Automatic Verification of the TLS HandShake Protocol

Gregorio Díaz, Fernando Cuartero, Valentín Valero and Fernando Pelayo
Formal Methods Concurrency Research Group
University of Castilla-La Mancha
Campus Universitario, Avd. España s/n
02071, Albacete, Spain
(gregorio,fernando,valentin,fpelayo)@info-ab.uclm.es

ABSTRACT

E-commerce is based on transactions between client and server agents. These transactions require a protocol that provides privacy and reliability between these two agents. A widely used protocol on e-commerce is Transport Layer Security (TLS). In this paper we present a way to use Formal Methods to ensure the e-commerce properties of this protocol. Specifically we use a known tool for Model Checking (UPPAAL) to describe and analyze the behaviour of the protocol (by means of timed automata). Thus, with this tool we can make an automatic verification of TLS.

Categories and Subject Descriptors

K.4.4 [**Electronic Commerce**]: [Security]; C.2.2 [**Network Protocols**]: [Protocol verification]; D.2.4 [**Software / Program verification**]: [Model Checking].

General Terms

Authentication Protocols and Model Checking.

Keywords

e-Commerce, Security, Authentication Protocols, System Verification and Model Checking.

1. INTRODUCTION

Electronic commerce (e-commerce) became a buzzword as the information society developed rapidly throughout the 1990s. Internet has made e-commerce available to a wider user group, notably smaller enterprises and households. Amongst the business community the search for increased productivity and efficiency is expected to lead to even more enterprises adopting e-commerce as a way of doing business in the future. Whilst an ever growing awareness of the opportunities, technological developments in infrastructure and access devices, and falling access costs will facilitate this, fears about security and a lack of skills could hold it back.

SAC'04 March 14-17, 2004, Nicosia, Cyprus

An important goal over e-commerce is *security* [2, 9]. In order to ensure it, the Internet Engineering Task Force (IETF)is working now in the Transport Layer Security (TLS) [5]. The TLS protocol provides communications privacy over Internet. This protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

According to EITO estimates, e-commerce on the Internet was valued at 172 billion EUR in 2001 in European Union, close to 2% of GDP (Gross Domestic Product). In that way safety errors on e-commerce could be very expensive. But we can use *system validation* in order to avoid it [3, 10]. *System validation* is the process of determining the correctness of specifications, designs and products. Furthermore, it is a technique to support the quality control of the system design. A technique that implements *system validation* is **Model Checking** [8].

2. MODEL CHECKING ON E-COMMERCE WITH UPPAAL

As said before, the system validation is an important goal in order to avoid design errors. Thus, in e-commerce must be a goal too. Verification allows us to check if our protocols hold the expected behaviours and if they hold some safety properties. In this paper, we present a model of TLS Handshake protocol, which will be validated and verified using Model Checking techniques.

To put it in a nutshell, model checking is an automated technique that, given a finite-state model of a system and a property stated in some appropriate logical formalism, it systematically checks the validity of this property.

Methodology using Model Checking:

1. **Model the system** to capture the system behaviour.
2. **Validating the correctness** of the system model.
3. **Specifying the property and Verifying the system** to check if the model holds it.

A tool that implements Model Checking is UPPAAL [7, 6, 1], which has been widely proved in theoretical and industrial use cases.

To model the system, UPPAAL uses as a basic semantical model *timed transition system*, but a particular class of them called *networks of timed automata*. Timed automata consist of Locations and Transitions. The locations represent the

system states and the transitions represent changes between these locations.

The validating phase allows users to mach if the model behaviour holds the system behaviour. To make it, UPPAAL provides us a *Simulator*. As its name suggests, it can run simulations of the system behaviour.

In the verification phase, we must first establish the properties that the system must hold. To specify them, UPPAAL uses a temporal logic. We verify with UPPAAL simple safety properties such as "can we guarantee that a bad thing will not occur?" or "are we sure that eventually a good thing will occur?". These properties could be formalized in temporal logic as $\forall\Box\neg bad-thing$ and $\exists\Diamond good-thing$. In finite-state systems this kind of properties can be verified by checking all possible reachable states of a system.

