Timed Automata Based Model Checking of Timed Security Protocols

Mirosław Kurkowski
Institute of Mathematics and Computer Science, Jan Długosz University
Armii Krajowej 13/15, 42-200 Częstochowa, Poland
m.kurkowski@ajd.czest.pl

Wojciech Penczek†
Institute of Computer Science, Polish Academy of Sciences
Ordona 21, 01-237 Warsaw, Poland, and
Institute of Informatics, Podlasie Academy
Sienkiewicza 51, 08-110 Siedlce, Poland
penczek@ipipan.waw.pl

Abstract. A new approach to verification of timed security protocols is given. The idea consists in modelling a finite number of users (including an intruder) of the computer network and their knowledge about secrets by timed automata. The runs of the product automaton of the above automata correspond to all the behaviours of the protocol for a fixed number of sessions. Verification is performed using the module BMC of the tool VerICS.

Keywords: timed security protocols, model checking, authentication.

1. Introduction

Automated verification of security protocols is a very active and important area of computer science, which has been an object of an intensive research for several years in both academic and commercial...
institutions. There are numerous approaches to verification of untimed security protocols [1, 2, 14, 11] as well as of time dependent protocols [5, 6, 8, 9, 10, 15]. Algorithmic approaches include mainly methods based on model checking. Intuitively, model checking of a security protocol consists in checking whether a model of the protocol contains an execution or a reachable state that is representing an attack on the protocol. Comparing to standard model checking methods for communicating protocols or for distributed systems, the main difficulty is caused by the need to model both the intruder which is responsible for generating attacks as well as changes of knowledge (about keys, nonces, etc.) of the participants.

Our former paper [14] offered a new method for verifying untimed security protocols. The main idea consisted in using networks of synchronized automata for modelling separately the executions of a protocol and a participant knowledge about secrets. Thanks to that we have developed a very distributed representation of the a participant behaviour in protocol executions, which is crucial for an efficient symbolic encoding and model checking. To this aim we defined a novel semantics of security protocols, where the notion of a computational structure and an interpretation was based on the ideas of [13].

In this paper we extend the above approach to timed security protocols. We give a method for representing the executions of a timed security protocol (within a computational structure for a bounded number of sessions) by the runs of the product timed automaton of a network of the timed automata for the participants and their knowledge. Then we show how to look for attacks on authentication. To this aim we use Bounded Model Checking (BMC) [12], which consists in translating the problem of reachability in the product timed automaton to satisfiability of some propositional formula.

Our approach is closer to the work by Corin at al. [5], where security protocols are directly modelled in terms of networks of timed automata extended with integer variables, and verified with UppAal [3]. The authors of [5] address timeouts and retransmissions, but do not show how one can model timestamps in such an approach. Another method [10], close to ours, also uses timed automata without integer variables, but indirectly. Timed Security protocol are modelled in the higher-level language \( IL^1 \) [7], and only then translated to timed automata. The paper [10] offers a new method of verifying security protocols, but the translation from \( IL \) can suffer from the state explosion in the resulting timed automata.

The rest of the paper is organised as follows. In Section 2 we introduce syntax for dealing with timed security protocols. A computational structure generating all the runs of the protocols considered is defined in Section 3. A method for finding attacks by analysing computations of the protocol is shown in Section 3.1. Section 4 defines network of timed automata for representing the participants of a protocol and their knowledge about secrets. Then, experimental results are given in Section 5 and some concluding remarks in Section 6.

## 2. Syntax of Timed Cryptographic Protocols

In this section we introduce syntax for dealing with timed security protocols. In what follows, for any set \( Z \) by \( 2^Z \) we denote a set of all the finite subsets of \( Z \).

Next, we define the following basic syntactic notions of our model\(^2\).

- \( T_P = \{ P_1, P_2, \ldots, P_n_P \} \) is a set of symbols representing the users of the computer network,
- \( T_I = \{ I_{P_1}, I_{P_2}, \ldots, I_{P_n_P} \} \) is a set of symbols representing the identifiers of the users,

\(^1\) \( IL \) is the acronym for the Intermediate Language (ver. 1.0).

\(^2\) We assume that \( n_P, n_N, n_e, n_L, \) and \( n_R \) are some fixed natural numbers.
Definition 2.1. By a set of letter terms $T$ we mean the smallest set satisfying the following conditions:
1. $T_P \cup T_I \cup T_K \cup T_N \cup T_T \cup T_L \subseteq T$.
2. If $X \in T$ and $Y \in T$, then the concatenation $X \cdot Y \in T$.
3. If $X \in T$ and $K \in T_K$, then $\langle X \rangle_K \in T$.
4. If $X \in T$, then the hash value of the letter term $X h(X) \in T$.

Next, we define some useful relations over the set $T$.

Definition 2.2. Let $\sim_T \subseteq T \times T$ be the smallest relation (called (immediate) subterm relation), which satisfies the following conditions:
1. If $X, Y \in T$, then $X \sim_T X \cdot Y$ and $Y \sim_T X \cdot Y$.
2. If $X \in T$ and $K \in T_K$, then $X \sim_T \langle X \rangle_K$ and $K \sim_T \langle X \rangle_K$.
3. If $X \in T$, then $X \sim_T h(X)$.

By $\preceq_T$ we denote the transitive and reflexive closure of $\sim_T$. Next, for any $\mathcal{X} \subseteq T$ we define a sequence of the sets $(\mathcal{X}^n)_{n \in \mathbb{N}}$ that are subsets of $T$:

$\mathcal{X}^0 \overset{\text{def}}{=} \mathcal{X}$,
$\mathcal{X}^{n+1} \overset{\text{def}}{=} \mathcal{X}^n \cup \{Z \in T \mid (\exists X, Y \in \mathcal{X}^n, K \in \mathcal{X} \cap T_K) Z = X \cdot Y \lor Z = \langle X \rangle_K \lor Z = h(X)\}$.

