:

$$master\_secret = PRF(premaster\_secret, \text{“}master\ secret\text{”}, client\_nonce, server\_nonce).$$

The *key material* is then calculated from the master secret and the two nonces using a further application of "PRF". Finally, the key material is split to provide the client's and server's encryption keys *ce* and *se*, MAC keys *cm* and *sm*, and initialisation vectors for block chaining. See [DA99, Tho00] for the gory details.

The subsequent messages are passed to the Record Layer to be protected using the negotiated cipher suite. The recipients must verify that the contents are correct.

ClientFinished 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 (using the pseudo-random function PRF) of the negotiated *master_secret* along with all the prior handshake messages ($handshake\_messages[1-9]$) hashed using the functions SHA and $MD5$. This is encapsulated inside a Record Layer message (the "0" represents that this is the zeroth message in this stream).

$$\textbf{ClientFinished}.\quad c \longrightarrow s:\ [PRF(master\_secret, \text{“}client\ finished\text{”}, MD5(handshake\_messages[1-9]), SHA(handshake\_messages[1-9])), 0]_{cm,ce}$$

ServerFinished This 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.

$$\textbf{ServerFinished}.\quad s \longrightarrow c:\ [PRF(master\_secret, \text{“}server\ finished\text{”}, MD5(handshake\_messages[1-10]), SHA(handshake\_messages[1-10])), 0]_{sm,se}$$

The **ClientFinished** and **ServerFinished** messages provide confirmation that the handshake has been successful and that the client and server agree upon the keys, but do not otherwise add much to the protocol.

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 application data over the established connection.

2.3 Simplifying the Handshake Protocol

We now transform the protocol, to simplify the subsequent analysis. These transformations follow the theory of *fault-preserving simplifying transformations* from [HL01]: they remove redundant information present in the protocol, whilst not losing any possible attacks. The end result is a fault-preserved version of the original protocol, whereby any attack on the original corresponds to an attack upon the simplified version. In [HL01], certain transformations were proved to preserve faults in this way. Later, we will prove that the simplified protocol is secure, and hence deduce that the original protocol is also secure. We will formalise the relationship between the full and simplified protocols in Section 3.3.

Like most industrial protocols, TLS 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.

We transform the protocol to remove some atomic fields from the protocol (the *removing atomic fields* transformation from [HL01]). Intuitively, these transformations can only be applied on values upon which we do not check for agreement, such as *message_length*, *cipher_suite_length*, *cipher_suite*, *content_type*, etc. We also remove the session-related fields *sessionID* and *sessionID_length* since we do not consider session resumption in our analysis. Within the public key certificates, we remove all fields except the public key and the identity of the principal. 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. For clarity, all the fields in the protocol are now given shorter names.

We now transform the protocol to replace certain applications of hash functions $h(m)$ by their unhashed versions m (the *removing hash functions* transformation from [HL01]). We apply this transformation to remove from the **Finished** messages the applications of the applications of *MD5* and *SHA*, and the application of *PRF* that produces the master secret. The resulting messages contain much duplication, so we remove the duplicates (the *coalesce messages* transformation from [HL01]). We encapsulate the result using two functions:

$$\begin{aligned} PRF_{cf}(pm, prev) &= PRF(pm, \text{“}client\ finished\text{”}, prev), \\ PRF_{sf}(pm, prev) &= PRF(pm, \text{“}server\ finished\text{”}, prev). \end{aligned}$$

We similarly encapsulate the production of the encryption and MAC keys into four functions:

$$\begin{aligned} cm &:= G_0(pm, r_c, r_s), & sm &:= G_1(pm, r_c, r_s), \\ ce &:= G_2(pm, r_c, r_s), & se &:= G_3(pm, r_c, r_s). \end{aligned}$$

We assume these functions have the properties of cryptographic hash functions (i.e. preimage resistance, second-preimage resistance and collision resistance [MvOV97]) in what follows.

The simplified version of the Handshake Protocol is then as follows:

Message 1. $c \longrightarrow s : \ r_c$
Message 2. $s \longrightarrow c : \ r_s$
Message 3. $s \longrightarrow c : \ server\_certificate\_chain$
Message 4. $c \longrightarrow s : \ client\_certificate\_chain$
Message 5. $c \longrightarrow s : \ \{pm\}_{PK(s)}$
Message 6. $c \longrightarrow s : \ \{VH(prev_5)\}_{SK(c)}$
Message 7. $c \longrightarrow s : \ [PRF_{cf}(pm, prev_6), 0]_{cm,ce}$
Message 8. $s \longrightarrow c : \ [PRF_{sf}(pm, prev_7), 0]_{sm,se}$

where VH, PRF_{cf} and PRF_{sf} are cryptographic hash functions, and $prev_5$, $prev_6$ and $prev_7$ represent the previous five, six and seven messages.

Certificate chains We now describe the certificate chains in more detail. The certificate chain is a list of public key certificates. The first certificate is that of the sender of the message. Each certificate except the last is signed (certified) by the secret key corresponding to the following certificate. The final certificate is (expected to be) that of the root certificate authority, and is self-signed (i.e. signed by the secret key corresponding to the public key that it certifies).

Given the simplifications we have made to the certificates above, the certificate chain for principal p, with certificates signed by v_0 (the root authority), $v_1, \ldots, v_{n-1}$ is expected to be of the form

$$\{p, PK(p)\}_{SK(v_{n-1})}, \{v_{n-1}, PK(v_{n-1})\}_{SK(v_{n-2})}, \ldots, \\ \{v_1, PK(v_1)\}_{SK(v_0)}, \{v_0, PK(v_0)\}_{SK(v_0)},$$

where $n \geq 1$. However, we do not want to assume *ab initio* that the chain is of that form, for to do so would be to assume that the penetrator cannot interfere with the certificate chain, to generate fake certificates. We therefore take the certificate chain to be of the form

$$\{p, k_p\}_{k_{n-1}^{-1}}, \{v_{n-1}, k_{n-1}\}_{k_{n-2}^{-1}}, \ldots, \{v_1, k_1\}_{k_0^{-1}}, \{v_0, k_0\}_{k_0^{-1}},$$

(where k^{-1} represents the secret key corresponding to public key k) and we will seek to prove (under suitable assumptions) that $k_p = PK(p)$.

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 [GT02, DGT07b], which are used in our analysis of TLS in the next section. Further, we adapt the idea of fault-preserving transformations from [HL01] to the strand spaces setting, and hence justify the simplifications we made to TLS above.

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 [GT02].

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: $G : \mathcal{T} \times \mathcal{T} \ldots \times \mathcal{T} \to \mathcal{K}$. We assume each key generation function is injective (i.e. collision-free), and distinct key generation functions have disjoint ranges. If a key k is in $\mathsf{ran}\, G$ for some $G \in \mathsf{kgf}$, 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, p, v and ca range over $\mathcal{T}_{name}$; k, k_c, k_s, etc, range 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_c, r_s, pm 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 1 Compound terms are built by three constructors:

- $\mathsf{encr} : \mathcal{K} \times \mathcal{A} \rightarrow \mathcal{A}$ representing encryption.
- $\mathsf{join} : \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}$ representing concatenation.
- $\mathsf{hash} : Hash \times \mathcal{A} \rightarrow \mathcal{A}$ representing hashing to digest messages, where $Hash$ is a set of hash functions.

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 3.2.

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 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 $r \sqsubseteq t_1$;.

Note that the subterms of a term t are just those terms that can feasibly be extracted from t given knowledge of appropriate keys, so does not include encrypting keys or the contents of hashed terms.

Definition 3 [GT02] 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 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 4 [THG98] A directed term is a pair $\langle \sigma, a \rangle$ with $\sigma \in \{+, -\}$ and $a \in \mathcal{A}$. A directed term is written as $+t$ or $-t$. $(\pm\mathcal{A})^*$ is the set of finite sequences of directed 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 $n_1 \to n_2$ 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 st. $n' \Rightarrow^+ n$ is used to denote that n' precedes n (not necessarily immediately) on the same strand.
4. An undirected term t *occurs in* $n \in \mathcal{N}$ iff $t \sqsubseteq msg(n)$.
5. Suppose $\mathcal{I}$ is a set of undirected 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 undirected 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 undirected 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 undirected 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.)

We now define strands for the roles of the TLS Protocol. We start by defining a function *RECMS* that models the changes the Record Layer applies to a stream of Application Layer messages. 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 Application 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 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:

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

We also define some notation to help with defining certificate chains.

Definition 6 If

$$ch\ =\ \langle (p_n, k_n), (p_{n-1}, k_{n-1}), \ldots, (p_1, k_1), (p_0, k_0) \rangle$$

is a sequence of (principal, public key) pairs, with $n \geq 1$, then we write

$$
\begin{array}{l}
cert\_chain(ch) = \\
\quad \{p_n \frown k_n\}_{k_{n-1}^{-1}} \frown \{p_{n-1} \frown k_{n-1}\}_{k_{n-2}^{-1}} \frown \ldots \frown \{p_1 \frown k_1\}_{k_0^{-1}} \frown \{p_0 \frown k_0\}_{k_0^{-1}}
\end{array}
$$

for the corresponding certificate chain.

As explained in Section 2, the TLS protocol has two primary roles, the client c and the server s; we also include in the model a secondary role of certificate authorities, which originate public key certificates. We define a set of strands for each of these roles.

Definition 7 Suppose ch_s, ch_c are as in Definition 6, with $\mathit{first}(ch_s) = (s, k_s)$, $\mathit{first}(ch_c) = (c, PK(c))$. Let $\mathit{Client}[c, s, r_c, r_s, k_s, ch_s, ch_c, pm, Sms]$ be the set of client strands whose trace is:

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

(Note that the assumption $\mathit{first}(ch_s) = (s, k_s)$, $\mathit{first}(ch_c) = (c, PK(c))$ applies for every $\mathit{Client}[c, s, r_c, r_s, k_s, ch_s, ch_c, pm, Sms]$ strand we consider from now on.)