3. THE TLS PROTOCOL

The Transmission Control Protocol/Internet Protocol (TCP/IP) governs the transport and routing of data over Internet. Other protocols, such as the HyperText Transport Protocol (HTTP), Lightweight Directory Access Protocol (LDAP), or Internet Messaging Access Protocol (IMAP), run "on top of" TCP/IP, in the sense that they all use TCP/IP to support typical application tasks such as displaying web pages or running email servers.

The Transport Layer Security protocol [5], TLS for short, runs above TCP/IP and below higher-level protocols such as HTTP or IMAP. It uses TCP/IP on behalf of the higher-level protocols, and it allows a TLS-enabled server to authenticate itself to an TLS-enabled client, it allows the client to authenticate itself to the server, and it allows both machines to establish an encrypted connection. These capabilities address fundamental concerns about communication over Internet and other TCP/IP networks:

TLS server authentication allows a user to confirm a server's identity. TLS enabled client software can use standard techniques of public-key cryptography to check that a server's certificate and public ID are valid and have been issued by a certificate authority (CA) listed in the client's list of trusted CAs. This confirmation might be important if the user, for example, is sending a credit card number over the network and wants to check the receiving server's identity.

TLS client authentication allows a server to confirm a user's identity. Using the same techniques as those used for server authentication, TLS-enabled server software can check that a client's certificate and public ID are valid and have been issued by a certificate authority (CA) listed in the server's list of trusted CAs. This confirmation might be important if the server, for example, is a bank sending confidential financial information to a customer and wants to check the recipient's identity.

An encrypted TLS connection requires all information sent between a client and a server to be encrypted by the sending software and decrypted by the receiving software, thus providing a high degree of confidentiality. Confidentiality is important for both parties to any private transaction. In addition, all data sent over an encrypted TLS connection is protected with a mechanism for detecting tampering–that is, for automatically determining whether the data has been altered in transit.

The TLS Record Protocol is used for encapsulation of various higher level protocols. One such encapsulated protocol, the TLS Handshake Protocol, allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data. The TLS Handshake Protocol provides connection security that has three basic properties:

- The peer's identity can be authenticated using asymmetric, or public key cryptography (e.g., RSA, DSS, etc.). This authentication can be made optional, but is generally required for at least one of the peers.
- The negotiation of a shared secret is secure: the negotiated secret is unavailable to eavesdroppers, and for any authenticated connection the secret cannot be obtained, even by an attacker who can place himself in the middle of the connection.
- The negotiation is reliable: no attacker can modify the negotiation communication without being detected by the parties of the communication.

3.1 The TLS Handshake protocol

The cryptographic parameters of the session state are produced by the TLS Handshake Protocol, which operates on top of the TLS Record Layer. When a TLS client and server first start communicating, they agree on a protocol version, select cryptographic algorithms, optionally authenticate each other, and use public-key encryption techniques to generate shared secrets.

The TLS Handshake Protocol involves the following steps:

- Exchange hello messages to agree on algorithms, exchange random values, and check for session resumption.
- Exchange the necessary cryptographic parameters to allow the client and server to agree on a premaster secret.
- Exchange certificates and cryptographic information to allow the client and server to authenticate themselves.
- Generate a master secret from the premaster secret and exchanged random values.
- Provide security parameters to the record layer.
- Allow the client and server to verify that their peer has calculated the same security parameters and that the handshake occurred without tampering by an attacker.

The handshake protocol can be summarized as follows: The client sends a client hello message to which the server must respond with a server hello message, or else a fatal error will occur and the connection will fail. The client hello and server hello are used to establish security enhancement capabilities between client and server. Following the hello messages, the server will send its certificate if it is to be authenticated. If the server is authenticated, it may request a certificate from the client. Now the server will send the server hello done message, indicating that the hello-message phase of the handshake is complete. The server will then wait for a client response. If the server has sent a certificate request message, the client must send the certificate message. The client key exchange message is now sent, and

