TLS ANALYSIS USING CADP

ALBERTO CALIXTO AND RAÚL MONROY

Abstract. This paper reports on an analysis of the Transport Layer Security protocol [DA99], TLS. We have used LOTOS [BB88] to model TLS, the μ-calculus [Koz83] to express security properties that we wish it to meet and CADP [FGK+96] to conduct formal analysis. Both the specification and the analysis closely follow Steve Schneider's [Sch96a] approach to the verification of security protocols. We show how we adapted Schneider's approach and applied it to the analysis of TLS, using different formalisms and proof methods. We report on the security properties that TLS is proved to enjoy.

1. Introduction

This paper reports on an analysis of the Transport Layer Security protocol, TLS [DA99]. TLS builds on the specification of Secure Socket Layer Protocol [AFK96], SSL, an authentication protocol for client/server applications, published by Netscape. TLS is used widely and so the more security properties is proved to enjoy the sooner it will become established.

We have used LOTOS [BB88] to model TLS, the μ-calculus [Koz83] to express security properties that we wish it to meet and CADP [FGK+96] to conduct TLS analysis. LOTOS, a Language of Temporal Specifications, is a process algebra suitable for modelling and analysing communicating systems. It is well-established in both industry and academia. LOTOS unites a number of interesting issues borrowed from leading process algebras. CADP is an integrated toolbox that offers a wide range of methods for modelling, analysing and simulating LOTOS specifications.

Both the specification and the analysis closely follow Steve Schneider's approach to the verification of security protocols [Sch96a]. Roughly, the verification problem is split into three steps. First, we use LOTOS to model the behavioural aspects of three principal entities: i) a communication sub-network, ii) a collection of principals involved in the authentication process and iii) a

We are grateful to the anonymous referees for providing invaluable, useful suggestions and advice on an earlier draft of this paper. This research was supported by CONACYT grants SEP-REDII and 33337-A.

spy. The communication sub-network abstracts out physical components and is assumed to provide an error-free, message delivery service. As expected, the principals reify the message exchange indicated by the protocol under analysis. The spy is defined as powerful as possible and yet he is not supernatural. Then all these processes are put together using parallel composition; giving them capabilities of interaction. Second, we brainstorm security properties and then we express them using a suitable formalism. Finally, in the third step, the specification is tested against the properties, collecting statistics.

1.1. Paper Overview

In the sequel we show the results of our experimentations. First, though, we provide an abstract, informal description of TLS (see §2.) Next, we outline Schneider's approach to the verification of authentication protocols (see §3.) Then, we illustrate how we have adopted Schneider's theoretical model and apply it to the verification of TLS (see §4.) Finally, following a discussion of the experimental results obtained throughout our investigations, which appears in §4.2, we summarise the lessons drawn from our work (see §6.)

2. TLS Specification: an Overview

TLS aims at providing both privacy and integrity of the information exchanged by a client/server application, running over the Internet. Rather than a single, monolithic protocol, TLS is a protocol schema: Users may set various security parameters so as to suit a set of security requirements. TLS is both versatile and scalable. However, it is also hard to implement, let alone analyse. A full analysis of TLS would require one to consider an unwieldy number of combination of methods. Fortunately, the modular structure of TLS helps conducting, at least partially, the analysis in terms of its components.

TLS consists of two protocols: i) *record* and ii) *handshake*. Record is low-level, working on top of a transport layer protocol, such as TCP. Record controls end-to-end communication. It splits messages into blocks of fixed, manageable size, encoding them, and optionally compressing them, previous transmission. Upcoming messages are, in turn, decoded, optionally decompressed, and then assembled to their original form before being passed up on to higher entities. So, record aims to encapsulate higher-level protocols, including handshake.

Handshake aims to set up and terminate a session that allows secure communication amongst two parties: a client, who requested the session, and a server. Setting up a session amounts to fixing 6 parameters: the session identifier, the

peers' certificate, the compression method to be used, the cipher algorithm, whether or not multiple connections are permitted and a disposable master secret. The first 5 parameters are all low-level: They vary the run of the protocol but do not add to fulfilling a security requirement. Handshake is a procedure whereby the client and the server authenticate each other. Upon success both client and server end up in possession of the master secret, which is used as a disposable encryption key. As long as the master secret is uncompromised, the connection will remain secure. This paper reports on our experiments in the analysis of Handshake using the CADP toolbox [FGK$^+$96].