Definition 8 Suppose ch_s, ch_c are as in Definition 6, with $\mathit{first}(ch_s) = (s, PK(s))$, $\mathit{first}(ch_c) = (c, k_c)$. Let $\mathit{Server}[s, c, r_c, r_s, ch_s, ch_c, k_c, pm, Sms]$ be the set of server strands whose trace is:

$$\begin{array}{ll}\langle - & r_c, \\ + & r_s, \\ + & \mathit{cert\_chain}(ch_s), \\ - & \mathit{cert\_chain}(ch_c), \\ - & \{pm\}_{PK(s)}, \\ - & \{VH(prev_5)\}_{k_c^{-1}}, \\ - & [PRF_{cf}(pm \frown prev_6), 0]_{cm,ce}, \\ + & [PRF_{sf}(pm \frown prev_7), 0]_{sm,se} \rangle \end{array}$$
$$\frown \mathit{RECMS}\ (sm, se, cm, ce)\ Sms.$$

Definition 9 Let $CA[ca, a]$ be the set of certificate authority strands whose trace is:

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

For simplicity, we assume that the public key certificates 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, $\mathit{Client}[c, s, r_c, r_s, k_s, ch_s, ch_c, *, *]$ is shorthand for $\bigcup_{pm,Sms} \mathit{Client}[c, s, r_c, r_s, k_s, ch_s, ch_c, pm, Sms]$. We shall refer to regular primary nodes according to their position in a client or a server strand; for example writing Client_2 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 10 [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 1 illustrates a bundle representing a single run of the Handshake Protocol.

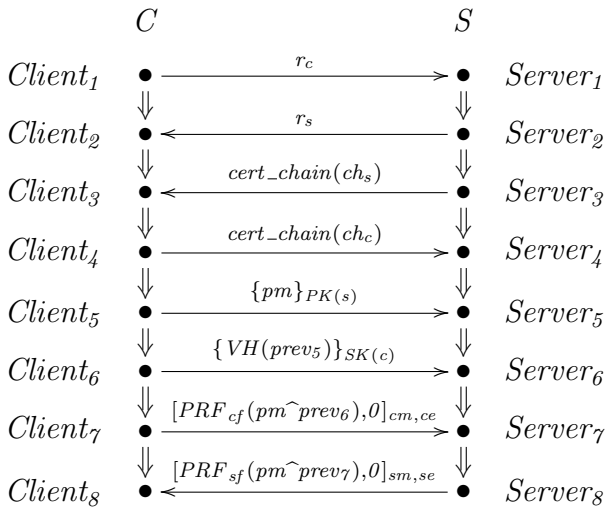


Figure 1: A bundle representing a single execution of the Handshake Protocol.

Definition 11 [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 $n \in \mathcal{N}_{\mathcal{B}}$. 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 1 is *8*. We say that a strand *st* is in $\mathcal{B}$ if at least some of its nodes are in $\mathcal{B}$, i.e. its $\mathcal{B}$-height is positive.

Definition 12 If $\mathcal{B} = \langle \mathcal{N}_\mathcal{B}, \rightarrow_\mathcal{B} \cup \Rightarrow_\mathcal{B} \rangle$ is a bundle then $\preceq_\mathcal{B} = (\rightarrow_\mathcal{B} \cup \Rightarrow_\mathcal{B})^*$.

$\preceq_\mathcal{B}$ can be considered as a causal precedence relationship, because $n \preceq_\mathcal{B} n'$ 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].

Proposition 13 [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.

3.1.3 The Penetrator

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

The powers of the penetrator (sometimes referred to as the intruder or adversary) are characterised by three ingredients [THG98]:

- the set $\mathcal{K}_P$ of keys known initially to the penetrator;
- the set $\mathcal{T}_P$ of other atomic messages known initially to the penetrator;
- the set of strands that allow the penetrator to generate new messages from the messages 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. We add strands corresponding to hashing and key generation. The fact that there are no strands to undo these operations captures their one-way nature. The fact that the operators are treated algebraically (and that their arguments are required for the penetrator to produce their output) captures their collision resistance.

Definition 14 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: $\langle -t_0 \frown t_1, +t_0, +t_1 \rangle$.

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 $h \in \mathit{Hash}$.

KG. Key generation: $\langle -r_1, \ldots, -r_n, +G(r_1, \ldots, r_n) \rangle$, where $r_1, \ldots, r_n \in \mathcal{T}$ are atoms, 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 [GT02]:

Definition 15 Bundles $\mathcal{B}$, $\mathcal{B}'$ on a strand space Σ are equivalent iff they have the same regular nodes.

3.2 The Normal Form Lemma

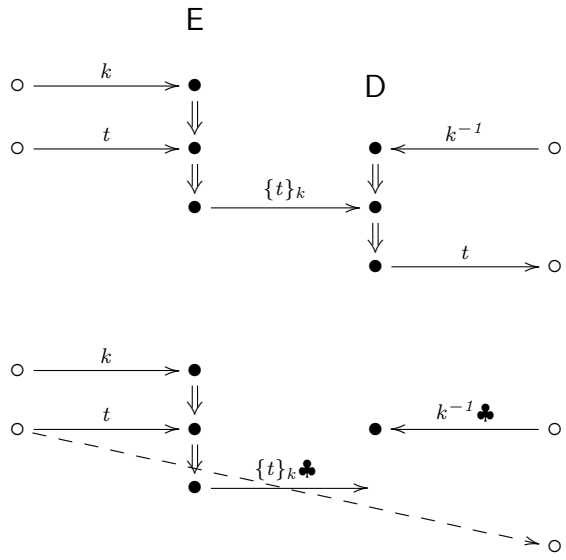
In Section 3.1.3, we specified that the penetrator can perform several operations represented by strands; these operations can be carried out in any order. This is complicated by the unbounded number of protocol sessions. Consequently, the presence of the penetrator can make the problem of security protocols analysis difficult.

In [GT02, GT00a], Guttman and Thayer restrict the order of the penetrator strands in order to assist the analysis of security protocols. An essential element in this restriction is the *Normal Form Lemma*. In this section we extend this restriction to the extended penetrator model developed previously. We follow closely the reasoning of Thayer *et al.* in [GT02] and therefore complete proofs are not provided.

Firstly, we remove some of the redundancies in the penetrator's behaviour without weakening his powers. There are two types of redundancies [GT02]:

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 2.

2. C-S redundancies: Here the penetrator concatenates two values t_0 and t_1 to form $t_0 \hat{} t_1$, and then separates the concatenated term into its subterms. This type of redundancy can be eliminated as shown in Figure 3.



♣ Discarded messages

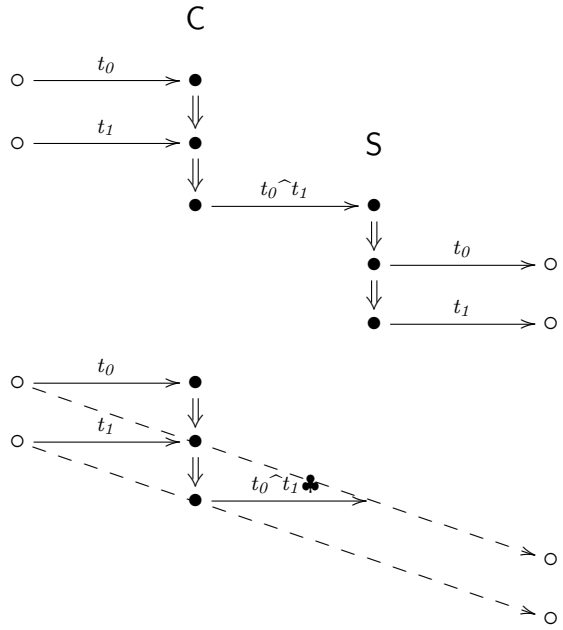
Figure 2: E-D redundancies and how to eliminate them [GT02].

By eliminating the above redundancies we end up with an equivalent bundle since we only remove penetrator nodes:

Proposition 16 [GT02] Every bundle is equivalent to a bundle with no redundancies of type C-S and E-D.

The penetrator's activity can be described using the notion of a *path* [GT02]. A path p through the bundle $\mathcal{B}$ is any finite sequence of nodes and edges where for each consecutive pair m and n either $m \Rightarrow^+ n$ with $msg(m)$ negative and $msg(n)$ positive, or else $m \rightarrow n$. A *penetrator path* is one in which all nodes other than possibly the first or the last node are penetrator nodes.

In order to restrict the pattern of penetrator paths, we use the notion of *constructive* and *destructive* edges. Our definition of constructive and destructive edges is slightly different from the one provided in [GT02] since we refer to the extended penetrator model in Definition 14.



♣ Discarded messages

Figure 3: C-S redundancies and how to eliminate them [GT02].

Definition 17 A $\Rightarrow^+$ edge is *constructive* if it is a part of an 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 cryptographically quite similar, we define H to be constructive and the key generation strands KG, to be destructive. The reason for this will become clear below.

Proposition 18 In a bundle, a constructive edge immediately followed by a destructive edge has one of the following two forms:

1. Part of an E strand immediately followed by part of a D strand;
2. Part of a C strand immediately followed by part of an S strand.

Proof: This result is proved in [GT02] without the H and KG strands. The extension to these latter strands is straightforward: the output of an H strand cannot be passed to a destructive edge, since hash functions are uninvertible; the input of a KG cannot be provided by a constructive edge, since the inputs to key generation strands are atoms. □

Note that this result would not be true if we had defined 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 4), in which case a constructive edge would be followed by a destructive edge, and the Normal Form Lemma, below, would not hold.