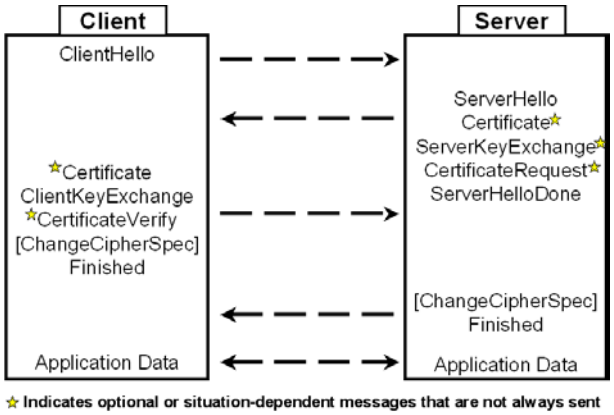

Figure 1: Message flow for a full handshake

the content of that message will depend on the public key algorithm selected between the client hello and the server hello. If the client has sent a certificate with signing ability, a digitally-signed certificate verify message is sent to explicitly verify the certificate. At this point, a change cipher spec message is sent by the client, and the client copies the pending Cipher Spec into the current Cipher Spec. The client then immediately sends the finished message under the new algorithms, keys, and secrets. In response, the server will send its own change cipher spec message, transfer the pending to the current Cipher Spec, and send its finished message under the new Cipher Spec. At this point, the handshake is complete and the client and server may begin to exchange application layer data. We can see the corresponding flow chart in Figure 1. Note that the ChangeChipherSpec is an independent TLS Protocol content type, and it is not actually a TLS handshake message.

4. USING THE MODEL CHECKING TOOL UPPAAL

UPPAAL is a tool suite for automatic verification of safety and bounded liveness properties of real-time systems modeled as networks of timed automata [6]. The UPPAAL engine transforms a certain class of linear hybrid systems to networks of timed automata, and implements techniques based on constraint-solving. UPPAAL also supports diagnostic model-checking, providing diagnostic information in the verification of a particular failure case of a real-time systems. The current version of UPPAAL is available at http://www.uppaal.com. This tool was developed during the spring of 1995 but, nowadays it is being extended with many additional features, as distribution, guided, parameterized, cost-optimal, hierarchical (UML) or probability (P_UPPAAL [4]).

In this section, we show how we can model the TLS Handshake protocol, validate the protocol and finally verify safety properties by using the tool UPPAAL.

4.1 Modeling the TLS Handshake protocol

The TLS protocol can considered as a real-time system consisting of communicating processes with shared clocks. Then it can be described by networks of timed automata extended with auxiliary data variables together with a notion of parallel composition. Instead of interpreting parallel composition as logical conjunction, we use a CCS-like interpretation of parallel composition, allowing one-to-one communication and interleaving.

By definition, a timed automaton is a standard finite-state automaton extended with a finite collection of real valued clocks. Clocks are assumed to proceed at the same rate and their values may be compared with natural numbers or reset to 0. We have extended the notion of timed automata to include integer variables, i.e. integer valued variables that may be compared to natural numbers or assigned to any value of the form $ax+b$ where $a, b \in \mathbf{Z}$ and x is the variable being reassigned. The model also allows clocks not only to be reset, but also to be set to any non-negative integer value.

DEFINITION 1. (Atomic Constraints) *Let C be a set of real valued clocks an I a set of integer valued variables. An atomic clock constraint over C is a constraint of the form: $x \sim n$, for $x \in C$, $\sim \in \{\leq, \geq, =\}$ and $n \in \mathbf{N}$. An atomic integer constraint over I is a constraint of the form: $i \sim n$, for $i \in I$, $\sim \in \{\leq, \geq, =\}$ and $n \in \mathbf{Z}$.*

By $C_c(C)$ we will denote the set of all clock constraints over C, and $C_i(I)$ will denote the set of all integer constraints over I.

DEFINITION 2. (Guards) *Let C be a set of real valued clocks and I a set of integer valued variables. A guard g over C and I is a formula generated by the following syntax: $g ::= c | g \wedge g$, where $c \in (C_c(C) \bigcup C_i(I))$.*

$\mathcal{B}(C, I)$ stands for the set of all guards over C and I.

DEFINITION 3. (Assignments) *Let C be a set of real valued clocks and I a set of integer valued variables. A clock assignment over C is a tuple $\langle v, c \rangle$, where $v \in C$ and $c \in \mathbf{N}$. An integer assignment over I is a tuple $\langle v, c_1, c_2 \rangle$ representing the assignment $v = C_1 \cdot v + c_2$, where $v \in I$ and $C_1, c_2 \in \mathbf{Z}$.*

We will use $\mathcal{A}(C, I)$ to denote the power-set of all assignments over I and C.