In the rest of this section, we discuss the abstract specification of handshake used in our analysis. Our specification is only a subset of TLS. It omits various possibilities, e.g. some optional messages are here mandatory. However, we believe it representative as it aims to achieve the strongest TLS's security goals. Table 1 illustrates a typical protocol run. Messages are sent in the order of appearance. The notation *Label* $A \rightarrow B : M$ is used to convey an interaction, called *Label*, in which A sends message M to B, which B receives.

ClientHello	*Client* $\rightarrow$ *Server* :	$C, Nc, Sidc, Pc$
ServerHello	*Server* $\rightarrow$ *Client* :	$S, Ns, Sids, Ps$
ServerCertificate	*Server* $\rightarrow$ *Client* :	$Cert(S, Ks^+)$
ClientCertificate	*Client* $\rightarrow$ *Server* :	$Cert(C, Kc^+)$
ClientKeyExchange	*Client* $\rightarrow$ *Server* :	PMS_{Ks^+}
CertificateVerify	*Client* $\rightarrow$ *Server* :	$\{Hash\{Nc, Ns, PMS\}\}_{Kc^-}$
ClientFinished	*Client* $\rightarrow$ *Server* :	$\{Hash\{M,$ "*ClientFinished*"$, P\}\}_M$
ServerFinished	*Server* $\rightarrow$ *Client* :	$\{Hash\{M,$ "*ServerFinished*"$, P\}\}_M$
where $M = PRF(Nc, Ns, PMS)$ and		
where $P = \{PMS, C, Nc, Sidc, Pc, S, Ns, Sids, Ps\}$		

Table 1: The protocol handshake

At the beginning, the client, C, contacts the server, S: *ClientHello*. It supplies a session identifier, *Sidc*, a nonce, Nc, and secondary information, such as cipher algorithm and compression method, *Pc*. In response, S sends her nonce, *ServerHello*. Afterwards, both C and S exchange their public-key certificates, indicated by *ServerCertificate* and *ClientCertificate*.[1] The notation $\{\!|X|\!\}_K$ denotes the message encrypted or signed using the key K. K_A^+ and K_A^- respectively stand for agent A's public and private key.

[1] TLS specifies that the server's public key is delivered in an X.509v3 certificate [CCI88] , so as [Pau99], we assume that a spy cannot impersonate the server.

Next, C generates a fresh, 48-byte random string, called a *pre-master-secret*, and sends it to S, c.f. *ClientKeyExchange*. It later sends an authentication message, *CertificateVerify*, which encrypts the hash of some session parameters. Now, both server and client compute the *master-secret*, M, applying a secure pseudo-random number function, PRF, to the two nonces and the pre-master-secret. Both client and server will use the master secret as a symmetric key. To terminate the authentication process, they exchange a concluding message, which encrypts the hash of the parameters of the session, together with a string. The content of the string will be set according to the origin of the sender, c.f. *ClientFinished* and *ServerFinished*.

With this we complete the abstract description of TLS. We shall now discuss Schneider's approach to the verification of security protocols [Sch96a, Sch96b].

3. Proving Properties of Security Protocol: the Schneider Approach

Steve Schneider approaches the study of security protocols in three steps. In the first step, a formal specification of the computer network under analysis should be provided. The network includes both the principals negotiating a secure session and a spy. They all communicate through a medium, which encapsulates lower-level network component, e.g. routers, firewall, physical channels and so on. The spy may incur in all sorts of imaginable computer crimes, such as tampering or impersonation. The principals are modelled so as to follow the protocol subject to verification. The entire network is specified using Hoare's process algebra: *Communicating Sequential Processes*, CSP [Hoa85]. This step outputs a formal model of the protocol, called a process, which precisely describes the actions of the system, as well as of its subcomponents.

In the second step, the semantic of the process algebra is used to express each abstract security requirement. Often, process properties can be expressed either as a equation, using a suitable notion of process equivalence, or as a logical formula, written in a suitable modal-temporal logic. Schneider's approach makes use of the trace semantics to processes, which induces both a useful notion of process equivalence and can express properties in terms of execution traces.

In the third step model checking of the LOTOS specification takes place. Often, process algebras can be mechanised with a high level of automation. Schneider has used FDR [Ros94] to conduct his experimentations. In what follows, we discuss the first two steps further.