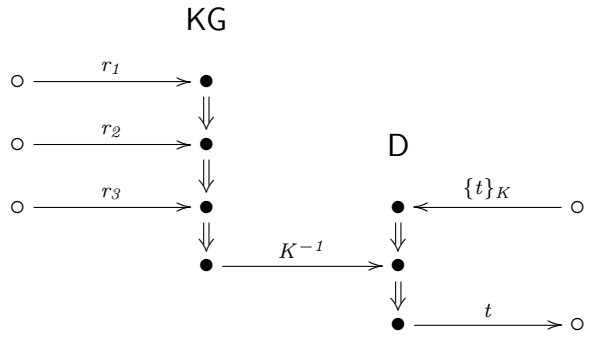


Figure 4: Considering KG strands as constructive, the Normal Form Lemma would not hold for the above bundle.

Definition 19 [GT02] A bundle $\mathcal{B}$ is normal if for every penetrator path of $\mathcal{B}$, every destructive edge precedes every constructive edge.

Proposition 20 (**Penetrator Normal Form Lemma**) For every bundle $\mathcal{B}$ there exists an equivalent normal bundle $\mathcal{B}'$.

The proof is the same as in [GT02], and follows from Propositions 16 and 18.

3.3 Fault-Preserving Simplifying Transformations

We now adapt the theory of fault-preserving simplifying transformations from [HL01] to the strand spaces model, and hence explain the relationship between our strand spaces model of the simplified protocol and a more faithful model of the full protocol.

Definition 21 If $t \in \mathcal{A}$ is a term, and $S \subseteq \mathcal{A}$ is a set of terms, we write $S \vdash t$ if there is a web of penetrator strands (i.e. a collection of strands, connected by $\rightarrow$ edges) that produces t, where S gives the inputs to the web, i.e. the terms received from outside the web and the terms on the initial penetrator strands (i.e. M and K strands).

A transformation on a protocol can be defined by a renaming function $f : \mathcal{A} \to \mathcal{A}$ over terms. The idea is that all the messages in the protocol are renamed according to f. We will require two properties of the transformation f. Firstly, f must preserve deductions:[1]

$$\forall S \subseteq \mathcal{A}, t \in \mathcal{A} \bullet S \vdash t \Rightarrow f(S) \vdash f(t). \tag{1}$$

The second property concerns the initial knowledge of the penetrator. When the transformed protocol is analysed, the penetrator's initial knowledge $\mathcal{T}_P \cup \mathcal{K}_P$ should be replaced by a new value $\mathcal{T}'_P \cup \mathcal{K}'_P$ such that:

$$f(\mathcal{T}_P \cup \mathcal{K}_P) \subseteq \mathcal{T}'_P \cup \mathcal{K}'_P. \tag{2}$$

Proposition 22 Suppose $\mathcal{B}$ is a bundle with initial penetrator knowledge $\mathcal{T}_P \cup \mathcal{K}_P$. Suppose f satisfies equations (1) and (2) (for $\mathcal{T}'_P$ and $\mathcal{K}'_P$). Then there is a bundle $\mathcal{B}'$ with initial penetrator knowledge $\mathcal{T}'_P \cup \mathcal{K}'_P$ obtained by renaming terms on regular strands by f; i.e. there is a bijection ϕ from the regular nodes of $\mathcal{B}$ to those of $\mathcal{B}'$ such that $msg(\phi(n)) = f(msg(n))$ for each regular node n in $\mathcal{B}$. Further, if term t appears on a penetrator node in $\mathcal{B}$, then $f(t)$ appears on a penetrator node in $\mathcal{B}'$.

Proof: We construct the regular strands of $\mathcal{B}'$ by renaming the regular strands of $\mathcal{B}$ under f. For each initial penetrator strand in $\mathcal{B}$ with term t, we add a corresponding initial penetrator strand with term $f(t)$ in $\mathcal{B}'$; equation (2) justifies this. Let ϕ be the corresponding bijection from the regular and initial nodes of $\mathcal{B}$ to those of $\mathcal{B}'$, so $msg(\phi(n)) = f(msg(n))$ for each regular or initial node n in $\mathcal{B}$.

For the connecting penetrator strands, consider a particular term t' received at a regular node n' in $\mathcal{B}'$. Let t and n be the corresponding term and node in $\mathcal{B}$, so $t' = f(t)$, $n' = \phi(n)$. In $\mathcal{B}$, t is formed by the penetrator from the terms on the regular and initial nodes $\{n_0 \mid n_0 \preceq_{\mathcal{B}} n\}$, i.e. $\{msg(n_0) \mid n_0 \preceq_{\mathcal{B}} n\} \vdash t$. Hence by equation (1), $\{f(msg(n_0)) \mid n_0 \preceq_{\mathcal{B}} n\} \vdash f(t)$; i.e. $\{msg(\phi(n_0)) \mid n_0 \preceq_{\mathcal{B}} n\} \vdash t'$. Hence we can produce penetrator strands in $\mathcal{B}'$ to produce t' from the corresponding regular and initial strands $\{\phi(n_0)) \mid n_0 \preceq_{\mathcal{B}} n\}$.

Finally, for each term t that appears on a penetrator node in $\mathcal{B}$, we can similarly produce penetrator strands in $\mathcal{B}'$ to produce $f(t)$. □

[1] We lift f to sets by point-wise application.

(We consider $\mathcal{B}$ and $\mathcal{B}'$ to be in distinct strand spaces, as they contain different types of strands.)

In [HL01], various transformations are shown to satisfy equations (1) and (2)[2]; these include the transformations we used on TLS in Section 2.3 (namely remove atomic fields, remove hash functions, and coalesce messages). Hence, for every bundle corresponding to the full TLS protocol, there is a corresponding bundle for the simplified protocol, where terms on regular strands have been appropriately renamed.

We would like to deduce properties ψ for all bundles $\mathcal{B}$ of the full protocol by proving corresponding properties ψ' for all bundles $\mathcal{B}'$ of the simplified protocol. This deduction will be valid for properties such that a failure of ψ in the original bundle $\mathcal{B}$ implies a failure of ψ' in the transformed bundle $\mathcal{B}'$ of Proposition 22.

In particular, many properties state that certain terms may not appear on penetrator nodes, i.e. the penetrator may not learn these terms. Proposition 22 shows that the appearance of such a term t in a bundle of the full protocol would be reflected by the appearance of the term $f(t)$ in a bundle of the simplified protocol. Hence if we verify that no such term $f(t)$ appears in bundles of the simplified protocol, we may deduce the desired property of the full protocol.

Similarly, many properties state that bundles may not contain particular combinations of regular strands, such as a client strand with no corresponding server strand. Failures of such properties will normally be preserved by the transformation, since Proposition 22 shows that the regular strands are preserved by the transformation, modulo renaming by f: hence in order to deduce such a property in the full protocol, essentially the same property needs to be verified in the simplified protocol. If it is required that particular strands agree upon the values of certain terms (such as keys), then we should choose f to be injective on those terms (so that the strands for the simplified protocol agree upon those terms only if the strands of the full protocol do).

In our analysis of TLS, many of the properties (all properties up to Lemma 42) will be of one of the forms described in the previous two paragraphs. Our analysis will therefore show that corresponding results hold for the full protocol. It is important here that our simplifying transformations are injective on the critical terms, such as keying material.

Unfortunately, our later results (about the Record Layer) cannot be handled in the same way, since failures of the properties in the full protocol are not necessarily reflected by failures of corresponding properties in the

[2]The relation $\vdash$ was defined differently in [HL01] from how we defined it above; however, it is straightforward to prove that the two definitions are equivalent.

simplified protocol. There are essentially two reasons for this:

- Our simplifying function might not be injective on the Record Layer messages, in particular if their payloads may contain fields that we remove in the simplifications, such as cipher suites.
- Some of the later theorems talk about *paths* in bundles, and Proposition 22 says nothing about the preservation of paths.

However, for these later results, essentially the same proofs hold for both the full and simplified protocols: these proofs talk only about Record Layer messages, and the simplifications to Record Layer messages have been very minor (most of the simplifications are to the Handshake messages). We present just the proofs for the simplified protocol, for ease of exposition.

3.4 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 10) 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 [GT02].

To illustrate an authentication goal, consider a bundle $\mathcal{B}$ in a TLS strand space. Then the Handshake goal of authenticating the server to the client can be captured as follows: if st_c is a client strand in $\mathit{Client}[c, s, r_c, r_s, k_s, ch_s, ch_c, pm, *]$ 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 $st_s \in \mathit{Server}[s, c, r_c, r_s, PK(c), ch_s, ch_c, pm, *]$ 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 [GT02]. 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 $\mathit{Client}[c, s, r_c, r_s, k_s, ch_s, ch_c, pm, *]$ 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 $n \in \mathcal{B}$, $msg(n) \neq pm$.

3.5 Proving Secrecy and Authentication

In this section, we discuss techniques 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.

3.5.1 Safe Keys

We can define 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 encrypted terms R and hashes in t if in the abstract syntax tree of t, every path from the root to an occurrence of t_0 as a subterm of t 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. Recall also that the body t_0 of a hashed term $t = hash(h, t_0)$ is not considered a subterm of t, so this condition includes the possibility that t_0 occurs only within hashes; similarly, recall that the encrypting key in an encrypted term is not considered a subterm. So, for example, t_0 occurs only within $\{\{t_0\}_k\}$ and hashes in $hash(h, t_0)\,\widehat{}\,\{t_0\}_k\,\widehat{}\,\{a\}_{t_0}$. We define the property formally as follows:

Definition 23 We say that t_0 occurs only within R and hashes 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 and hashes in t_1; or (b) $t = t_1\,\widehat{}\,t_2$ and t_0 occurs only within R and hashes in both t_1 and t_2.

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

On the other hand, t_0 *occurs outside* R and hashes in t if t_0 is a subterm of t and t_0 does not occur only within R and hashes 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 can now define the set $\mathcal{S}$ of safe keys, i.e., keys that the penetrator cannot obtain. Our definition differs from that 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, has at least one safe ingredient.