DEFINITION 4. (Timed automata) *A timed automaton A over a finite set of actions Act, clocks C, and integer variables I is a tuple $\langle L, l_0, E \rangle$, where L is a finite set of nodes (control-nodes), l_0 is the initial node, and $E \subseteq L \times \mathcal{B}(C, I) \times Act \times \mathcal{A}(C, I) \times L$ is the set of edges. We will write $l \xrightarrow{g,a,r} l'$ to denote, $\langle l, g, a, r, l' \rangle \in E$, .*

In order to study compositionality problems we introduce a parallel composition of timed automata. In order to get the kind of parallel composition we want, we have to introduce the notion of co-actions, which is done by defining a synchronization function $\mathcal{T}$.

DEFINITION 5. (Synchronization Set) *Let $\mathcal{T} \subseteq Act \times Act$ be a set of pairs such that:*

$$\langle a, b \rangle \in \mathcal{T} \Rightarrow \langle b, a \rangle \in \mathcal{T} \text{ for all } a, b \in Act$$

DEFINITION 6. (Parallel Composition) *Let A_1, A_2 be two timed automata. Then, the parallel composition $(A_1|A_2)$ is a timed automaton $\langle L, l_0, E \rangle$ where $(l_1|l_2) \in L$ whenever $l_1 \in L_1$ and $l_2 \in L_2$, $l_0 = (l_{1,0}|l_{2,0})$. The set of edges E is defined as follows:*

- $(l_1|l_2) \xrightarrow{g,\tau,r} (l_1'|l_2')$ *if* $(l_1 \xrightarrow{g_1,a_1,r_1} l_1') \wedge (l_2 \xrightarrow{g_2,a_2,r_2} l_2') \wedge (g = g_1 \cup g_2) \wedge (\langle a_1, a_2\rangle \in \mathcal{T}) \wedge (r = r_1 \cup r_2)$
- $(l_1|l_2) \xrightarrow{g,a,r} (l_1'|l_2)$ *if* $(l_1 \xrightarrow{g,a,r} l_1')$
- $(l_1|l_2) \xrightarrow{g,a,r} (l_1|l_2')$ *if* $(l_2 \xrightarrow{g,a,r} l_2')$

Note that parallel composition is commutative and associative.

Now, we show how the networks of timed automata may evolve. A state of a timed automata A is a pair $\langle l, u\rangle$, where l is a node of A and u is an assignment, mapping each clock in C to a value in $\mathbf{R}_+$, and each integer variable in I to a value in $\mathbf{Z}$. We will use $g(n)$ to denote that the assignment u satisfies the guard g. The initial state of A is $\langle l_0, u_0\rangle$, where u_0 is the assignment mapping all variables to 0. An automata may take two types of transitions, from state to state:

- Delay transitions: $\langle l, u\rangle \xrightarrow{\in(d)} \langle l, u'\rangle$ following the rules given in Definition 7.
- Action transitions: $\langle l, u\rangle \xrightarrow{g,a,r} \langle l', u'\rangle$ following the rules given in Definition 8.

DEFINITION 7. (Delay transitions) *Let* $\langle l, u\rangle$ *and* $\langle l', u'\rangle$ *be two states of a timed automaton* A*, and let* d *be a positive real. Then*

$$\langle l, u\rangle \xrightarrow{\in(d)} \langle l', u'\rangle \begin{cases} l' = l \\ u'(x) = u(x) + d & \text{if } x \in C \\ u'(x) = u(x) & \text{if } x \in I \\ d \leq M(l, u) \end{cases}$$

Where $M(l, u)$ is the maximal delay of $\langle l, u\rangle$, as follows:

$$M(l, u) = \begin{cases} sup\{t | g(u + t)\} & \text{if } \exists l' : l \xrightarrow{g,a,r} l' \\ \infty & \text{otherwise} \end{cases}$$

This value provides us an upper bound for the time that the automaton can stay at that node l. Once reached that time we are forced to evolve by executing some enabled action transition.