3.1. Modelling an Authentication Protocol

Modelling an authentication protocol amounts to modelling a computer network involving computer nodes and a computer communication network. A computer node may be either a server or a principal, both involved in the authentication process. Also it may be an spy, who endangers any attempt at establishing a secure session. The entire network is modelled as a compound process, which collects together, via parallel composition, the communication network and each computer node.

Computer nodes communicate one another only through the communication network. They all run concurrently. Here is the associated CSP model:

$$\mathit{Node}_0 \;|||\; \mathit{Node}_1 \;|||\; \ldots \;|||\; \mathit{Node}_{n-1} \;|||\; \mathit{Node}_n$$

where $P \;|||\; Q$ denotes a process in which P and Q proceed independently, without any possibility of interaction.[2] Each node is modelled independently, according to both the protocol and the role it plays, e.g. a server, a principal or a spy. By default, the spy is Node_0. Except for the spy, each node has two channels, one is used for input, rec, and the other for output, $trans$. Using these channels, each node, Node_i, $i \in \{1, \ldots, n\}$, interacts with the communication network, *Medium*:

$$\left(|||_{\,i \in \{1,\ldots,n\}} \mathit{Node}_i\right) \;\left|\{trans, rec\}\right|\; \mathit{Medium}$$

Let K be a set of channels, then $P \;\left|\{K\}\right|\; Q$ denotes an agent where P and Q may proceed independently, but may also interact one another via some channel in K. $|||$ is a special case of $\left|\{K\}\right|$, when K is empty. Except for the spy, a node reads only messages addressed to itself.

To other nodes, the spy is an ordinary node. However, the spy is capable of altering the conditions of the communication network. He may intercept, read and send (fake) messages. The spy is given only one extra channel, out, with which he interacts with the environment in order to display its knowledge.[3] Here is the model of the spy:

$$\begin{aligned}
\mathit{Spy}(S) \;=\; & rec?j?i?m \rightarrow \mathit{Spy}(S) \\
\Box\; & rec?j?i?m \rightarrow \mathit{Spy}(S \cup \{m\}) \\
\Box\; & \Box_{I \cup S \vdash m} trans!i!j!m \rightarrow \mathit{Spy}(S) \\
\Box\; & \Box_{I \cup S \vdash m} out!m \rightarrow \mathit{Spy}(S)
\end{aligned}$$

[2] Henceforth, we shall use $|||_{\,i \in I} P_i$ for the composition of all processes P_i, $i \in I$.

[3] Steve Schneider gives the spy three additional channels to enable elimination, interception and forgery. We think this design decision unnecessary and a source of confusion. So our model gives the spy the same channels as though he were an ordinary node.

Where we have the following observations:

- $\alpha \rightarrow P$ denotes a process that can do α and then become P. α is an *event*, involving a channel, $\{a, b, c, \ldots\}$, input, $\{?a, ?b, ?c, \ldots\}$, and output parameters, $\{!a, !b, !c, \ldots\}$. If the parameter is input, the string following ? is a variable; otherwise, the string is a concrete expression. In $rec?i?j?m$, $i, j \in \{0, \ldots, n\}$ and m is any message that is valid with respect to the protocol under analysis.
- $P \Box Q$ disjoins the capabilities of P and Q; as soon as one performs an action, the other will be dismissed.
- I represents the spy's knowledge before a protocol run; e.g. long-term keys of compromised agents.
- *Spy* offers four options:
  1. message interception, $rec?j?i?m \rightarrow \mathit{Spy}(S)$,
  2. message copy, $rec?j?i?m \rightarrow \mathit{Spy}(S \cup \{m\})$,
  3. message forgery,

     $$\Box_{I \cup S \vdash m} trans!i!j!m \rightarrow \mathit{Spy}(S)$$

     and
  4. content display, in which, upon request, it displays the knowledge it has gain from a protocol run, $\Box_{I \cup S \vdash m} out!m \rightarrow \mathit{Spy}(S)$.

$\vdash$ models what the spy may infer given both his knowledge and the network traffic [Ros86, Ros94]. Given that messages follow either of various specific formats, the spy may forge a message that can be understood by ordinary nodes.