Definition 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 hashes 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 $\{\{t\}_k \mid k^{-1} \in \mathcal{M}_i(\mathcal{B})\}$ and hashes in $\mathcal{B}$; and (3) if a is complex, of the form $G(r_1, \ldots, r_n)$, then at least one of the ingredients $r_1, \ldots, r_n$ 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 component is protected by a hash or an encryption with some key k such that $k^{-1} \in \mathcal{M}_i(\mathcal{B})$; in most protocols, i is typically *0* or *1*.

Proposition 25 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: (sketch). We can prove by induction on i that if r is an atom in $\mathcal{M}_i(\mathcal{B})$ and $r \notin \mathcal{K}_p \cup \mathcal{T}_p$, then for every equivalent bundle $\mathcal{B}'$ there is no node $n \in \mathcal{B}'$ such that $msg(n) = r$. The proof proceeds by a straightforward case analysis over the ways in which r could be produced. □

3.5.2 Authentication Tests

In [GT00a], Thayer and Guttman introduced the concept of *Authentication Tests* to prove the authentication goals of security protocols. 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 among the most powerful tools of the strand space model because of their simplicity.

In Section 4.3 we will apply the Incoming Test to the Handshake Protocol; we do not use the Outgoing Test, so do not discuss it further here. We extend the Incoming Test to include hashed values.

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: (Based on [DGT07a].) Let $\Phi = \{n \in \mathcal{B} \mid t \sqsubseteq msg(n)\}$. Φ is non-empty since $n_1 \in \Phi$, so has at least one $\preceq_{\mathcal{B}}$-minimal element, denoted by m_1. Suppose, for a contradiction, that m_1 is a penetrator node. Since $t \sqsubseteq msg(m_1)$ we can deduce that m_1 must 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 a regular node. □

The node m_1 is called an *incoming transforming node* and n_1 as an *incoming test node* [DGT07b]. In Lemma 38 we show that $Client_8$ is an incoming test node, and use this to establish authentication guarantees for the client. Similarly, in Lemma 39 we show that $Server_6$ is an incoming test node, and use this to establish authentication guarantees for the server.

4 Security Analysis of TLS

In this section we analyse TLS, and then formalise and prove the security services it provides. We use the simplified version of TLS provided in Section 2 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, and show how it can be captured in the Strand Spaces Model. 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 Application Layer protocols. In Section 4.3 we analyse the Handshake protocol, obtaining guarantees for both the client and the server. In Section 4.4 we analyse the Record Layer, verifying authentication, secrecy and session independence properties. Finally, in Section 4.5 we prove a result that shows that (under certain assumptions) TLS does not interfere with the Application Layer protocol that is layered on top of it.

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 Requirements for TLS

In this section we describe the requirements for TLS, which we verify in later sections. The requirements are based on Broadfoot and Lowe's abstract model of TLS from [BL03], which describes the security services those authors believe TLS provides. Those requirements were presented as a trace-based specification, i.e. a specification in terms of a property that all traces of TLS should satisfy; we describe below how those requirements should be adapted to the strand spaces model.

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 Application 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 Application Layer at B as part of session s.

The authentication and integrity requirements from [BL03] are that within each session between A and B, the messages accepted by the Application Layer of B as being from A are guaranteed to be sent by A, intended for B, and sent in that particular order. Using CSP notation with the above channels, the sequence of messages accepted by B from A in session s is denoted $tr \downarrow receive.B.A.s$ (i.e. the sequence of data passed on channel $receive.B.A.s$); likewise, the sequence of messages sent by A to B in session s is denoted $tr \downarrow send.A.B.s$. Hence the authentication and integrity requirements are expressed as follows:

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

where $\leq$ denotes the prefix relation and *Traces* is the set of traces exhibited by the system. This property is called *stream* or *prefix* authentication.

We now consider how the above requirement can be expressed in the strand spaces model. In strand spaces, a regular strand representing an honest agent models the transmissions and receptions involving that agent in a session [Hea02]. Consequently, if A, B are honest agents, $send.A.B.s.t$ corresponds to a positive node n in a regular strand st_A representing A such that st_A is apparently communicating with B, such that $msg(n)$ is of the form $+[t, i]_{km,ke}$ for some $km \in \mathcal{K}_{MAC}$, $ke \in \mathcal{K}_{Enc}$, and $i > 0$. Similarly, $receive.B.A.s.t$ corresponds to a negative node n in a regular strand st_B representing B such that st_B is apparently communicating with A, and $msg(n)$ is of the form $-[t, i]_{km,ke}$. Two messages can be considered part of the same session iff they are on the same regular strand.

Given a client strand in $Client[c, s, *, *, *, *, *, *, Sms]$ or server strand in $Server[s, c, *, *, *, *, *, *, Sms]$, we write $sent(Sms)$ and $received(Sms)$ for the sent and received messages of Sms:[3]

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

We will then capture the stream authentication property by proving (Theorem 1) that, subject to some assumptions:

- for each client strand $st_c \in Client[c, s, r_c, r_s, k_s, ch_s, ch_c, pm, Sms]$, there is a unique server strand $st_s \in Server[s, c, r_c, r_s, k_c, ch_s, ch_c, pm, Sms']$ such that $received(Sms) \leq sent(Sms')$, and
- for each server strand $st_s \in Server[s, c, r_c, r_s, k_c, ch_s, ch_c, pm, Sms]$, there is a unique client strand $st_c \in Client[c, s, r_c, r_s, k_s, ch_s, ch_c, pm, Sms']$ such that $received(Sms) \leq sent(Sms')$.

The confidentiality requirement from [BL03] is that if the penetrator P is able to cause an honest agent A to accept a message m from him, then m can be produced from the penetrator's initial knowledge PIK and those messages that have previously been deliberately sent to him:

$$\begin{aligned} &\forall A : Honest; P : Dishonest; s : Session; m : Message; tr : Traces \bullet \\ &\quad tr' \frown \langle receive.A.P.s.m \rangle \leq tr \Rightarrow PIK \cup sentToPenetrator(tr') \vdash m. \end{aligned} \tag{3}$$

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

[3]We use standard functional programming notation for sequence comprehensions, as in, e.g. Haskell.

In fact, TLS does not satisfy property (3)! Suppose the penetrator observes the client handshake nonce r_c from a regular strand, and then uses it as the message m that he sends to some regular agent A using his own identity; then this contradicts (3), since $PIK \cup sentToPenetrator(tr') \not\vdash r_c$. This illustrates that information can leak from the Handshake Protocol to the Application Protocol (although in a fairly unimportant way). In [BL03], this problem was recognised and avoided by explicitly including such nonces in the penetrator's initial knowledge. However, such an approach does not seem appropriate to our analysis in the current paper. Instead we prove that TLS satisfies properties that are very similar in spirit to this property.

The formalism (3) actually captures two rather different properties. First it captures the intuitive notion of confidentiality: that the penetrator cannot extract the contents of a record layer message — for if he could, he would be able to embed that contents within a message that he sends using his own identity, contradicting (3). Within the strand spaces model, we capture this property by proving (Theorem 2) that, subject to some assumptions, there is no path that starts at a Record Layer node n in a regular client or server strand, and includes a penetrator node whose message is a subterm of the Application Layer message of n.

The second property captured by (3) is *non-hijackability* (or *session independence*): If a TLS session is initiated with the penetrator using one of his identities, he cannot *hijack* any Application Layer messages observed in another TLS session between honest agents, and replay it in his own session; i.e., the two sessions are independent. To illustrate this idea, consider an alternative secure transport protocol that encodes application layer message m from B to A as $\{\{m\}_{PK(A)}\}_{SK(B)}$; then the penetrator could replace B's signature with his own to produce $\{\{m\}_{PK(A)}\}_{SK(P)}$, and then send that to A using his own identity; this would contradict (3).

We prove (Theorem 3) that, again subject to some assumptions, if a penetrator path starts at a regular Record Layer node n in a strand st_A apparently communicating with a regular strand st_B, and ends at a regular Record Layer node n', then n' must be in st_B and st_B is apparently communicating with st_A. This implies that if a penetrator path p ends at a negative regular node n'' that lies in a regular strand apparently communicating with the penetrator, then the message is created using only terms that the penetrator knows.

As noted above, information can leak from the Handshake Protocol to the Application Protocol, so that the Handshake Protocol *interferes* with the Application Protocol. We end by showing (Theorem 4) that, again subject to some assumptions, such interference is incidental: there is a bundle that is equivalent from the point of view of the Application Layer, without any such interference. This result can be used to justify an analysis of the Application

Layer protocol that abstracts away from the implementation of TLS and just models the services it provides.

In the following sections, we use the strand spaces model to prove that TLS provides stream authentication, secrecy, session independence and interference freedom for Application 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 Application Layer messages is not specified by TLS, the Application 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 Application Layer messages such that the correctness of TLS is independent of the Application 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*, and *origination* to the Application Layer level.

Definition 26 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 $msg(n) = \pm[t, i]_{mk,ek}$ where $mk \in \mathcal{K}_{MAC}$ and $ek \in \mathcal{K}_{Sym}$. The Application Layer term of n is then given by: $appmsg(n) = t$.
- An undirected term t originates in the Application Layer iff there exists a positive Record Layer node n such that $t \sqsubseteq appmsg(n)$, and for every Record Layer node n' such that $n' \Rightarrow^+ n$, $t \not\sqsubseteq appmsg(n')$.

We now state our assumptions using the concepts defined above. First, the fresh values used in the TLS Handshake Protocol are uniquely originating.

Assumption 27 For every regular client strand $st_c \in \mathit{Client}[c, *, r_c, *, *, *, *, pm, *]$ 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 $st_s \in \mathit{Server}[s, *, *, r_s, *, *, *, *, *]$ such that $SK(s) \notin \mathcal{K}_p$, r_s originates uniquely on node $\langle st_s, 2 \rangle$.