DEFINITION 8. (Action transitions) *Let* $\langle l, u\rangle$ *and* $\langle l', u'\rangle$ *be two states of a timed automata* A*. Then*

$$\langle l, u\rangle \xrightarrow{g,a,r} \langle l, u\rangle \text{ iff } \wedge g(u) \wedge \left(u'(x) = \begin{cases} c_0 & \text{if } x \in C \wedge \langle x, c_0\rangle \in r \\ c_1 u(x) + c_0 & \text{if } x \in I \wedge \langle x, c_1, c_0\rangle \in r \\ u(x) & \text{otherwise} \end{cases} \right)$$

Now we have presented the main features of UPPAAL models. Then, let us see how we can describe the TLS Handshake protocol by means of timed automata. In this task we firstly identify two processes in the protocol: the Client and the Sever processes.

Once we have identified the protocol processes, we model the message flow between them and their internal behaviour. The message flow is described by means of synchronizations between the Client and the Server (figures 2 and 3, respectively). The internal behaviour of each process can be easily modeled, by some "local" state and transitions. Note that the "Anonymous" message is really a server "KeyExchange" message. But this "KeyExchange" message is sent in an anonymous negotiation. Furthermore, the "HandShakeFailure" message represents the possible errors.

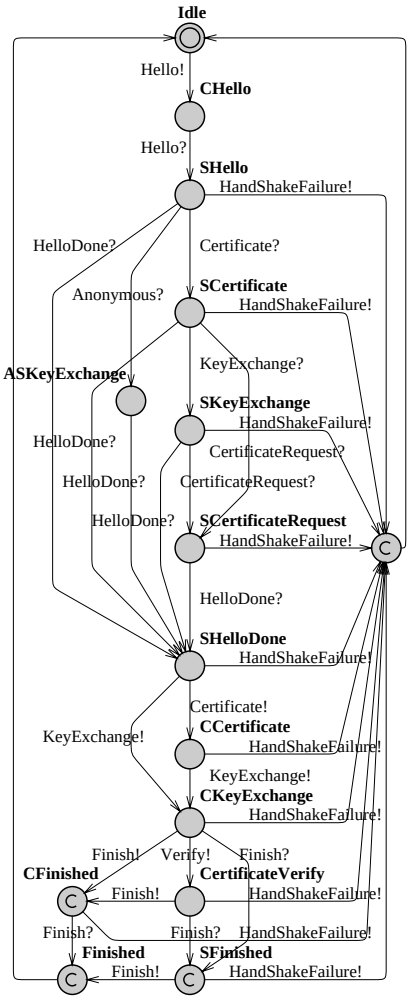

Figure 2: The Client process

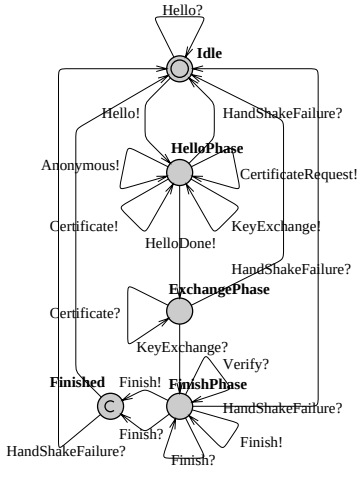

Figure 3: The Server process

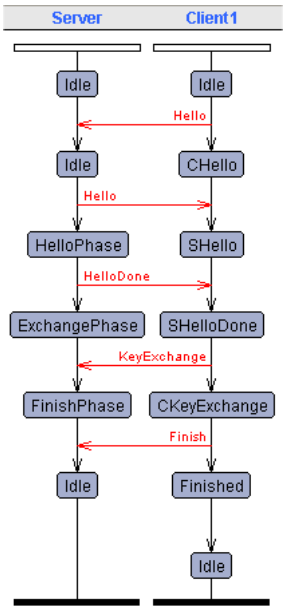

Figure 4: Trace for an abbreviated Handshake

4.2 Validating the protocol

In the validating phase we can check whether the model holds the system behaviour or not. This can partially be made by means of simulations. These are made by choosing different transitions and delays along the system evolution. At any moment during the simulation, you can see the variable values and the enabled transitions. Thus, you can choose the transition that you want to execute. Nevertheless, you can also select the random execution of transitions, and thus, the system evolves by executing transitions and delays which are selected randomly. We have some other options in the Simulator. For example, you can save simulations traces that can later be used to recover an specific execution trace. Actually, the simulation is quite flexible at this point, and you can back or forward in the sequence.

Then, with respect to our model of the TLS Handshake protocol, our main goal in the validation phase is to check the correctness of the message flow, taking into account the protocol definition.