Medium essentially provides a passive service. It gives computer nodes the ability to manipulate it in some way, hence making it easy to model and analyse both protocols and attacks. *Medium* is defined as follows:

$$\mathit{Medium} \stackrel{\text{def}}{=} M(\emptyset)$$

where

$$\begin{array}{rcl} M(B) & = & \Box_i trans?i?j?m \rightarrow \mathit{M}(B \cup \{i.j.m\}) \\ & \Box & \Box_{i.j.x \in B} rec!j!i!m \rightarrow \mathit{M}(B - \{i.j.m\}) \end{array}$$

Here is the final model of the computer network, *Net*, at starting point:

$$\mathit{Net} = \left(\Vert_{i \in \{1, \ldots, n\}} \mathit{Node}_i \right) \left|\{L\}\right| \mathit{M}(\emptyset) \left|\{L\}\right| \mathit{Spy}(\emptyset)$$

where $L = \{trans, rec\}$. Now we discuss how to model interesting properties of authentication protocols.

3.2. Modelling Properties of Authentication Protocols

In CSP, a process is given meaning using a set containing all possible traces associated with process execution, including the empty trace, *traces*(P):

$$\begin{aligned} \textit{traces}(Spy) \quad = \quad & \{\langle\rangle, \langle trans!i!j!m\rangle, \\ & \langle trans!i!j!m, rec!j!i!m\rangle, \ldots\} \end{aligned}$$

So a property is given as a predicate over traces. Process P enjoys property S iff all its traces do, in symbols:

$$P \textbf{ sat } S \quad \Leftrightarrow \quad \forall tr \in \textit{traces}(P).\ S$$

Often, it is useful to specify properties of traces with respect to a given set of events. Let K be a set of events, then the *projection* of trace tr on K, $tr \upharpoonright K$, is defined as the maximum sub-trace of tr all of whose events appear in K.

Using the semantics of process, we can cast security properties as properties over traces. An authentication protocol is said to provide *confidentiality* for a set of messages M iff any message of M that is circulating over the network will be received only by the intended receiver. This is given by:

$$\textit{Net} \textbf{ sat } \forall m \in M.\ S(m)$$

where $S(m)$ is an abbreviation for:

$$tr \upharpoonright trans.0.?u.m \neq \langle\rangle \Rightarrow tr \upharpoonright rec.?u.0.m \neq \langle\rangle$$

Equivalently, $S(m)$ may be expressed as an equation, as follows:

$$\textit{Net} \left|\{rec.?u.0.m\}\right| \textit{Stop} = \textit{Net} \left|\{\begin{smallmatrix} trans.0.?u.m \\ rec.?u.0.m \end{smallmatrix}\}\right| \textit{Stop}$$

where $P = Q$ only if *traces*(P) = *traces*(Q), and where *Stop* denotes the deadlock process, capable of executing no actions ever.

Authentication is modelled in terms of two sets of events, T and R. T authenticates R with respect to trace tr iff any one time tr contains an event $e_1 \in T$, then e_1 ought to be preceded by some event $e_2 \in R$, in symbols:

$$P \textbf{ sat } (T \textbf{ auth } R) \ \Leftrightarrow \ P \left|\{R\}\right| \textit{Stop} \textbf{ sat } tr \upharpoonright T = \langle\rangle$$

With this, we complete our revision of Schneider's approach to the verification of security protocols. We are now ready to present how we have adopted his approach to suit CADP and the results of our experimentations.

4. A CADP Analysis of TLS Using Schneider's Approach

Our work follows Schneider's approach closely, the main differences being that we use LOTOS [BB88] and work within the CADP (CAESAR/ALDÉBARAN)

toolbox [FGK+96]. LOTOS is the process algebra of the International Standard Organisation (ISO). It inherits theoretical issues from CSP, CCS [Mil89] and ACP [BK85]. So a CSP specification can be translated into a LOTOS specification with little effort. LOTOS specifications may be value-passing, as long as the associated value domains are all finite. They may manipulate complex data structures, described using the well-known theory of algebraic abstract data types [Gut77].

CADP is an integrated toolbox for specifying, analysing and simulating communicating systems. It was specially designed to deal with large case studies. CADP comprehends a wide range of verification methods and supports communication with other lower-level formalisms. Throughout our experiments, we use 5 CADP tools:

1. CAESAR, to translate a LOTOS specification into a labelled transition system, portraying its entire behaviour;
2. CAESAR.ADT, to help CAESAR dealing with abstract data types;
3. ALDÉBARAN, to prove process equivalence under various notions;
4. VERIFICATOR, to test the validity of a μ-calculus formula; and
5. OCIS, to simulate protocol execution.