Secret keys and the premaster secret do not originate on regular Record Layer nodes:

Assumption 28 None of the following originates in the Application Layer:

- secret keys from $\mathcal{K}_{Sec}$;
- terms of the form $G_0(*, *, *)$, $G_1(*, *, *)$, $G_2(*, *, *)$ or $G_3(*, *, *)$;
- the premaster secret pm used in any *Client* strand's handshake.

In particular, this prevents the premaster secret that was used in the handshake subsequently being used within the Application Layer.

Any public key certificate that originates at a Record Layer node or a **Certificate** node (node 3 or 4 of a client or server strand) has the correct association between principal and public key, or is signed by a compromised key.

Assumption 29 Let $\mathcal{B}$ be a bundle in Σ, and $n \in \mathcal{B}$ a Record Layer node, or the third or fourth node of a client or server strand. If $\{pk^\frown a\}_{sk} \sqsubseteq msg(n)$ then $pk = PK(a)$ or $sk \in \mathcal{K}_P$.

4.2.2 Disjoint Encryption Assumptions

We now adapt disjoint encryption assumptions from [GT00b, GT02], to prevent interactions between the two layers of TLS. In particular, these assumptions forbid layering TLS on top of itself. To see why such assumptions are necessary, suppose the Application Layer Protocol contains a simple nonce challenge of the following form:

Message 1. $a \longrightarrow b : \{x\}_{PK(b)}$
Message 2. $b \longrightarrow a : x$

The penetrator can simply intercept message 5 of the Handshake Protocol and send it to the server as message 1 of the Application 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 30 $\{t\}_K$ 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 $\{t\}_K \sqsubseteq appmsg(n_2)$.

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 31 Let Σ be a TLS space. We assume the following:

1. The Application Layer protocol does not remove pm from the protection of the encryption with the server's key[4]: for every $n \Rightarrow^+ n' \in \Sigma_{RL}$, if n is negative, $appmsg(n)$ contains a subterm of the form $\{pm\}_{k_s}$ (for any pm and k_s), and n' is positive, then every occurrence of pm within $appmsg(n')$ is within a subterm $\{pm\}_{k_s}$.
2. For every node $n \in \Sigma_{RL}$, no subterm of $appmsg(n)$ is of the form $\{VH(prev_5)\}_{k_c}$ (for any $prev_5$ and k_c).
3. For every node $n \in \Sigma_{RL}$, no subterm of $appmsg(n)$ is of the form $[M, i]_{mk,ek}$ (for any M, i, mk and ek).

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.

[4]The statement of this assumption is based on the simplified version of TLS we are considering; in the full version, message 5 is of the form $\{3.1\hat{}pm\}_{k_s}$ (the "3.1" is a version number), so the assumption would have to be changed appropriately.

Lemma 32 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 28, there is no regular node n_2 such that k originates in the Application 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 two 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 33 Let $\mathcal{B}$ be a bundle in Σ. For every $n \in \mathcal{B}$, for every public key certificate $\{pk^\frown a\}_{sk} \sqsubseteq msg(n)$, either $pk = PK(a)$ or $sk \in \mathcal{K}_P$.

Proof: A certificate $\{pk^\frown a\}_{sk}$ 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 regular strand on a Record Layer node or a **Certificate** node. In this case Assumption 29 gives us the desired result.
3. A penetrator E strand. By Lemma 32, a penetrator can only use an initially known secret key to sign the certificate and therefore $sk \in \mathcal{K}_P$.

□

Definition 34 We say that the chain $ch = \langle (p, k_p), (v, k_v) \rangle^\frown ch'$ is *uncompromised*, written $uncomp(ch)$, if $k_v \notin \mathcal{K}_P$.

Lemma 35 Let $\mathcal{B}$ be a bundle in Σ.

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

Proof: Each regular strand specified above can only accept a public key certificate signed by a secret key k_v that is not in $\mathcal{K}_P$. By Lemma 33, if the certificate is signed by an uncompromised key then the public key included in the certificate is reliably associated with its owner. □

Note that the above lemma only insists that the key that verifies the principal's public key is uncompromised: it allows for compromised keys further up the chain. At first sight, this might suggest that the rest of the certificate chain is unnecessary. However, in implementations, the certificate chain, together with certificate revocation lists, etc., is designed to provide evidence that the verifying key indeed belongs to the verifier, and so is uncompromised. If keys further up the chain are compromised and on revocation lists, then an implementations is likely to reject the chain, anyway, since it will be lacking evidence that the key is uncompromised.

4.3.2 The Client's Guarantees

Firstly, we prove the secrecy of the premaster secret.

Lemma 36 Let $\mathcal{B}$ be a bundle in Σ, and let $st_c \in \mathit{Client}[c, s, r_c, r_s, k_s, ch_s, ch_c, pm, \mathit{Sms}]$ be a regular client strand in $\mathcal{B}$ such that $\mathit{uncomp}(ch_s)$ and $SK(s) \notin \mathcal{K}_P$. Then for all nodes $n \in \mathcal{B}$, $\mathit{msg}(n) \neq pm$.

Proof: If we prove that premaster secret only *occurs safely* in Σ then we have proved its secrecy by Proposition 25.

Let R be the set of terms $\mathsf{R} = \{\{t\}_K \mid K^{-1} \textit{ is safe}\}$. Let S be the set of regular nodes such that $S = \{n \mid pm \textit{ occurs outside } \mathsf{R} \text{ and hashes in } \mathit{msg}(n)\}$. Suppose, for a contradiction, that S is non-empty. Then by bundle induction (Proposition 13), it has a $\preceq$-minimal element m, which is positive. By Assumption 27, pm originates uniquely on Client_5. Therefore, $\mathit{Client}_5 \preceq m$. We perform a case analysis over m.

- Case $m = \mathit{Client}_5$. Then $\mathit{msg}(m) = \{pm\}_{k_s}$. By Lemma 35, $k_s = PK(s)$. 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 and hashes, so $m \notin S$.
- Case m is a positive Record Layer node. pm does not originate in the Application Layer, by Assumption 28. Further, by clause 1 of Assumption 31, no Application Layer edge transforms a shared encryption so that pm occurs outside R and hashes. Consequently, $m \notin S$.

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

We now show that the session keys remain secret.

Lemma 37 Let $\mathcal{B}$ be a bundle in Σ, and let $st \in \mathit{Client}[c, s, r_c, r_s, k_s, ch_s, ch_c, pm, \mathit{Sms}]$ be a regular client strand such that $SK(s) \notin \mathcal{K}_p$

and $uncomp(ch_s)$. 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 within hashes in the initial Handshake. Also, by Assumption 28 no such term originates in the Application Layer. It follows from the secrecy of the premaster secret (Lemma 36) and the definition of safe keys (Definition 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 38 Let $\mathcal{B}$ be a bundle in Σ, and $st_c \in Client[c, s, r_c, r_s, k_s, ch_s, ch_c, pm, *]$ be a client strand of $\mathcal{B}$-height at least 8, such that $SK(s) \notin \mathcal{K}_p$ and $uncomp(ch_s)$. Then there exists a unique server strand $st_s \in Server[s, c, r_c, r_s, PK(c), ch_s, ch_c, pm, *]$ of $\mathcal{B}$-height at least 8.

Proof: We prove that $msg(Client_8) = [PRF_{sf}(pm \frown prev_7), 0]_{sm,se}$ is an incoming authentication test. By Lemma 37, sm and se remain secret. Therefore, the term $[PRF_{sf}(pm \frown prev_7), 0]_{sm,se}$ must have originated in some regular node n. By clause 3 of Assumption 31, n is not a Record Layer node. Hence, by inspection of the Handshake protocol, n can only be $Server_8$ in some server strand $st_s \in Server[s', c', r'_c, r'_s, k'_c, ch'_c, ch'_s, pm', *]$.

We can deduce that $s' = s$, $c' = c$, $r'_c = r_c$, $r'_s = r_s$, $k'_c = PK(c)$, and $pm' = pm$, since all of these variables are included in $prev_7$; and we can deduce that $ch'_s = ch_s$ and $ch'_c = ch_c$ since $cert\_chain(ch_s)$ and $cert\_chain(ch_c)$ are included in $prev_7$ (and $cert\_chain$ is injective). Therefore, $st_s \in Server[s, c, r_c, r_s, PK(c), ch_s, ch_c, pm, *]$.

Now we want to prove that such st_s is unique. By Assumption 27, r_s originates uniquely in Σ in a server strand. Hence, there can be at most one such st_s. □

Recall that in Section 2.3 we simplified the protocol to remove a number of fields that we deemed unimportant for security purposes. It it clear that if we had kept any of these fields in the protocol, the above argument could also be used to show that the principals agreed upon the values of such fields (since they would have been included in $prev_7$). In particular, this would show that the protocol is not prone to a ciphersuite rollback attack, where the penetrator changes *cipher_suites* in the **ClientHello** message, causing the principals to accept a weaker ciphersuite than they would otherwise have done; early versions of SSL (the predecessor of TLS) were subject to such attacks [WS96].

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

Proof: As in Lemma 38, we show that $Server_6$ forms an incoming test node. By Lemma 35, $k_c = PK(c)$, 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 31, m_1 cannot be a Record Layer node. Hence, by inspection of the Handshake protocol, $m_1 = Client_6$ for some client strand $st_c \in Client[c', s', r_c', r_s', k_s', ch_s', ch_c', pm', *]$. As in Lemma 38 it is easy to prove that $k_c = PK(c)$, $c' = c$, $r_c' = r_c$, $r_s' = r_s$, $s' = s$, $k_s' = PK(s)$, $ch_s' = ch_s$, $ch_c' = ch_c$, and $pm' = pm$. Therefore, $st_c \in Client[c, s, r_c, r_s, PK(s), ch_s, ch_c, pm, *]$.