Intuitively, the set $\mathcal{X}^{n+1}$ contains the, gradually built, letter terms from $\mathcal{X}^n$ using the operations of composition, encryption and hashing. For $\mathcal{X} \in 2^n_{fin}$ the set $\text{Comp}(\mathcal{X}) \overset{\text{def}}{=} \bigcup_{n \in \mathbb{N}} \mathcal{X}^n$ is composed of all the letters that can be constructed out of elements of $\mathcal{X}$ only (decription is not allowed here). By $\mathcal{R}_+$ we denote the set of positive real numbers while $\mathcal{R}_+ = \mathcal{R}_+ \cup \{0\}$.

In order to deal with timed protocols, we define the set of time constraints to be used in specifications of time dependences of protocol instructions.

Definition 2.3. The set of time constraints $\mathcal{C}$ is given by the following grammar:

$$\text{tc} ::= true \mid \tau_i - \tau_j \leq \mathcal{L}_\mathcal{F} \mid \text{tc} \land \text{tc},$$ where $\tau_i \in T_R, \tau_j \in T_T; \mathcal{L}_\mathcal{F} \in T_L$.

Now, we are in a position to define the syntax of a protocol step and then the syntax of a protocol itself. Our notion of a step is clearly more complicated than in the common language as it provides the information not only about the sender $P$, the receiver $Q$, and the letter $\mathcal{L}$ sent from $P$ to $Q$, but also about the letters and the generated secrets that are necessary to compose $\mathcal{L}$. The intended aim of this extra information is to point out to additional actions of the sender like generating new secrets or composing the letter $\mathcal{L}$.

$\langle X \rangle_K$ is a term that is interpreted as a ciphertext containing the letter $X$ encrypted with the key $K$.
Definition 2.4. By a (protocol) step $\alpha$ we mean a pair $(\alpha^1, \alpha^2)$, where $\alpha^1$ is a triple $(P, Q, L) \in T_P \times T_P \times T$, and $\alpha^2$ is a 4-tuple $(\tau, \mathcal{X}, G, tc) \in T_R \times 2^T \times 2^{T_K \cup T_N \cup T_T} \times C$ with the following intuitive meaning: $P$ - the sender of the step, $Q$ - the receiver of $L$, $L$ - the letter sent from $P$ to $Q$, and $\tau$ - the time of the step execution, $\mathcal{X}$ - the set of sent letters necessary to compose $L$, $G$ - the set of generated secrets necessary to compose $L$, $tc$ - the step time constraint, satisfying the following conditions:

1. $\mathcal{X}$ (no sending to itself).
2. $\mathcal{X} \in \text{Comp} (\mathcal{X}) \land (\forall \mathcal{Y} \subseteq \mathcal{X} \rightarrow (\mathcal{Y} \in \text{Comp} (\mathcal{Y}) \Rightarrow \mathcal{Y} = \mathcal{X}))$, ($\mathcal{X}$ is a minimal set from which $L$ can be constructed).
3. $L \subseteq \mathcal{X}$ (the secrets of $G$ are elements of $\mathcal{X}$).

By a protocol $\Sigma$ we mean a finite sequence $(\alpha_1, \ldots, \alpha_n)$ of steps.

Notice that $\alpha^1 = (P, Q, L)$ describes a protocol step with the following intuitive meaning Sender $P$ sends the message $L$ to Receiver $Q$. This is usually denoted in Common Language by $P \rightarrow Q : L$.

Example 2.1. We consider Wide Mouth Frog Protocol (WMF) as a simple working example. The syntax of WMF is as follows. $T_P = \{A, B, S\}$, $T_I = \{I_A, I_B\}$, $T_K = \{K_{AS}, K_{BS}, K_{AB}\}$, $T_T = \{\tau_A, \tau_S\}$, $T_L = \{L_F\}$. WMF is given by the following sequence of steps: $(\alpha_1, \alpha_2)^5$, where:

$\alpha_1 = (\alpha^1_1, \alpha^2_1) = (A; S; I_A, \langle \tau_A, I_B, K_{AB} \rangle K_{AS})$, $\alpha_2 = (\alpha^1_2, \alpha^2_2) = (S; B; \langle \tau_S, I_A, K_{AB} \rangle K_{BS})$.

The time constraint $\tau_1 - \tau_A \leq L_F$ says that $S$ can receive the message $\langle \tau_A, I_B, K_{AB} \rangle K_{AS}$ only under condition that the difference between the time $\tau_1$ of sending/receiving the message and the time $\tau_A$ of generating the ticket is less than $L_F^6$. Similarly, the time constraint $\tau_2 - \tau_A \leq L_F$ says that $S$ can send (and $B$ - receive) the message only under the condition that the difference between the time $\tau_2$ of sending/receiving the message and the time $\tau_A$ of generating the ticket is less than $L_F$.

Example 2.2. The syntax of the Kerberos Protocol (KP)$^7$ is as follows. $T_P = \{A, B, S\}$, $T_I = \{I_A, I_B\}$, $T_K = \{K_{AS}, K_{BS}, K_{AB}\}$, $T_T = \{\tau_B, \tau_S\}$, $T_L = \{L_F\}$. KP is given by the following sequence of steps: $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$, where:

$\alpha_1 = (\alpha^1_1, \alpha^2_1) = (A; S; I_A, I_B)$, $\alpha_2 = (\tau_1; \{I_A, I_B\}; \emptyset; \text{true})$.

$^5$For simplicity reasons we assume that the time of sending is equal to the time of receiving the message, so no delay on delivery of the message. However, delays are allowed between two consecutive steps of the protocol.

$^6$The description of WMF in common language is as follows:

1. $A \rightarrow S : I_A, < T_A, I_B, K_{AB} > K_{AS}$
2. $S \rightarrow B : < T_S, I_A, K_{AB} > K_{BS}$.