4.1. The TLS model, a Partial Transcript

Our specification differs from Schneider's theoretical model, c.f. §3, only in three aspects: i) parameters range over finite domains, ii) recursive computation is disabled, so the protocol can be run only once and iii) the number of connection resumptions is fixed in advanced. These issues should not be taken as restrictions: They are all standard, necessary to guarantee the model is amenable to mechanical analysis, yielding a finite state system.

The full specification of TLS, the analysis we have conducted on it and the outcome of such analysis are electronically available in the following URL (in Spanish):

`http://research.cem.itesm.mx/raulm/pub/code/tls/`

Here we shall just describe important aspects of both the specifications and the experiments. To begin with, Table 2 shows the domains associated with the abstract data types used in our model. Except for *MS*, *Certificate*, *Hash* and *Crypted*, they all have straightforward interpretation. Here we shall look only into *MS*. *MS* contains elements of the form $PRF(n_0, n_1, p)$, where n_0 and n_2 are both nonces and p is a pre-master secret. Notice, however, that *PRF* on its own is not part of *MS*.

Type	Domain
User	$\{0, 1, 2, \ldots, n\}$
Session	$\{0, 1, 2, \ldots, n\}$
Parameters	$\{P_0, P_1, \ldots, P_n\}$
Nonce	$\{N_0, N_1, \ldots, N_n\}$
PMS	$\{PMS_0, PMS_1, \ldots, PMS_n\}$
PublicKey	$\{pubkeyserver, pubkeyclient, pubkeyspy\}$
PrivateKey	$\{privkeyserver, privkeyclient, privkeyspy\}$
Message	$\{ClientHello, ServerHello, \ldots, ClientFinished, \ldots\}$
MS	$\{PRF(n_0, n_1, m) \mid n_0, n_1 \in$ *Nonce* and $p \in$ *PMS*$\}$
Certificate	$\{Cert(i, k) \mid x \in$ *User* and $k \in$ *PublicKey*$\}$
Hashed	$\{Hash(n_0, n_1, i, p) \mid n_0, n_1 \in$ *Nonce*, $i \in$ *User* and $p \in$ *PMS*$\}$ $\cup \{Hash(M, m, \ldots) \mid M \in$ *MS*, $m \in$ *Message*, $\ldots\}$
Crypted	$\{Crypt(k, P) \mid k \in$ *PublicKey* and $P \in$ *PMS*$\}$ $\cup \{Crypt(k, m) \mid k \in$ *PrivateKey* and $m \in$ *Hashed*$\}$ $\cup \{Crypt(p, m) \mid m \in$ *MS* and $m \in$ *Hashed*$\}$

Table 2: Type Domains

Medium is as described in §3 except that the set of messages it holds ought to be finite:

$$Medium = M(n, 0, \emptyset)$$

where

$$\begin{aligned} &M(n, m, B) = \\ &\quad \textbf{if } n < m \textbf{ then} \\ &\qquad \Box_i \; trans?i?j?x \rightarrow M(n{+}1, m, B \cup \{i.j.x\}) \\ &\Box \textbf{ if } n > 0 \textbf{ then} \\ &\qquad \Box_{i.j.x \in B} \; rec!j!i!x \rightarrow \boldsymbol{M}(n{-}1, m, B{-}\{i.j.x\}) \end{aligned}$$

Notice that, unlike Schneider, we give no special treatment to the spy. Also notice that while *Medium* may hold up to m messages, its state, n, determines its entire capabilities of interaction. Given that it has no fixed service policy, *Medium* gives rise to an extraordinary state explosion, especially when m is set at a large value.

The models associated with both the client and the server are as expected, following the message communication indicated by TLS. The model of the client neatly distinguishes two phases: i) the initial, standard procedure to establish a session and ii) the procedure to recommence a session. The first phase

involves both the exchange of messages and the verification of protocol termination. The client interacts with the system through three channels, *trans*, *rec* and *endc*. This latter channel is used only for testing purposes indicating termination of a protocol run. The server is as the client except that the channel used to indicate termination is named *ends*. Our model gives the server the ability to handle multiple connections. So we can conduct protocol analysis assuming two or more concurrent sessions.