Now we want to prove that such st_c is unique. By Assumption 27, 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 27) to establish the uniqueness of the client strand.)

□

Lemma 40 Let $\mathcal{B}$ be a bundle in Σ, and $st \in Server[s, c, r_c, r_s, k_c, ch_s, ch_c, pm, Sms']$ a server strand such that $SK(s), SK(c) \notin \mathcal{K}_p$, $uncomp(ch_s)$ and $uncomp(ch_c)$. 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 37. 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 36, and that the server and the client agree on the value of pm by Lemma 39. □

Note the difference between the conditions required by the client's and server's secrecy guarantees (Lemmas 37 and 40, respectively). 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.

Our proof is modular in the sense that our analysis of the Record Layer replies only upon the results we have established for the Handshake Protocol (Lemmas 37–40) and the independence assumptions from Section 4.2. This shows that the Record Layer Protocol could be used with *any* handshake protocol that satisfies these results and assumptions (suitably adapted to the new protocol).

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 Application 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 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 41 Let $\mathcal{B}$ be a bundle in Σ, and st be a 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 42 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 assumption (Section 3.1.1), 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 Assumption 27, $pm_1 \neq pm_2$ since $st_1 \neq st_2$. Also

by assumption (Section 3.1.1), key generation 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 Application Layer is authenticated, i.e. has been sent earlier by the expected sender and was intended for that principal.

Lemma 43 Let $\mathcal{B}$ be a bundle in Σ, and $st_c \in Client[c, s, r_c, r_s, k_s, ch_s, ch_c, pm, Sms]$ a client strand such that $SK(s) \notin \mathcal{K}_P$ and $uncomp(ch_s)$. Let n be a node in st_c such that $msg(n) = -[t, i]_{se,sm}$ for $i > 0$. Then there exists a node n' in the unique corresponding server strand $st_s \in Server[s, c, r_c, r_s, k_s, ch_s, ch_c, pm, Sms']$ such that $msg(n') = +[t, i]_{se,sm}$.

Proof: By Lemma 38, the server strand $st_s \in Server[s, c, r_c, r_s, k_s, ch_s, ch_c, pm, *]$ 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 37 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 42. □

Lemma 44 Let $\mathcal{B}$ be a bundle in Σ, and $st_s \in Server[s, c, r_c, r_s, k_s, ch_s, ch_c, pm, Sms]$ a server strand such that $SK(c), SK(s) \notin \mathcal{K}_P$, $uncomp(ch_c)$ and $uncomp(ch_s)$. Let n be a node in st_s such that $msg(n) = -[t, i]_{ce,cm}$ for $i > 0$. Then there exists a node n' in the unique corresponding client strand $st_c \in Client[c, s, r_c, r_s, k_s, ch_s, ch_c, pm, Sms']$ such that $msg(n') = +[t, i]_{ce,cm}$.

Proof: The proof is similar to the previous lemma, and uses Lemmas 39, 40, and 42. □

We now prove that if a principal receives a stream of messages in the Application Layer, then the corresponding principal must have sent the same messages in the same order earlier. We write $\#Sms$ for the length of Sms.

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

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

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 43, there is a unique server strand $st_s \in \mathit{Server}[s, c, r_c, r_s, k_s, ch_s, ch_c, pm, Sms']$ 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 $\mathit{received}(Sms)$ must be in $\mathit{sent}(Sms')$, and t must have the same index i in both sequences. It follows that $\mathit{received}(Sms) \leq \mathit{sent}(Sms')$. □

Recall that we are considering a simplified version of the full TLS protocol. Essentially the same properties as in Theorem 1 hold for full TLS (obviously, the strands mentioned in the theorem have to be replaced with strands of the full protocol, and the "8" replaced by "11" to account for the messages we have dropped). It is easy to check that the proofs of the above theorem and of the contributing Lemmas 43 and 44 still hold for the full protocol: these proofs do not depend upon the messages that we have simplified (other than the fact that these messages are disjoint from the Record Layer messages).

Recall also that we are not considering session re-negotiation in this paper. If we were, the statement of the above theorem would need to be adapted. As noted in Section 2, a recent attack shows that there is no cryptographic binding between the sessions before and after a re-negotiation, so the messages received in those sessions might originate from different strands.

4.4.2 Secrecy

We now prove that the Record Layer provides secrecy for the Application Layer. Since Application Layer messages may contain terms that are known to the penetrator before starting the Application Layer exchange, such as identities, certificates, etc., the Record Layer cannot guarantee that the penetrator does not know any of the contents of the Application Layer messages. The secrecy provided by the Record Layer guarantees that the penetrator

cannot learn anything "new" from messages exchanged in the Application Layer between two regular strands.

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

1. For each client strand $st_c \in \mathit{Client}[c, s, r_c, r_s, k_s, ch_s, ch_c, pm, \mathit{Sms}]$ such that $\mathit{SK}(s) \notin \mathcal{K}_P$ and $\mathit{uncomp}(ch_s)$, there is no penetrator path that starts at a Record Layer node $n' \in st_c$ and includes a penetrator node n such that $\mathit{msg}(n) \sqsubseteq \mathit{appmsg}(n')$.
2. For each server strand $st_s \in \mathit{Server}[s, c, r_c, r_s, k_c, ch_s, ch_c, pm, \mathit{Sms}]$ such that $\mathit{SK}(c), \mathit{SK}(s) \notin \mathcal{K}_P$, $\mathit{uncomp}(ch_c)$ and $\mathit{uncomp}(ch_s)$, there is no penetrator path that starts at a Record Layer node $n' \in st_s$ and includes a penetrator node n such that $\mathit{msg}(n) \sqsubseteq \mathit{appmsg}(n')$.

The theorem talks about normal bundles. For non-normal bundles the result does not hold if the record node contains some term t already known to the penetrator: the penetrator can pair $\mathit{msg}(n')$ with t, and then split them again to obtain t. By restricting to normal bundles we prevent such paths, and show the penetrator cannot *extract* terms from $\mathit{msg}(n')$.

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 $\mathit{appmsg}(n') = t$. From Definition 5,

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

Since $\mathcal{B}$ is normal and $\mathit{msg}(n) \sqsubseteq t$, 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 37. □

As with Theorem 1, essentially the same theorem as Theorem 2 holds for full TLS: the proof maps back to the full protocol.

It is worth making clear what the above theorem does *not* say. First, the theorem does not show that the protocol provides so-called *strong secrecy* [Bla04, Aba97], where the penetrator should be able to detect no difference between different values of the Application Layer message.

We believe that strong secrecy does hold for TLS; note in particular that if an agent sends the same message twice within the same session (so encrypted with the same key), then they will use different sequence numbers, and so produce different ciphertexts, so the penetrator will not be able to detect that the same ciphertext was sent twice. Extending the Strand Spaces Model to verify strong secrecy properties seems like an interesting challenge, which we leave for future work.

Further, TLS does not guard against implicit flows of information caused by the functionality of the Application Layer protocol layered on top of TLS.

As a simple example, suppose a certain message in the Application Layer protocol may be either "abort" or "continue", and that subsequent messages are sent only if this message is "continue": then clearly the penetrator may deduce the contents of the message by observing the presence of subsequent messages.

Cortier et al. [CRZ06] carry out a systematic investigation of the relationship between the two properties, although in the context of probabilistic encryption (which is not used in TLS).

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 identity? 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 $st_c \in \mathit{Client}[c, s, r_c, r_s, k_s, ch_s, ch_c, pm, \mathit{Sms}]$ is a regular client strand such that $SK(s) \notin \mathcal{K}_P$ and $\mathit{uncomp}(ch_s)$. 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 $st_s \in \mathit{Server}[s, c, r_c, r_s, PK(c), ch_s, ch_c, pm, *]$, as in Theorem 1.
2. Suppose $st_s \in \mathit{Server}[s, c, r_c, r_s, k_c, ch_s, ch_c, pm, \mathit{Sms}]$ is a regular server strand such that $SK(c), SK(s) \notin \mathcal{K}_P$, $\mathit{uncomp}(ch_c)$ and $\mathit{uncomp}(ch_s)$. 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 $st_c \in \mathit{Client}[c, s, r_c, r_s, PK(s), ch_s, ch_c, pm, *]$, as in Theorem 1.

As with Theorem 2, this theorem restricts attention to normal bundles, to ignore paths that do not represent a causal link.

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 $msg(n)$: using precisely the same argument as in the proof of Theorem 2, the penetrator is unable to decrypt $msg(n)$. 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.

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

As with the earlier theorems, essentially the same theorem as above holds for full TLS: again, the proof maps back to the full protocol.

4.5 Interference freedom

We now consider another property provided by TLS. Informally, the property shows that TLS does not interfere with the Application Layer protocol; this means that when we analyse the Application Layer protocol we can abstract away the details of TLS and just consider the services it provides.

We define bundles to be *interference-free*, as follows. The condition says that no multi-layer attacks exist.

Definition 45 Let $\mathcal{B}$ be a bundle. $\mathcal{B}$ is *interference-free* iff for every penetrator path p in $\mathcal{B}$ that starts at a regular node n_1 and ends at a regular Record Layer node n_2:

1. n_1 is a Record Layer node such that $appmsg(n_1) = appmsg(n_2)$; or
2. n_1 is a Record Layer node, and p traverses two nodes n'_1 and n'_2 such that $n'_1 \preceq n'_2$, $msg(n'_1) = appmsg(n_1)$, and $msg(n'_2) = appmsg(n_2)$; or
3. p traverses either the key edge of an E strand used in record-layer construction of $msg(n_2)$, or the key edge of a D strand used in the record-layer unpacking of $msg(n_1)$.