We have made a number of simulations; and we have concluded that the system design satisfies the expected behaviour in terms of the message flow between the Client and the Server. For example, we show, in Figure 4, the negotiation trace for an abbreviated handshake.

4.3 Specifying and Verifying properties

Before starting the automatic verification, we must establish which are the properties that the model must fulfill.

We have divided these properties into three classes: Safety, Liveness and Deadlocks. These properties are specified by means of a *Temporal Logic*. The temporal Logic used by UPPAAL is described in [6].

Safety Properties allow us to check if our model satisfies some security restrictions. For example, if we have two trains that have to cross the same bridge, a security property is that both trains can not cross at the same time the bridge: $\forall\Box\neg(Train1.crossing \wedge Train2.crossing)$ or $\neg\exists\Diamond(Train1.crossing \wedge Train2.crossing)$

The main Safety properties are:

- A Server could not send the *ServerHello* message if the client has not sent the *ClientHello* message before, or vice versa:

$$\forall\Box Client.SHello \Rightarrow Server.HelloPhase \quad (1)$$

- A Client sends the *KeyExchange* message immediately after the server *certificate* message (or *ServerHello* message, if this is an anonymous negotiation):

$$\forall\Diamond(Client.CKeyExchange \Rightarrow Server.HelloPhase) \vee (Client.CKeyExchange \Rightarrow Server.ExchangePhase) \quad (2)$$

- The *Finished* message is always sent by the sender and Client, once the key exchange and the authentication processes are successful:

$$\forall\Box Client.Finished \Rightarrow Server.FinishPhase \quad (3)$$

Liveness Properties intend is to check that our model can evolve in the right order. Returning to the train example, if a train approaches the bridge, some time later, the train could cross it. $Train.approach \rightarrow Train.crossed$

Liveness Properties for our model are simple. If a client sends the message *ClientHello*, some time later, we could reach the message *finished*. Translating it into Temporal Logic we have:

$$Client.CHello \longrightarrow Client.Finished \quad (4)$$

On the other hand, if a server sends a ServerHellow, some time later, we could reach the message finished. Translating it into Temporal Logic we have:

$$Server.HelloPhase \longrightarrow Server.Finished \quad (5)$$

Deadlocks are clear restrictions. We could check if our model is deadlocks free:

$$\forall\Box\neg Deadlock \quad (6)$$

Let us now briefly describe how UPPAAL works in order to check properties. The technique is based on an interpretation using a finite-state symbolic semantics of networks. More precisely, we interpret the logic with respect to symbolic states of the form $(\bar{l}, D)$, where D is a constraint system (i.e. a conjunction of atomic clocks and data constraints) and $\bar{l}$ a control vector. Thus, a symbolic state $(\bar{l}, D)$ represents all the states $(\bar{l}, v)$, where v satisfies the constraint D. Based on this notion of symbolic state, the heart of the UPPAAL Model Checking procedure is the abstract reachability algorithm that we show in Figure 5, which reduces the reachability problem to that of solving simple constraint systems. This algorithm checks whether a timed automaton may reach a state satisfying a given formula β or not.

We observe that several operations of the algorithm are critical for an efficient implementation. Firstly, the algorithm depends heavily on the test operations for checking

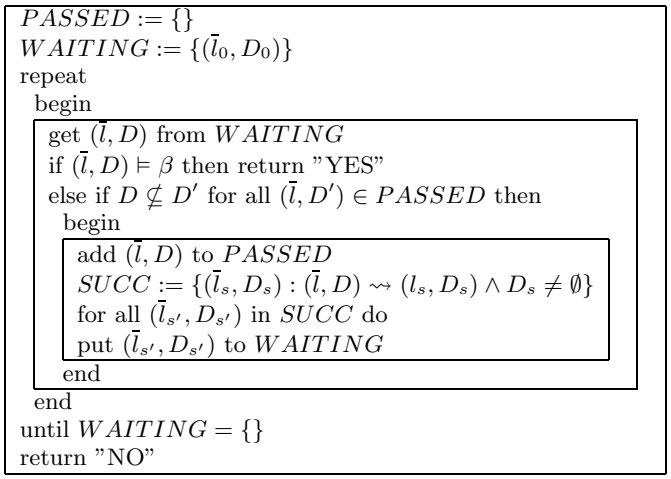

Figure 5: An Algorithm for Symbolic Reachability Analysis.

the inclusion $D \subseteq D'$ (i.e. the inclusion between constraints D and D') and the emptiness of D_s in constructing the successor set $SUCC$ of $(\bar{l}, D)$. Clearly, it is important to design efficient data structures and algorithms for the representation and manipulation of clock constraints. One such well-known data structure is that of Difference Bounded Matrices, DBM, which offers a canonical representation for constraint systems. It has been successfully employed by several real-time verification tools, e.g. UPPAAL and KRONOS.