Unlike the client and the server, the spy has no fixed model. We change the spy model according to the property under verification. The root behind this model decision is that a non sensible model of behaviour may yield a large number of states that need not be analysed, for they provide no useful information. Changing the behaviour of the spy, the CADP framework can be thought of as being a workbench in which we conduct under control analysis. Here is the standard model of the spy:

$$
\begin{array}{rcl}
\mathit{Spy}(S) & = & rec?i?j?m \rightarrow \mathit{Spy}(S, m) \\
\mathit{Spy}(S, m) & = & \mathit{Spy}(S \cup \{m\}) \\
 & \Box & \Box_{I \cup S \cup \{m\} \vdash x} trans!j!i!x \rightarrow \mathit{Spy}(S \cup \{m\}) \\
 & \Box & \Box_{I \cup S \cup \{m\} \vdash x} out!x \rightarrow \mathit{Spy}(S \cup \{m\})
\end{array}
$$

Notice that unlike Schneider we disable the possibility for message elimination, as it unnecessarily increases proof workload. Message interception amounts to message loss, which is handled directly by lower-level protocols, c.f. the protocol record. Naturally, message loss causes the entire system to deadlock and so we omit it as it does not add to security hazards. To avoid medium flooding, as may occur in Schneider's theoretical model, the spy cannot introduce any kind of message into the medium at will. Instead, the spy is assumed to manipulate the messages that go around the medium only.

The deduction relation, $\vdash$, is pretty similar to that used by Schneider except that we explicitly indicate that, according to TLS, the Spy is assumed not to forge valid certificates. Table 3 conveys the definition of the $\vdash$ relation. There m is any message, k is a symmetric key, k^- a public key and k^+ the corresponding private key. By contrast, k^* denotes a key of any kind. As before, $\mathit{Crypt}(k, m)$ denotes the encryption of message m using key k, while $\mathit{Hash}(m)$ denotes the hashed message m.

Having discussed the main aspects of our implementation of TLS, we discuss the conducted analysis.

Axiom	Definition
A_1	$m \in B \ \Rightarrow \ B \vdash m$
A_2	$B \vdash m \ \wedge \ B \vdash n \ \Rightarrow \ B \vdash m.n$
A_3	$B \vdash m \ \Rightarrow \ B \vdash \mathit{Hash}(m)$
C_1	$\vdash \mathit{Cert}(U, K_u^+) \ \Rightarrow \ B \vdash U \ \wedge \ B \vdash k^+$
K_1	$\vdash m \ \wedge \ B \vdash k^* \ \Rightarrow \ B \vdash \mathit{Cryp}(k, m)$
K_2	$\vdash \mathit{Crypt}(k^-, m) \ \wedge \ B \vdash k^+ \ \Rightarrow \ B \vdash m$
K_3	$B \vdash \mathit{Crypt}(k^+, m) \ \wedge \ B \vdash k^- \ \Rightarrow \ B \vdash m$
K_4	$B \vdash \mathit{Crypt}(k, m) \ \wedge \ B \vdash k \ \Rightarrow \ B \vdash m$

Table 3: Spy's message deduction system

4.2. Analysis and Experimental Results

While versatile, Schneider's approach to protocol verification quickly gives rise to a state explosion. While in principle one could try to analyse the weakest protocol configuration, both hardware and software impose serious constraints on the number of states that can be analysed. CADP, for example, cannot handle more than 4×10^6 states. Within our context, we found out that the larger configuration supported by CADP is given by $\langle 6, 2, 2, 2 \rangle$, a vector conveying the maximum number of nonces, encryption parameters, pre-master secrets and session resumptions, respectively. Increasing either of these figures will cause system overflow. Translating the protocol description into the associated labelled transition system takes about 36-48 Hs. Getting rid of the spy, the model can be translated in about 100 seconds. The test was run on a 450MHz Ultra 60, a dual processor Ultra SPARC machine with 512MB of RAM. CADP

conf$(out!px)$, where $px \in$ *parameters*	false	1 min
conf$(out!nx)$, where $nx \in$ *nonce*	false	1 min
conf$(out!sx)$, where $sx \in$ *session*	false	1 min
conf$(out!pubkey)$, where $pubkey \in$ *publickey*	false	2 min
conf$(out!k)$, where $k \in$ *privatekey*	true	2 min
conf$(out!px)$, where $px \in$ *pms*	true	3 min
conf$(out!msx)$, where $msx \in ms$	true	3 min