Note that, in contrast to earlier results, this definition makes no assumptions about long-term keys not being compromised. Case 1 describes the normal case of messages being transmitted unchanged. Case 2 describes the case where the penetrator is able to extract the application message from $msg(n_1)$, and construct $msg(n_2)$: clearly he must know the relevant keys, either because he is using his own identity within TLS, or because of compromised keys. Case 3 describes the case where the penetrator uses data learnt from n_1 as keying material; again, this will be because he is using his own identity within TLS, or because of compromised keys. In the case where there are no compromised keys, this adds up to saying that the Application Layer messages that the penetrator sends using his own identity can

be produced from the messages that have been sent to him and his initial knowledge: this is effectively what equation (3) from Section 4.1 said.

Not every TLS bundle is interference-free. For example, the penetrator may learn handshake keying material (the client's nonce, the server's nonce or the premaster secret) and then replay it as part of the application layer payload. Alternatively, he may learn a nonce or premaster secret from a Record Layer message (sent to him, or corresponding to a strand for which he has compromised the long-term key), and replay it in the handshake. We define such paths as follows.

Definition 46 Let $\mathcal{B}$ be a normal bundle, a an atomic term, and p a path in $\mathcal{B}$ that starts at a regular node n_1 and ends at a regular node n_2.

- We say that p is an *application-to-handshake crossing path for* a if n_1 is a Record Layer node, $a \sqsubseteq \mathit{appmsg}(n_1)$, n_2 is a Handshake node, and a appears as handshake keying material in n_2.
- We say that p is an *handshake-to-application crossing path for* a if n_1 is a Handshake node, a appears as handshake keying material in n_1, n_2 is a Record Layer node, and $a \sqsubseteq \mathit{appmsg}(n_2)$.

Nevertheless, there is a sense in which these crossing paths are irrelevant: we will show (Lemma 49, below) that the penetrator could have produced similar behaviours but using new values for a within the nodes n_2. More precisely, we will show that for every TLS bundle $\mathcal{B}$, there is an interference-free bundle $\mathcal{B}'$ that captures the same Application Layer behaviour. Formally, $\mathcal{B}$ and $\mathcal{B}'$ will be abstractly equivalent:

Definition 47 Bundles $\mathcal{B}$ and $\mathcal{B}'$ are *abstractly equivalent* iff there is a bijection ϕ between the regular nodes of $\mathcal{B}$ and $\mathcal{B}'$, such that

1. $\mathit{appmsg}(n) = \mathit{appmsg}(\phi(n))$ for all Record Layer nodes $n \in \mathcal{B}$;
2. $n \Rightarrow^+ n'$ iff $\phi(n) \Rightarrow^+ \phi(n')$, for all $n, n' \in \mathcal{B}$;
3. ϕ maps $\mathit{Client}[c, s, *, *, *, *, *, *, *]$ strands to $\mathit{Client}[c, s, *, *, *, *, *, *, *]$ strands, and maps $\mathit{Server}[s, c, *, *, *, *, *, *, *]$ strands to $\mathit{Server}[s, c, *, *, *, *, *, *, *]$ strands.
4. For all atomic terms a, a originates in the application layer at n in $\mathcal{B}$ iff a originates in the application layer at $\phi(n)$ in $\mathcal{B}'$.

In [KL09] we introduced a technique for reasoning about layered security protocols that abstracts away from the implementation of the secure transport protocol and just models the security services it provides. In [KL10] we proved that this approach is sound under the assumption that for every

bundle, there is an abstractly equivalent interference-free bundle. Hence our aim here is to prove that TLS satisfies this property.

We need to make some additional assumptions about the Application Layer protocol: our earlier assumptions ensured that the Application Layer protocol did not interfere with TLS; these new assumptions ensure that TLS does not interfere with the Application Layer protocol.

Assumption 48 We assume the following:

1. The Application and Handshake protocols satisfy *disjoint encryption*: no encrypted or hashed subterm of a handshake message is a subterm of an Application Layer message, with the exception that we do allow Application Layer messages to contain public key certificates that satisfy Assumption 29.
2. The client and server nonces do not originate in the Application Layer (we already assumed the same for the premaster secret).
3. All identities and public keys are within the penetrator's initial knowledge $\mathcal{T}_p \cup \mathcal{K}_p$.
4. The Handshake Protocol is independent of the keying material used: formally, Σ is closed under consistent renaming of keying material and the keys generated from them.
5. For every valid public key certificate $\{PK(a)^\frown a\}_{sk}$ that appears within some Application Layer message, there is a corresponding CA strand $CA[ca, a]$ where $sk = SK(ca)$.

(Assumption 48 is "local" to the current section.)

We begin by considering crossing paths. We will show that we may always find an abstractly equivalent bundle without such paths.

Lemma 49 Let $\mathcal{B}$ be a normal bundle in Σ. Then there is an abstractly equivalent normal bundle $\mathcal{B}'$ such that no atomic term a occurs both within an Application Layer message and as handshake keying material.

Proof: For each atomic term a that occurs both within an Application Layer message and as handshake keying material, pick a new atomic term a^H. We create a new bundle $\mathcal{B}'$, informally by replacing all handshake keying uses of a by a^H. More precisely, whenever a occurs in a regular strand in a handshake keying position, we rename it to a^H. We replace all occurrences of each key generated using a by a key correspondingly generated using a^H (such keys will be new, by the assumptions about key generating functions). We leave a unchanged within Application Layer messages. These regular strands are

still valid strands because we have assumed that keying material is not passed from the handshake to the Application Layer protocol, and that Σ is closed under renaming of keying material.

If a originates in $\mathcal{B}$ as an Application Layer term, we add M or K strands to originate a^H; if a originates in $\mathcal{B}$ as handshake keying material (so the originating instance has been renamed to a^H), we add M or K strands to originate a. Note that the terms corresponding to these new strands do not originate on regular strands, so unique origination is preserved.

We adapt the penetrator strands of $\mathcal{B}$ to fit $\mathcal{B}'$. For non-crossing strands, we simply rename appropriate occurrences of a to a^H. Consider now crossing paths for a in $\mathcal{B}$. If a is the premaster secret then, by the disjoint encryption assumption, any application-to-handshake crossing path must encrypt a to create a ClientKeyExchange message, and any handshake-to-application crossing path must remove a from its encryption in the ClientKeyExchange message. Hence, whether a is the premaster secret or a nonce, a occurs as the message of some node in the path.

Pick a particular crossing path p for a. Since $\mathcal{B}$ is normal, p starts with a (possibly empty) destructive subpath and continues with a (possibly empty) constructive subpath. We perform a case analysis.

- Suppose p is an application-to-handshake crossing path. We rename a to a^H within the constructive subpath of p (we do not need the destructive subpath); if a originated as an application layer term in $\mathcal{B}$, we provide a^H from the M or K strand; if a originated as handshake keying material in $\mathcal{B}$, we provide a^H from the first handshake-to-application crossing path.
- Now suppose p is a handshake-to-application crossing path. We need only the constructive subpath of p; if a originated as handshake keying material in $\mathcal{B}$, we provide a from the M or K strand; if a originated as an Application Layer term in $\mathcal{B}$, we provide a^H from the first application-to-handshake crossing path.

The transformations above can be repeated for each path (duplicating strands if they are part of several paths). Let $\mathcal{B}'$ be the resulting bundle. Bundles $\mathcal{B}$ and $\mathcal{B}'$ are abstractly equivalent, since we have not changed the application layer behaviour of regular strands. Further, the transformations have preserved the property of being normal. □

Theorem 4 Let $\mathcal{B}$ be a bundle in Σ. Then $\mathcal{B}$ is abstractly equivalent to a normal interference-free bundle $\mathcal{B}'$.

Proof: Without loss of generality, assume $\mathcal{B}$ is normal (or else consider an equivalent normal bundle). We describe how to construct the corresponding bundle $\mathcal{B}'$. Begin by transforming the bundle, as in Lemma 49, so that no atomic term a occurs both within an Application Layer message and as handshake keying material.

Let p be a path that starts at a regular node n_1 on strand st_1, and ends at a regular Record Layer node n_2 on strand st_2. If p traverses the key edge of a record-layer E or D strand, then it satisfies case 3 of Definition 45, and we are done. Otherwise, since $\mathcal{B}$ is normal, p consists of a destructive subpath followed by a constructive subpath; let n be the "inflection" node, i.e. the intermediate node between the subpaths, so $msg(n1) \sqsupseteq msg(n) \sqsubseteq msg(n_2)$.

If $msg(n) = msg(n_2)$ then, since no Record Layer message is a proper subterm of another TLS message, we must also have $msg(n_1) = msg(n)$, and so case 1 of Definition 45 holds.

Otherwise, because of the form of Record Layer messages, the final step of p must be an E strand, encrypting some term $t1 = t^\frown HMAC(mk^\frown i^\frown t)$ with some key ek, where $t = appmsg(n_2)$. By assumption, p does not follow the key edge. Without loss of generality, we may assume that $t1$ is formed from a C strand; if not, we can add an S and a C strand to split and immediately re-combine $t1$. Let n_2' be the node that provides t to the C strand, so $msg(n_2') = appmsg(n_2)$. If p traverses the edge providing the HMAC term, then we can transform the bundle so that the penetrator creates this HMAC term using an H strand with input from n_2': note that the penetrator has produced ek, which he must have done using a KG strand; hence he can similarly produce mk using a KG strand. In each case, then, the transformed path includes n_2'.

Now consider the node n_1 where p starts. We perform a case analysis.