$^7$The description of KP in common language is as follows:

1. $A \rightarrow S: I(A), I(B)$,
2. $S \rightarrow A: < T_S, L, K_{AB}, I(A) > K_{BS}, < T_S, L, K_{AB}, I(B) > K_{AS}$,
3. $A \rightarrow B: < T_S, I(A) > K_{AB}, < T_S, L, K_{AB}, I(A) > K_{BS}$,
4. $B \rightarrow A: < T_S > K_{AB}$.
\[ \alpha_2 = (\alpha_1^2), \alpha_1 = (S; A; (\tau_S, L_A, K_{AB}, I(A))K_{BS}, (\tau_S, L_A, K_{AB}, I(B))K_{BS}). \]
\[ \alpha_3 = (\alpha_2^3, \alpha_2^1), \alpha_3^2 = (\alpha_2^3, \alpha_2^1), \alpha_3^1 = (A; B; (\tau_S, I(A))K_{AB}, (\tau_S, L_A, K_{AB}, I(A))K_{BS}). \]
\[ \alpha_4 = (\alpha_1^4, \alpha_1^2), \alpha_4^2 = (\tau_2; \{\tau_S, L_{\mathcal{F}}, K_{AB}, I(A), I(B), K_{AB}, K_{BS}; \{\tau_S, L_{\mathcal{F}}, K_{AB}\}; \tau_3 - \tau_S \leq L_{\mathcal{F}}). \]

3. Computational Structure

In this section we define a computational structure, which generates all the computations under the interpretations considered of a security protocol. Later, we represent these computations by runs of the product automaton of a network of timed automata. We start with defining the following sets:

- \( P = \{p_1, p_2, \ldots, p_{n_p}\} \) - a set of the honest participants in the network,
- \( P_T = \{\tau_1, \tau(p_1), \tau(p_2), \ldots, \tau(p_{n_p})\} \) - a set of the dishonest participants containing the Intruder and the Intruder impersonating the participant \( p_i \) for \( 1 \leq i \leq n_p \),
- \( I = \{i_1, i_2, \ldots, i_{n_p}, i_3\} \) - a set of the identifiers of the participants in the network,
- \( K = \bigcup_{i=1}^{n_p} \{k_{p_i}, k_{p_i}^{-1}\} \) - a set of the cryptographic keys of the participants,
- \( N = \bigcup_{i=1}^{n_p} \{n_{p_i}^1, \ldots, n_{p_i}^{k_{p_i}}\} \cup \{n_1^1, \ldots, n_i^{k_{p_i}}\} \) - a set of the nonces,
- \( T = \bigcup_{i=1}^{n_p} \{t_{p_i}^1, \ldots, t_{p_i}^{n_{p_i}}\} \) is a set of the users’ time tickets,
- \( \Gamma = \{l_1, l_2, \ldots, l_{n_o}\} \subseteq R^+ \) is a set of the lifetimes.

**Definition 3.1.** By a set of letters \( L \) we mean the smallest set satisfying the following conditions:
1. \( L \) is the smallest set over the set \( L \).
2. If \( x, y \in L \), then the concatenation \( x \cdot y \in L \).
3. If \( x \in L \) and \( k \in K \), then \( \langle x \rangle_k \in L \), \( \langle x \rangle_k \) is a ciphertext consisting of the letter \( x \) encrypted with \( k \).
4. If \( x \in L \), then \( h(x) \in L \), \( h(x) \) is a hash value of the letter \( x \).

Next, we define some auxiliary relations over the set \( L \).

**Definition 3.2.** Let \( \prec \) be the smallest relation (called (immediate) subletter relation) satisfying the following conditions:
1. If \( x, y \in L \), then \( x \prec x \cdot y \) and \( y \prec x \cdot y \),
2. If \( x \in L \) and \( k \in K \), then \( x \prec \langle x \rangle_k \) and \( k \prec \langle x \rangle_k \).
3. If \( x \in L \), then \( x \prec h(x) \).

By \( \preceq \) we denote the transitive and reflexive closure of \( \prec \). Next, for any \( X \subseteq L \) we define a sequence of the sets \( \langle X^n \rangle_{n \in \mathbb{N}} \) that are also subsets of \( L \) where
\[ X^0 \equaldef X, \]
\[ X^{n+1} \equaldef X^n \cup \{z \in L \mid (\exists x, y \in X^n, k \in X \cap K) z = x \cdot y \lor z = \langle x \rangle_k \lor z = h(x)\}. \]

The intuition behind this definition is the same as for the corresponding one in Section 2, i.e., the set \( X^{n+1} \) contains the, gradually built, letters from \( X^n \) using the operations of composition, encryption and hashing. Next, define the set \( \text{Comp}(X) \equaldef \bigcup_{n \in \mathbb{N}} X^n \), which consists of all the letters that can be

\[ ^8 \text{As before, we assume that } n_p \text{ and } k_{p_i} \text{ are some fixed natural numbers. For simplicity, we take the same number of nonces for each user. Additionally we assume that } n_t \text{ and } n_l \text{ are some fixed natural numbers.} \]
composed out of elements of $X$ only (decription is not allowed here) and the set

$$\text{Sublet}(X) \overset{def}{=} \{ l \in L \mid (\exists x \in X) \, l \preceq x \}$$

which contains all the subletters of $X$.

**Definition 3.3.** Let $X \subseteq L$ and $K \subseteq K$. Define the set $\xi_K(X) \subseteq L$ as the smallest set of letters satisfying the following conditions:

1. $X \subseteq \xi_K(X)$,
2. if $l \cdot m \in \xi_K(X)$, then $l \in \xi_K(X)$ and $m \in \xi_K(X)$,
3. if $(l)_k \in \xi_K(X)$ and $k \in \xi_K(X) \cup K$, then $l \in \xi_K(X)$.

The set $\xi_K(X)$ contains all the letters which can be retrieved from $X$ by decomposing a concatenation or decrypting a letter using a key, which is either in $\xi_K(X)$ or in $K$. By $\xi(X)$ we mean the set $\xi_\emptyset(X)$.