Thus, we have specified the properties in UPPAAL as we have seen before and we have checked them by the UPPAAL verifier. The verifier outputs that we have obtained allow us to conclude that the safety properties 1, 2, 3 and the deadlock freeness property 6 are held by our model.

But we have obtained a negative answer in the liveness properties 4 and 5. By analyzing these properties, we supposed that when a *hello* message has arrived, the message *finished* will be sent later. But sometimes, the client or the server can send the *Handshake-Failure* message, which means "something wrong has happened" and then the negotiation ends. Then, we can replace the properties 4 and 5 by: When the hello message has been sent, sometimes the negotiation ends. It is expressed by 7 and 8, which are held by our model.

$$\forall\Diamond Client1.CHelloimplyServer.FinishPhase \quad (7)$$

$$\forall\Diamond Server.HelloPhaseimplyClient1.Finished \quad (8)$$

5. CONCLUSIONS AND FUTURE WORK

In this paper we have presented the validation and verification of the TLS Handshake Protocol. The properties verified ensure the correct message flow. The techniques used in this paper are not particularly novel, and they have been used in some industrial and theoretical use cases and protocols. Then we may ask ourselves, why incorrect protocols and software are still appearing. One reason seems to be that protocol developers and developers in general, are failing to learn from the mistakes of others. Another reason and perhaps the most significant one, is that they are usually unaware of formal technique of verification, or even they are applied incorrectly.

As future work we will develop a new tool in order to model, validate and verify protocols and other systems called. This tool is called P_UPPAAL [4]. It includes a new feature with respect to UPPAAL, P_UPPAAL has the capability to work with Probabilistic Real-Time Systems. The main goal is checking properties that merge at the same time probabilistic and real-time requisites, e.g. properties such as "It is possible to reach a certain state within three time units, with a probability greater than 0.3".

6. ACKNOWLEDGMENTS

This work has been supported by spanish grants CYCIT (ref. TIC 2003-07848-C02-02) and JCCM (ref PAC-03001).

7. REFERENCES

[1] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, Yi Wang, and C. Weise, *New Generation of* UPPAAL, Int. Workshop on Software Tools for Technology Transfer, June 1998.

[2] D. Bolignano, *Towards the formal verification of electronic commerce protocols*, In Proc. 10th IEEE Computer Security Foundations Workshop, 1997.

[3] Philippa Broadfoot and Gavin Lowe, *On distributed security transactions that use secure transport protocols*, 2003.

[4] G. Díaz, D. Cazorla, F. Pelayo, F. Cuartero, and V. Valero, *Verifying and capturing probabilistic bechaviours of real-time systems*, 19th Annual UK Performance Engineering Workshop, 2003.

[5] Internet Engineering Task Force, *The tls protocol version 1.1*, work in progress (June 2003), http://www.ietf.org/internet-drafts/draft-ietf-tls-rfc2246-bis-05.txt.

[6] K. Larsen, P. Pettersson, and Wang Yi, UPPAAL *in a Nutshell*, Int. Journal on Software Tools for Technology Transfer **1** (1997), no. 1–2, 134–152.

[7] F. Larsson, K. Larsen, P. Pettersson, and Wang Yi, *Efficient Verification of Real-Time Systems: Compact Data Structures and State-Space Reduction*, Proc. of the 18th *IEEE Real-Time Systems Symposium*, 1997.

[8] Gavin Lowe, *Towards a completeness result for model checking of security protocols*, Proc. of The 11th Computer Security Foundations Workshop, 1999.

[9] A. W. Roscoe, *Proving security protocols with model checkers by data independence techniques*, In Proceedings of the IEEE Computer Security Foundations Workshop, 1998.

[10] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe, *Modelling and analysis of security protocols*, Addison Wesley, 2001.