- Suppose n_1 is on a Handshake node. Consider what information p contributes from n_1 to n_2, i.e. the term of the inflection node.
  - It cannot contribute any handshake keying material, because of the earlier transformation corresponding to Lemma 49.
  - Further, it cannot contribute any encrypted term other than public key certificates, because of the disjoint encryption assumption.
  - Suppose p contributes identities of principals or public keys; recall that these were assumed to be in $\mathcal{T}_P \cup \mathcal{K}_P$; we can therefore transform p by removing the destructive subpath, and providing these terms from M and K strands.
  - Suppose p contributes one or more public key certificates of the form $\{pk^\frown a\}_{sk}$; recall (Assumption 29) that we assume $pk = PK(a)$ or $sk \in \mathcal{K}_P$; in the former case we can transform p to provide

the certificate from a CA strand; in the latter case, the penetrator can construct the certificate himself using an E strand with inputs from M and K strands.

- Otherwise n_1 is on a Record Layer node. Because of the disjoint encryption assumptions, p must start by extracting $appmsg(n_1)$. Let n_1' be the node where $msg(n_1') = appmsg(n_1)$. Hence case 2 of Definition 45 holds.

Finally note that the transformations above can be repeated for each path (duplicating strands if they are part of several paths). Let $\mathcal{B}'$ be the resulting bundle. Bundles $\mathcal{B}$ and $\mathcal{B}'$ are abstractly equivalent, since we have not changed the Application Layer behaviour of regular strands. Further, the transformations have preserved the property of being normal. □

As with the earlier theorems, a corresponding theorem as above holds for full TLS: essentially the same proofs for this theorem and the contributing Lemma 49 hold for the full protocol. If the Record Layer messages contain any terms of the types that we removed in the simplifications (e.g. cipher suites), then we have to assume that those terms are in $\mathcal{T}_P$, so they can be introduced in M strands, as we did with principals' identities and public keys.

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 fault-preserving simplifying transformations [HL01]. 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, session independence and interference freedom. 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 current research focuses on utilizing the rich framework of strand

spaces to model a wide variety of secure channels [KL09, KL10, Kam09]. 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.

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, such as unilateral TLS.

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 Application Layer have not been verified previously.

TLS has been analysed before using model checking techniques, for example in [MSS98, 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 Application Protocol to ensure this.

In [CK01] Canetti and Krawczyk consider key exchange protocols and their use for building secure channels. The authors present a template that describes how a key exchange protocol and an authenticated encryption scheme can be used as building blocks for a secure channel. They consider a key exchange protocol to be secure if, when the two parties involved in the exchange complete the protocol, they arrive at authenticated secret session keys. Then, they prove that, if these session keys are used in an Encrypt-then-Authenticate protocol, the resulting channel is a secure channel that provides both authentication and secrecy. This is very similar to the modular approach we adopted in analysing TLS. However, in TLS, the Record Layer Protocol applies the authentication MAC function then encrypts the

Application Layer messages, the opposite order to in Canetti and Krawczyk's setting. In addition, Canetti and Krawczyk do not discuss other security properties that can be provided by secure channels such as stream authentication and session independence. They also do not address the problem of multi-layer interaction.

More recently, Gajek et al. [GMP+08] perform a security analysis of TLS within the Universal Composable security framework [Can01]. Such an analysis appears to provide stronger guarantees about the protocol. However, we believe that a Dolev-Yao-style proof, like the one in the current paper, is better at identifying interactions between different parts of the protocol, and identifying requirements upon the Application Layer protocol in order for the two to not interfere.

Several researchers have considered composition of security protocols in a broader sense. As noted earlier, Guttman and Thayer [GT00b] show that two protocols are independent if they satisfy disjoint encryption, i.e. they share no encrypted component. Cortier et al. [CDD07] prove a similar result, but using distinct protocol tags for each protocol.

In [ACG+08], Andova et al. consider protocols that are composed sequentially, with data being passed from one to another. They show how the security properties of the two component protocols combine. They make an independence assumption about the two protocols, effectively that they satisfy disjoint encryption.

Datta et al. [DDMP05, DDMR07] describe a logic for stating and proving properties of protocols. The logic allows properties of sub-protocols to be combined. Non-interference between protocols is achieved by stating and verifying *invariants* that all sub-protocols must satisfy. The logic is used in modular proofs in, e.g., [MP04, DDMP05, HSD+05, RDD+06].

Acknowledgements

We would like to thank the anonymous referees, Chris Dilloway, Tom Gibson-Robinson and Sebastian Gajek for their many 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

[Aba97] Martín Abadi. Secrecy by typing in security protocols. In *Theoretical Aspects of Computer Software*, volume volume 1281 of LNCS, pages 611–638. Springer, 1997.

[ACG+08] S. Andova, C.J.F. Cremers, K. Gjøsteen, S. Mauw, S.F. Mjølsnes, and S. Radomirović. A framework for compositional verification of security protocols. *Information and Computation*, 206:425–459, February 2008.

[AR00] Martín Abadi and Philip Rogaway. Reconciling two views of cryptography (the computational soundess of formal encryption). In *Proc. 1st IFIP International Conference on Theoretical Computer Science (IFIP TCS)*, volume 1862 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2000.

[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)*, 2003.

[Bla04] Bruno Blanchet. Automatic proof of strong secrecy for security protocols. In *IEEE Symposium on Security and Privacy*, pages 86–100, 2004.

[Bla09] Bruno Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, 2009.

[BP05] M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. *IEEE Transactions on Dependable and Secure Computing*, 2(2):109–123, 2005.

[Can01] R. Canetti. Universally compsable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001.

[CDD07] Véronique Cortier, Jérémie Delaitre, and Stéphanie Delaune. Safely composing security protocols. In *Proceedings of the 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, volume 4855 of *Lecture Notes in Computer Science*, pages 352–363. Springer, 2007.

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

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, pages 453–474, London, UK, 2001. Springer-Verlag.

[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.

[CRZ06] Véronique Cortier, Michaël Rusinowitch, and Eugen Zălinescu. Relating two standard notions of secrecy. In *Proc. of the 20th Int. Conference Computer Science Logic (CSL'06)*, volume 4207 of *Lecture Notes in Computer Science*, pages 303–318. Springer, 2006.

[CW05] Véronique Cortier and Bogdan Warinschi. Computationally sound, automated proofs for security protocols. In *Proc. 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005.

[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.

[DDMP05] A. Datta, A. Derek, J.C. Mitchell, and D. Pavlovic. A derivation system and compositional logic for security protocols. *Journal of Computer Security*, 13(3):423–482, 2005.

[DDMR07] A. Datta, A. Derek, J.C. Mitchell, and A. Roy. Protocol composition logic. *Electronic Notes in Theoretical Computer Science*, 172:311–358, 2007.

[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.

[Ext09] Authentication gap in TLS renegotiation. Extended Subset, `http://extendedsubset.com/?p=8`, 2009.

[GMP+08] Sebastian Gajek, Mark Manulis, Olivier Pereira, Ahmad-Reza Sadeghi, and Jörg Schwenk. Universally composable security analysis of TLS. In *Proc. 2nd Provable Security International Conference (ProvSec'08)*, volume 5324 of *LNCS*. Springer, 2008.

[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.

[GT02] Joshua D. Guttman and F. Javier Thayer. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 283(2):333–380, 2002.

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

[Hea02] J. Heather. Strand spaces and rank functions: More than distant cousins. In *Proceedings of Computer Security Foundations Workshop (CSFW 2002)*, pages 104–116. IEEE Computer Society, 2002.

[HL01] Mei Lin Hui and Gavin Lowe. Fault-preserving simplifying transformations for security protocols. *Journal of Computer Security*, 9(1, 2):3–46, 2001.

[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.

[Kam09] Allaa Kamil. *The Modelling and Analysis of Layered Security Architectures in Strand Spaces*. DPhil, Oxford University, 2009.

[KL09] Allaa Kamil and Gavin Lowe. Specifying and modelling secure channels in strand spaces. In Piepaolo Degano and Joshua Guttman, editors, *Formal Aspects in Security and Trust (FAST 2009)*, volume 5983 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2009.

[KL10] Allaa Kamil and Gavin Lowe. Understanding abstractions of secure channels. In Piepaolo Degano, Sandro Etalle, and Joshua Guttman, editors, *Formal Aspects of Security and Trust (FAST 2010)*, volume 6561 of *Lecture Notes in Computer Science*, pages 50–64. Springer, 2010.

[LMMS98] P. Lincoln, J.C. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic polynomial-time framework for protocol analysis. In *Proc. 5th ACM Comference on Computer and Communications Security (CCS)*, pages 112–121, 1998.

[Low96] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In Tiziana Margaria and Bernhard Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.

[MP04] Catherine Meadows and Dusko Pavlovic. Deriving, attacking and defending the GDOI protocol. In Peter Ryan, Pierangela Samarati, Dieter Gollmann, and Refik Molva, editors, *Proceedings of ESORICS 2004*, volume 3193 of *Lecture Notes in Computer Science*, pages 53–72. Springer Verlag, 2004.

[MSS98] J.C. Mitchell, V. Shmatikov, and U. Stern. Finite-state analysis of SSL 3.0. In *Proceedings of the 7th USENIX Security Symposium*, 1998.

[MvOV97] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[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.

[RDD+06] A. Roy, A. Datta, A. Derek, J. C. Mitchell, and J.-P. Seifert. Secrecy analysis in protocol composition logic. In *Proceedings of 11th Annual Asian Computing Science Conference*, 2006.

[RRDO10] E. Rescorla, M. Ray, S. Dispensa, and N. Oskov. Transport Layer Security (TLS) renegotiation indication extension. Internet Engineering Task Force (IETF) Request for Comments: 5746, `http://tools.ietf.org/html/rfc5746`, 2010.

[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)*. IEEE Computer Society, 1999.

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

[WS96] David Wagner and Bruce Schneier. Analysis of the SSL 3.0 protocol. In *Proceedings of the Second USENIX Workshop on Electronic Commerce*, 1996.

[YC05] Alec Yasinsac and Justin Childs. Formal analysis of modern security protocols. *Information Sciences*, 171(1-3):189 – 211, 2005.