Next, we define partial interpretations and interpretations of the letter terms of $T$ which are used for defining the runs of protocols.

**Definition 3.4.** By a partial interpretation of the set of the letter terms $T$ we mean any injection $\overline{f} : T \rightarrow L$ satisfying the following conditions:

1. $\overline{f}(T_P) \subseteq P \cup P_\ast$, $\overline{f}(T_I) \subseteq I$, $\overline{f}(T_K) \subseteq K$, $\overline{f}(T_N) \subseteq N$, $\overline{f}(T_L) \subseteq \Gamma$, $\overline{f}(T_T) \subseteq T$,
2. $(\forall X, Y \in T) \overline{f}(X \cdot Y) = \overline{f}(X) \cdot \overline{f}(Y)$ (homomorphism),
3. $(\forall X \in T)(\forall K \in T_K) \overline{f}((X)_K) = \overline{f}(X)_{\overline{f}(K)}$ (homomorphism),
4. $(\forall X \in T) \overline{f}(h(X)) = h(\overline{f}(X))$ (homomorphism),
5. If $\overline{f}(P) = p$ for $p \in P$, then $\overline{f}(I_P) = i_p, \overline{f}(N_P) \in \{ n_p^1, \ldots, n_p^{k_p} \}, \overline{f}(K_P) = k_p$ and $\overline{f}(K^{-1}_P) = k_p^{-1}$.
6. If $\overline{f}(P) = i$, then $\overline{f}(I_P) = i, \overline{f}(K_P) = k_i$ and $\overline{f}(K^{-1}_P) = k_i^{-1}$.
7. If $\overline{f}(P) = i(p)$, then $\overline{f}(I_P) = i_p, \overline{f}(K_P) = k_p$ and $\overline{f}(K^{-1}_P) = k_p^{-1}$.
8. $\overline{f}(T_P) \backslash P_\ast \neq \emptyset$.

The condition 1 states that the atomic terms are mapped into the corresponding objects of the computational structure, i.e., the symbols representing the participants are mapped into the participants, etc. The conditions 2 and 3 guarantee the homomorhical separation between the symbols mapped. The condition 4 says that the symbols related to a given participant are mapped into the corresponding objects (the identifiers, the keys, the nonces) in the structure. The condition 5 determines that if Intruder $i$ wants to behave honestly in an execution of the protocol, then it uses its own identifier and keys. There is no condition on the nonces used by the Intruder, as we assume that it can use any nonce. The condition 6 states that if Intruder $i$ impersonates another participant $p$ in some interpretation, then in any execution under this interpretation $p$’s keys and $i$’s identifier need to be used by $i$. Then, due to the condition 1, no participant symbol is mapped to $p$ in this interpretation. The last condition says that at least one honest participant takes part in each interpretation.

**Definition 3.5.** For a given partial interpretation $\overline{f}$ by an interpretation (associated with $\overline{f}$) of the set of the letter terms $T$ and the set of time variables $T_R$ we mean any injection $f : T \cup T_R \rightarrow L \cup R$, such that $f \mid_{T \backslash T_P} = \overline{f}$ and $f(T_T \cup T_R) \subseteq R$. The time tickets and the times of sending/receiving of messages are finally mapped into time moments represented by nonnegative real numbers.

In order to define later an interpretation of a protocol step in which Intruder is the sender, we need the notion of a set of generators for a letter.
Definition 3.6. Let \( l \in \mathbf{L} \) be a letter and \( X \subseteq \mathbf{L} \). The set \( X \) is said to be a set of generators of \( l \) (denoted by \( X \vdash l \)) if the following conditions are met:
1. \( X \subseteq \text{Sublet}\{\{l\}\} \),
2. \( l \in \text{Comp}(X) \),
3. \( \forall m \in X)(m \notin \text{Comp}(X \setminus \{m\}) \),
4. \( \forall m \in X)(l \notin \text{Comp}(X \setminus \{m\}) \).

Intuitively, we have \( X \vdash l \) if all the elements of \( X \) are subletters of \( l \), \( l \) can be composed out of the elements of \( X \), and \( X \) is such a minimal set.

We extend an interpretation \( f \) to the time constraints of \( \mathcal{C} \), which is defined inductively as follows:
- \( f(\text{true}) := \text{true} \),
- \( f(\tau_1 - \tau_2 \leq \mathcal{L}_\tau) := f(\tau_1) - f(\tau_2) \leq f(\mathcal{L}_\tau) \),
- \( f(\text{tc}_1 \land \text{tc}_2) := f(\text{tc}_1) \land f(\text{tc}_2) \).

Having defined a set of letter generators and an interpretation of \( \mathcal{T} \), we are now in a position to apply it to a protocol step and then to the whole protocol.

Definition 3.7. Consider a step \( \alpha = (\alpha_1, \alpha_2) = ((\mathcal{P}, \mathcal{Q}, \mathcal{L}), (\tau, \mathcal{X}, \mathcal{G}, \text{tc})) \) of a given protocol \( \Sigma \) and an interpretation \( f \) of \( \mathcal{T} \cup \mathcal{T}_\mathcal{R} \) which satisfies the following condition \( \bigwedge_{\tau \in \mathcal{G} \cap \mathcal{T}_\mathcal{R}} (f(\alpha_1) = f(\tau)) \), i.e., the time assigned to each time ticket generated is equal to the time of the protocol step. By the \( f \)-interpretation of the step \( \alpha \) (denoted by \( f(\alpha) \)) we mean the following tuple:
- \( ((f(\mathcal{P}), f(\mathcal{Q}), f(\mathcal{L})), (f(\alpha_1), f(\mathcal{X}), f(\mathcal{G}), f(\text{tc}))) \), if \( f(\mathcal{P}) \in \mathcal{P} \),
- \( ((f(\mathcal{P}), f(\mathcal{Q}), f(\mathcal{L})), (f(\alpha_1), \{X \mid X \vdash f(\mathcal{L})\}, f(\text{tc}))) \), if \( f(\mathcal{P}) \in \mathcal{P}_\mathcal{R} \).