Table 4: results about confidentiality, *i*=$\emptyset$

does not provide the trace semantics of FDR. However, it comes with a number

Conf$(out!Px)$, where $Px \in$ *Parameters*	false	1 min
Conf$(out!N)$, where $N \in$ *Nonce*	false	1 min
Conf$(out!Sx)$, where $Sx \in$ *Session*	false	1 min
Conf$(out!K)$, where $K \in$ *Publickey*	false	2 min
Conf$(out!K)$, where $K \in$ *PrivateKey-I*	true	2 min
Conf$(out!Px)$, where $Px \in PMS$	false	3.5 min
Conf$(out!MSx)$, where $MSx \in MS$	false	4 min

Table 5: Results about confidentiality, *I*=$\{privkeyserver\}$

of expressive, powerful logics, including the μ-calculus. We model confidentiality, c.f. §3, as follows:

$$\textit{Conf}(\alpha) = \nu X.\ ([-]X \wedge [\alpha]\mathtt{ff})$$

Roughly, this formula means that at the current state α cannot happen, $[\alpha]\mathtt{ff}$, and that this property will hold in the next state following any possible trace of execution, $[-]X$. Put another way, *Conf*(α) holds only if α never happens. Clearly, for *Conf*(α) to be faithful to Schneider's idea of confidentiality, α must portray the disclosure of a message that one wishes the spy will not know about. Tables 4 and 5 show some example test formula, involving confidentiality.

Auth(m_1, m_2), the formula scheme used to check authentication properties, is given by:

$$\nu X.([-]X \wedge [m_2]\mathtt{ff} \wedge [m_1](\mu Y.([-m_2]Y \vee \langle m_2\rangle \mathtt{tt})))$$

Roughly, m_2 may not occur unless m_1 does. This formula is faithful to Schneider's interpretation of authentication only if $m_1 \in T$ and $m_2 \in R$. Table 6 shows some example test formula involving authentication. In the information displayed in Table 6, α_i, $i \in \{1, \ldots, 5\}$, appears in Table 7.

Auth(α_1, α_0)	true	1 min
Auth(α_0, α_1)	false	1 min
Auth(α_2, α_1)	true	1 min
Auth(α_2, α_5)	false	1 min
Auth(α_4, α_3)	true	1 min
Auth(α_5, α_0)	false	1 min
Auth(α_5, α_4)	true	1 min
Auth(α_4, α_5)	false	1 min

Table 6: Authentication results

	datum
α_0	$trans\,!C\,!S\,!ClientHello\,!0\,!P_0\,!n_0$
α_1	$rec\,!S\,!C\,!ClientHello\,!0\,!P_0\,!N_0$
α_2	$rec\,!S\,!C\,!ClientCertificate\,!Cert(C,k_c^+)$
α_3	$trans\,!C\,!S\,!ClientKeyExchange!Cript(k_s^+,PMS_0)$
α_4	$trans\,!C\,!S\,!ClientFinished\,\ldots$
α_5	$rec\,!C\,!S\,!ClientFinished\,\ldots$

Table 7: Authentication messages

Here is the model of the possibility property stating that some traces of execution reach the end of a protocol run:

$$\mathit{End}(\alpha) = \mu X.\ (\langle - \rangle X \vee \langle \alpha \rangle \mathtt{tt}$$

Aiming at testing this possibility property, we introduce in our model the channels $ends$ and $endc$, see §4. Thus, here, $\alpha \in \{ends, endc\}$.

In both cases, $I = \emptyset$ and $I = \{privkeyserver\}$—see Tables 4 and 5—, we found that:

$$\mathit{End}(ends) = \mathit{End}(endc) = \text{true}$$

Our experimentations show that TLS is secure, provided both that the certificates are valid, and that the private keys of server and client are confidential.

5. Related Work

Dietrich has analysed SSL 3.0 using *Non-monotonic Cryptographic Protocols*, a belief logic [Die97]. Dietrich does not consider re-negotiations. By contrast, in our formalisation, session identifiers are recorded; once issued, a session identifier will prevail throughout the entire analysis. Dietrich's work involves long, detailed manual proofs, whereas ours relies upon fully automatic model checking techniques.

Also Wagner and Schneier have analysed SSL 3.0 [WS96]. Their analysis concentrates on the associated encryption system oriented, which our work omits, trying to break it down through repeated re-negotiations. They report, for example, on an attack to the key exchange mechanism under the context of Diffie-Hellman. They report other attack involving the elimination of the *Change Cipher Spec* message, which our model implicitly adds.