In the case when Intruder is the sender, we assume that it can compose a letter \( f(\mathcal{L}) \) from any set which generates \( f(\mathcal{L}) \). We also assume that Intruder has got a set of nonces at its disposal and it does not need to generate them. The reason is that Intruder can use the same nonce many times and in many sessions.

In order to define protocol executions and knowledge of the participants and Intruder we need to introduce the following auxiliary notions. If \( f(\alpha_1) = ((p, q, l), (\tau, X, G, c)) \), for some \( p, q \in \mathcal{P} \cup \mathcal{P}_\mathcal{R} \), \( X \in 2^{\mathcal{L}_m} \), \( G \in 2^{\mathcal{K}_{\mathcal{L}m}^\mathcal{N}} \), and \( l \in \mathbf{L} \), then we use the following notations:
- \( \text{Send}^f(\alpha_1) = p \) (the sender of \( f(\alpha_1) \)), \( \text{Lett}^f(\alpha_1) = l \) (the letter of \( f(\alpha_1) \)), \( \text{Gen}^f(\alpha_1) = G \) (the set of generated new secrets in \( f(\alpha_1) \)), \( \text{Resp}^f(\alpha_1) = q \) (the responder of \( f(\alpha_1) \)), \( \text{Part}^f(\alpha_1) = \{\text{Send}^f(\alpha_1), \text{Resp}^f(\alpha_1)\} \), \( \text{Time}^f(\alpha_1) = \tau \) and \( \text{TConstr}^f(\alpha_1) = c \).

In addition if \( \text{Send}^f(\alpha_1) \in \mathcal{P} \), then let \( \text{Comp}^f(\alpha_1) = X \) (the set of letters that are sufficient to compose \( \text{Lett}^f(\alpha_1) \)) and if \( \text{Send}^f(\alpha_1) \in \mathcal{P}_\mathcal{R} \), then let \( \text{Comp}^f(\alpha_1) = \bigcup_{X \vdash \text{Lett}^f(\alpha_1)} X \) (the union of sets which generate \( \text{Lett}^f(\alpha_1) \)). Similarly, for a partial interpretation \( \overline{f} \) the interpretation \( f \) is associated with, we use notations:
- \( \text{Send}^{\overline{f}}(\alpha_1), \text{Lett}^{\overline{f}}(\alpha_1), \text{Gen}^{\overline{f}}(\alpha_1), \text{Resp}^{\overline{f}}(\alpha_1) \) for \( \overline{f}(\mathcal{P}), \overline{f}(\mathcal{L}), \overline{f}(\mathcal{G}), \overline{f}(\mathcal{Q}) \), respectively.

Definition 3.8. Let \( f \) be an interpretation satisfying the following conditions:
- \( f(\alpha_1), f(\alpha_2), \ldots, f(\alpha_n) \) are \( f \)-interpretations of the steps of the protocol \( \Sigma = (\alpha_1, \alpha_2, \ldots, \alpha_n) \), and
- \( \bigwedge_{i=1}^{n} \text{TConstr}^f(\alpha_i) \equiv \text{true} \). Then, by the \( f \)-execution of a protocol \( \Sigma \) we mean the sequence:
  \( f(\Sigma) = (f(\alpha_1), f(\alpha_2), \ldots, f(\alpha_n)) \).

For a set of interpretations \( \mathcal{F} \), we define the set \( \text{Comp}^f_\mathcal{F} (\text{Comp}^f_\mathcal{T}) \) of the letters, which the participant \( p \in \bigcup_{f \in \mathcal{F}} f(\mathcal{P}) \setminus \mathcal{P}_\mathcal{R} \) (Intruder \( i \), resp.) needs in order to compose all the letters sent in an execution under any interpretation \( f \in \mathcal{F} \).
Definition 3.9. The set $Comp^p_F = \bigcup_{1 \leq i \leq n} \bigcup_{f \in F \mid Send^f(\alpha_i) = p} Comp^{f(\alpha_i)}$ for an honest user $p$ is the union of all the sets $Comp^{f(\alpha_i)}$ for all $i \leq n$ and $f \in F$, where $Send^f(\alpha_i) = p$.

The set $Comp^p_F = \bigcup_{1 \leq i \leq n} \bigcup_{f \in F \mid Send^f(\alpha_i) \in P_e} Comp^{f(\alpha_i)}$ is the union of all the sets $Comp^{f(\alpha_i)}$ for all $i \leq n$ and $f \in F$, where $Send^f(\alpha_i) \in P_e$.

Consider any finite sequence of interpretations of $k$ protocol steps $\tau = (f^1(\alpha_{i_1}), f^2(\alpha_{i_2}), \ldots, f^k(\alpha_{i_k}))$.

For every $p \in \bigcup_{i=1}^k f^i(T_P)$ we define a sequence of the participant’s knowledge $(\kappa_p^j)_{j=1,\ldots,k}$ at the steps of the protocol.

Definition 3.10. For an honest user $p \in \bigcup_{f \in F} f(T_P) \setminus P_e$ his knowledge at the step $j$ is given inductively as follows:

$\kappa_p^0 = I \cup \{k_p^{-1}\} \cup \{q \mid q \in P\} \cup \{k_1\}$,

$\kappa_p^{j+1} = \begin{cases} \kappa_p^j & \text{if } p \notin Part^{f^{j+1}(\alpha_{i_{j+1}})}, \\ \kappa_p^j \cup \text{Gen}^{f^{j+1}(\alpha_{i_{j+1}})} & \text{if } p = Send^{f^{j+1}(\alpha_{i_{j+1}})}, \\ Comp_p \cap {}_{\xi(k_p^{-1})}(\kappa_p^j \cup \{\text{Lett}^{f^{j+1}(\alpha_{i_{j+1}})}\}) & \text{if } p = \text{Resp}^{f^{j+1}(\alpha_{i_{j+1}})}. \end{cases}$

The intuition behind the above definition is as follows. The knowledge of a participant not participating in a protocol step is not changing. If a participant is the initiator of a step, then his knowledge is extended with the set of the generated nonces. If a participant is the responder of a step, then his knowledge is extended by all the letters, which can be retrieved from the former knowledge and the letter actually received. But, for efficiency reasons it is restricted to a subset of $Comp_p$, i.e., to the letters which the participant needs in order to compose any letter in any execution determined by $F$.

For the Intruder the knowledge is defined in a slightly different way.

Definition 3.11. For the Intruder the knowledge at each step $j$ of the protocol is common for all $p \in \bigcup_{f \in F} f(T_P) \cap P_e$ and it is given inductively as follows:

$\kappa_p^0 = I \cup \{k_p^{-1}, k_1\} \cup \{q \mid q \in P\} \cup \{n_1, \ldots, n_k\}$,

$\kappa_p^{j+1} = \begin{cases} \kappa_p^j & \text{if } \text{Resp}^{f^{j+1}(\alpha_{i_{j+1}})} \notin P_e, \\ \text{Comp}_p \cap {}_{\xi(k_p^{-1})}(\kappa_p^j \cup \{\text{Lett}^{f^{j+1}(\alpha_{i_{j+1}})}\}) & \text{if } \text{Res}^{f^{j+1}(\alpha_{i_{j+1}})} \in P_e. \end{cases}$

Some explanation about the above definition is in place. If the Intruder is not the responder of a letter, then his knowledge does not change. Otherwise, the Intruder is retrieving all the possible letters from his knowledge and the letter he has received (restricted to a subset of $\text{Comp}_p$ for efficiency reasons).

For simplicity, we assume that the Intruder does not generate his nonces as he can use them several times in many executions. This does not introduce any limitations.

In the following definition we formulate the conditions which guarantee that a sequence of protocol step interpretations is a computation of the protocol.

Definition 3.12. By a computation of the protocol $\Sigma$ we mean any finite sequence of protocol step interpretations: $\tau = (f^1(\alpha_{i_1}), f^2(\alpha_{i_2}), \ldots, f^k(\alpha_{i_k}))$ which meets the following conditions:

1. $(\forall k \in N_+)[i_k > 1 \Rightarrow (\exists j < k)(f^j = f^k \land i_j = i_k - 1)]$,
2. \((\forall k, j \in \mathbb{N}_+)[k \neq j \Rightarrow Gen^{f_k(\alpha_k)} \cap Gen^{f_j(\alpha_j)} = \emptyset]\),

3. \((\forall j \in \mathbb{N}_+)[Lett^{f_j(\alpha_j)} \in Comp^{f_{j-1}(\alpha_{j-1})} \cup Gen^{f_j(\alpha_j)}]\),

4. \(\forall n=1,...,k-1 Time^{f_n(\alpha_n)} < Time^{f_{n+1}(\alpha_{n+1})}\),

5. \(\forall n=1,...,k TConst^{f_n(\alpha_n)} = true\).

The first condition states that for each protocol step (except for the first one) in interpretation \(f\), there is a preceding step in the same interpretation. The second one says that the sets of generated nonces are disjoint, whereas the third one guarantees that the letter \(Lett(f_j(\alpha_j))\) can be sent by \(Send(f_j(\alpha_j))\) only if it can be composed from the set of currently generated nonces and the knowledge of the participant \(Send(f_j(\alpha_j))\) at the step \(j-1\). The next condition guarantees that the time is progressing between two successive steps of the execution. The last condition says that all time constraints for all interpretations hold true.

3.1. Attacks upon Protocols

Security protocols are used in order to establish a secure communication channel between two parties involved in the communication. This is obtained by ensuring that each party is confident about several security properties: the other party is who they say they are (authentication), a confidential information is not visible to non-authorised parties (secrecy), the information exchanged by two parties cannot be altered by an intruder (integrity), and finally the parties taking part in the transaction cannot deny it later (non-repudiation). In this paper we focus on checking authentication and secrecy only. We say that a given protocol is correct if it cannot be executed in such a way that identifiers or keys of one participant are used by someone else. Depending on the protocol and its goal, we verify authentication, secrecy, or a combination of both of them. Having this in mind, we give the following definition of an attack.

**Definition 3.13.** By an attacking execution we mean any execution under an interpretation \(f\), where \(f(P) = \iota(p)\), for some \(P \in T_P\) and \(p \in P\).

By an authentication attack upon a protocol we mean any of its computations such that an attacking execution is its subsequence. By a secrecy attack upon a protocol we mean any of its computations in which a secret information is a part of the knowledge of the intruder.

4. Networks of Communicating Timed Automata

In this section we represent the computations of a protocol by runs of a network of communicating timed automata, where each timed automaton represents one component of the protocol. Let \(C\) be a set of time constraints, defined in a similar way to Def. 2.3, but using the clocks \(\mathcal{X}\) and integers.

**Definition 4.1.** A timed automaton \((TA, for short)\) is a five-tuple \(A = (A, L, l^0, E, \mathcal{X})\), where

- \(A\) is a finite set of actions, where \(A \cap \mathcal{R}_+ = \emptyset\),
- \(L\) is a finite set of locations \((l^0 \in L\) is an initial location),
- \(\mathcal{X}\) is a finite set of clocks,
- \(E \subseteq L \times A \times C \times 2^\mathcal{X} \times L\) is a transition relation,
Each element $e$ of $E$ is denoted by $l \xrightarrow{a,cc,X} l'$, which represents a transition from the location $l$ to the location $l'$, executing the action $a$, with the set $X \subseteq \mathcal{X}$ of clocks to be reset, and with the clock constraint $cc \in C$ defining the enabling condition (guard) for $e$.