Paulson has analysed TLS using the inductive approach to the verification of security protocols [Pau99]. Our results are similar to Paulson's. Our models differ only in the protocol termination section: He considers *ClientFinished*

and *ServerFinished* equal while we take them different. Paulson's work involves long, detailed and intricate proof. His method is not natural, for guarantees ought to be proven within the logic. By contrast, in Schneider's approach, guarantees are built into the logic. Nevertheless, Paulson's approach is much powerful than Schneider's.

Extending his approach, Schneider has come out with new proof methods for the analysis of security protocols [Sch98]. The new methods do not accept the application of model checking techniques. So Schneider has advocated the use of mechanical theorem provers. However, as long as there is no reasonable level of automation in such tools, practitioners are not likely to use them. Our experimentations show that, even though constrained to finite-state systems, verification tools, such as CADP, can establish interesting properties of a security protocol.

6. Conclusions

We have reported on an analysis of TLS. Our experimentations show that TLS is secure, under two provisos: i) the certificates are valid, and ii) the private keys of server and client are confidential. We have used LOTOS to model TLS, the μ-calculus to express security properties that we wish it to meet and CADP to check our model against the specifications. Both the specification and the analysis closely follow Schneider's approach to the verification of security protocols. We have adapted Schneider's approach and applied it to the analysis of TLS, using different formalisms and proof methods. Our experimentations show that, even though constrained to finite-state systems, verification tools, such as CADP, can establish interesting properties of a security protocol.

References

[AFK96] P. Karlton A. Frier and P. Kocher. *The SSL 3.0 Protocol*. Netscape Communications Corp., Nov 18 1996.

[BB88] Tommaso Bolognesi and Ed Brinksma. Introduction to the iso specification language lotos. *isdn*, 14(1):25–59, January 1988.

[BK85] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985.

[CCI88] CCITT. *Recommendation X.509: The Directory - Authentication Framework*, 1988.

[DA99] T. Dierks and C. Allen. *The TLS Protocol Version 1.0*, Jan 1999. RFC 2246.

[Die97] S. Dietrich. *A Formal Analysis of the Secure Sockets Layer Protocol*. PhD thesis, Adelphi University, Garden Cyty, New York, 1997. Department of Mathematics and Computer Science.

[FGK+96] Jean-Claude Fernandez, Hubert Garavel, Alain Kerbrat, Radu Mateescu, Laurent Mounier, and Mihaela Sighireanu. Cadp (cæsar/aldebaran development package): A protocol validation and verification toolbox. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the 8th Conference on Computer-Aided Verification (New Brunswick, New Jersey, USA)*, volume 1102 of *LNCS*, pages 437–440. sv, August 1996. http://www.inrialpes.fr/vasy/cadp.

[Gut77] J. Guttag. Abstract data types and the development of data structures. In *Communications of the ACM*, pages 306–404, Jun 1977.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[Koz83] D. Kozen. Results on the Prositional μ-Calculus. In *Theoretical Computer Science*, pages 333–354, 1983.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[Pau99] Lawrence C. Paulson. Inductive analysis of the Internet protocol TLS. In *ACM Transactions on Information and System Security*, number 3 in 1, pages 332–351, Computer Laboratory University of Cambridge, 1999.

[Ros86] A. W. Roscoe. *Lecture notes on domain theory*. Oxford University, 1986.

[Ros94] A. W. Roscoe. Prospects for describing, specifying and verifying key-exchange protocols in csp and fdr. In *Formal Systems (Europe) Ltd*, 1994.

[Sch96a] S. Schneider. Security properties and csp. In *IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, 1996.

[Sch96b] S. Schneider. Using csp for protocol analysis: the needham-schroeder public-key protocol. Technical report, University of London, Royal Holloway, Nov 1996.

[Sch98] S. Schneider. Verifying authentication protocol in csp. *IEEE Transactions of Software Engineering*, 24(9):741–758, 1998.

[WS96] D. Wagner and B. Schneider. Analysis of the ssl 3.0 protocol. In *USENIX Workshop on Electronic Commerce(1996)*, pages 29–40. USENIX Association, D. Tygar Ed, 1996.

Authors addresses:
Departamento de ciencias computacionales
Tecnológico de Monterrey, Campus Estado de México
Carretera al Lago de Guadalupe, Km. 3.5, Atizapán, México
00471505@itesm.mx, raulm@itesm.mx