Given a transition $e := l \xrightarrow{a,cc,X} l'$, we write $\text{source}(e)$, $\text{target}(e)$, $\text{action}(e)$, $\text{guard}(e)$ and $\text{reset}(e)$ for $l$, $l'$, $a$, $cc$ and $X$, respectively. The clocks of a timed automaton allow to express the timing properties. An enabling condition constrains the execution of a transition without forcing it to be taken.

### 4.1. Semantics of Timed Automata

Let $\mathcal{A} = (A, L, l^0, E, \mathcal{X})$ be a timed automaton. A concrete state of $\mathcal{A}$ is defined as an ordered pair $(l, v)$, where $l \in L$ and $v \in \mathcal{R}^{0,\mathcal{X}}_c$ is a valuation.

The concrete state space of $\mathcal{A}$ is a transition system $C_c(\mathcal{A}) = (Q, s^0, \rightarrow_c)$, where

- $Q = L \times \mathcal{R}^{0,\mathcal{X}}_c$ is the set of all the concrete states,
- $s^0 = (l^0, v^0)$ with $v^0(x) = 0$ for all $x \in \mathcal{X}$ is the initial state, and
- $\rightarrow_c \subseteq Q \times (E \cup \mathcal{R}_+ \times Q)$ is the transition relation, defined by action- and time successors as follows:
  
  - for any $\delta \in \mathcal{R}_+$, $(l, v) \xrightarrow{\delta_c} (l, v + \delta)$ (time successor),
  - for $a \in A$, $(l, v) \xrightarrow{a_c} (l', v')$ iff $(\exists cc \in C)(\exists X \subseteq \mathcal{X})$ such that $l \xrightarrow{a,cc,X} l' \in E$, $v \in [cc]$ and $v' = v[X := 0]$ (action successor).

Intuitively, a time successor does not change the location $l$ of a concrete state, but it increases the clocks.

An action successor corresponding to an action $a$ is executed when the guard $cc$ holds for $v$.

For $(l, v) \in Q$ and $\delta \in \mathcal{R}_+$, let $(l, v) + \delta$ denote $(l, v + \delta)$. A $s_0$-run $\rho$ of $\mathcal{A}$ is a maximal sequence $\rho = s_0 \rightarrow_c s_0 + a_0 \rightarrow_c s_1 + \delta_1 \rightarrow_c s_1 + a_1 \rightarrow_c s_2 + \delta_2 \rightarrow_c \ldots$, where $a_i \in A$ and $\delta_i \in \mathcal{R}_{0+}$, for each $i \geq N$.

Now, we are going to use networks (sets) of timed automata for modelling executions of the protocol as well as for modelling the knowledge of the participants.

### 4.2. Product of a family of timed automata

A set of timed automata\(^9\) can be composed into a global (product) timed automaton as follows: the transitions of the timed automata that do not correspond to a shared action are interleaved, whereas the transitions labelled with a shared action are synchronised. There are many different definitions of a parallel composition. Our definition determines the multi-way synchronization, i.e., it requires that each component that contains a communication transition (labelled with a shared action) has to perform this action.

**Definition 4.2.** Let $\mathcal{I} = \{i_1, \ldots, i_{n_3}\}$ be a finite ordered set of indices, and $\mathcal{A} = \{A_i \mid i \in \mathcal{I}\}$, where $A_i = (A_i, L_i, l^0_i, E_i, \mathcal{X}_i)$, be a set (network) of timed automata indexed with $\mathcal{I}$. The automata in $\mathcal{A}$ are executed with the shared action.

\(^9\)This definition can be found in the survey [PP07].
called components. Let $A(a) = \{i \in I \mid a \in A_i\}$ be a set of the indices of the components containing the action $a \in \bigcup_{i \in I} A_i$. A composition (product) of the timed automaton $A_i \parallel \ldots \parallel A_{i_n}$ is a timed automaton $A = (A, L, l^0, E, X)$, where

- $A = \bigcup_{i \in I} A_i$, $L = \prod_{i \in I} L_i$, $l^0 = (l^0_1, \ldots, l^0_{i_n})$, $X = \bigcup_{i \in I} X_i$,

and the transition relation is given by

$$((l_i, \ldots, l_{i_n}), a, \bigwedge_{i \in A(a)} \exists c_i, \bigcup_{i \in A(a)} X_i, (l'_i, \ldots, l'_{i_n})) \in E \iff$$

$$\iff (\forall i \in A(a))(l_i, a, c_i, X_i, l'_i) \in E_i \land (\forall i \in I \setminus A(a)) l'_i = l_i.$$

### 4.3. Automata for modelling executions of the participants

Assume we are dealing with a protocol $\Sigma = (\alpha_1, \ldots, \alpha_n)$. Consider any partial interpretation $\overline{f}$. All the executions $f(\Sigma)$, where $f$ is associated with $\overline{f}$, are modelled by the timed automaton $A_{\overline{f}} = (\Sigma_{\overline{f}}, Q, s^f_0, \delta_{\overline{f}}, X_{\overline{f}})$, where:

- $\Sigma_{\overline{f}} = \{k_{f,i}^1 | 1 \leq i \leq n \land \text{Send}^f(\alpha_i) \in P\} \cup \bigcup_{i=1}^{n} \bigcup_{X \subseteq C} (\{s^f_i\} \land X \cap \text{Lett}^f(\alpha_i))$,

- $Q_{\overline{f}} = \{s^f_0, s^f_1, s^f_2, \ldots, s^f_n\}$ is the set of states, where $s^f_0$ is the initial state,

- $X_{\overline{f}} = \bigcup_{i=0,1,\ldots,n} \{z_r | \tau \in (T_{\tau} \cap \text{Gen}^f(\alpha_i))\}$,

- $\delta_{\overline{f}} = \{(s^f_{i-1}, k_{f,i}^1, Z(T_{\text{Constr}}(\alpha_i)), \{z_r | \tau \in (T_{\tau} \cap \text{Gen}^f(\alpha_i-1))\}, s^f_i) | 1 \leq i \leq n \land k_{f,i}^1 \in \Sigma_{\overline{f}}\} \cup \{(s^f_{i-1}, k_{f,i}^X, Z(T_{\text{Constr}}(\alpha_i)), \{z_r | \tau \in (T_{\tau} \cap \text{Gen}^f(\alpha_i-1))\}, s^f_i) | 1 \leq i \leq n \land k_{f,i}^X \in \Sigma_{\overline{f}}\}$.

Time constraints $Z(T_{\text{Constr}}(\alpha_i))$ are defined inductively as follows:

- $Z(\text{true}) = \text{true}$;
- $Z(\tau_l \leq l) = z_{\tau_p} \leq l$;
- $Z(T_{\text{Constr}1} \land T_{\text{Constr}2}) = Z(T_{\text{Constr}1}) \land Z(T_{\text{Constr}2})$.

The intuition behind the above definition is as follows. Each state $s^f_i$ of the automaton is reached after executing one of steps $f(\alpha_i)$ of the execution $f(\Sigma)$ for some $f$ associated with $\overline{f}$. Additionally the state $s^f_i$ can be reached iff $T_{\text{Constr}}(f(\alpha_i)) = \text{true}$ and all the clocks $\{z_r | \tau \in (T_{\tau} \cap \text{Gen}^f(\alpha_i-1))\}$ are reset.

If the sender of this step is honest, then there is only one possibility to execute this step as the sender needs to have the required knowledge for composing the letter sent in this step. However, if Intruder is the sender of this step, then there are many possibilities to execute this step determined by the sets of generators of the letter to be sent. Each of these cases is labelled with a different label $k_{f,i}^X$.

### 4.4. Automata for modelling knowledge of the participants

Consider a finite set of protocol partial interpretations $F$. For each honest participant $p \in (\bigcup_{f \in F} f(T_f) \setminus P_i)$ and each element $l \in Comp^p_{E_F} \setminus \kappa^0_p$, we define the following (knowledge) automaton $A^p_i = (\Sigma^p_i, Q^p_i, \delta^p_i, X^p_i, \lambda^p_i)$, where
If the automaton \( A \)

\[
\text{Cond}(q, k, T_{\text{Constr}}^{(\alpha)}, \{z_{\tau} | \tau \in (T_{\text{T}} \cap Gen^{(\alpha_1-1)})\}, s_{t}) \land
\]

\[(i) \quad (p = Send^{(\alpha)} \land l \in Gen^{(\alpha)}) \lor
\]

\[(ii) \quad (p = Resp^{(\alpha)} \land l \in \xi_{(k_{p}^{-1})}(\{\text{Lett}^{(\alpha)}\}) \land
\]

\[\land (p = Resp^{(\alpha_2)} \Rightarrow l \notin \xi_{(k_{p}^{-1})}(\text{Lett}^{(\alpha_2)})) \land
\]

\[\land (p = Send^{(\alpha)} \Rightarrow l \notin Gen^{(\alpha)})\]

\[\text{Cond}_{2}(k) := (\exists \tau \in F)(\exists \sigma \in \delta_{T})
\]

\[\sigma = (s_{t_{-1}}, k, T_{\text{Constr}}^{(\alpha)}, \{z_{\tau} | \tau \in (T_{\text{T}} \cap Gen^{(\alpha_1-1)})\}, s_{t}) \land
\]

\[(iii) \quad (p = Send^{(\alpha)} \land l \in \text{Comp}^{(\alpha)} \setminus Gen^{(\alpha)}) \lor
\]

\[(iv) \quad (p = Resp^{(\alpha)} \land l \in \xi_{(k_{p}^{-1})}(\{\text{Lett}^{(\alpha)}\}) \land
\]

\[\land (p = Resp^{(\alpha_2)} \Rightarrow l \notin \xi_{(k_{p}^{-1})}(\text{Lett}^{(\alpha_2)})) \land
\]

\[\land (p = Send^{(\alpha)} \Rightarrow l \notin Gen^{(\alpha)})\]

- \( Q^{p} = \{q^{p}_i, s^{p}_i\} \) is the set of states (\( q^{p}_i \) is the initial state),

- \( \delta^{p}_i \) is the transition relation given as follows

\[(q^{p}_i, k, \text{True}, \emptyset, s^{p}_i) \in \delta^{p}_i \text{ iff } \text{Cond}_{1}(k), (s^{p}_i, k, \text{True}, \emptyset, s^{p}_i) \in \delta^{p}_i \text{ iff } \text{Cond}_{2}(k).\]

If the automaton \( A^{p}_t \) is in the state \( q^{p}_t \), then this means that the participant \( p \) does not know \( l \). If the automaton \( A^{p}_t \) moves to the state \( s^{p}_t \), then this corresponds to the fact that \( p \) learns about \( l \) and can use it. The condition \( (i) \) specifies that \( l \) is generated by \( p \) at the step \( f^{(\alpha_i)} \). The condition \( (ii) \) says that \( p \) learns about \( l \) at the step \( f^{(\alpha_i)} \). This is modelled only once in order to reduce the number of the transitions. The condition \( (iii) \), which defines the loop, enables \( p \) to use \( l \) while composing new letters. The condition \( (iv) \) enables to receive \( l \) in a different execution that the one, which was used to define the condition \( (ii) \).

For the Intruder \( i \) and each letter \( l \in \text{Comp}_{T}^{0} \setminus \kappa_{i}^{0} \) we define the knowledge automaton \( A^{p}_{i} = (\Sigma^{i}, Q^{i}, q^{i}_t, \delta^{i}_t, X^{i}) \) in a similar way to the definition of \( A^{p}_t \). So, we have: \( Q^{i}_t = \{q^{i}_t, s^{i}_t\} \) is a set of the states, \( q^{i}_t \) is the initial state, \( \delta^{i}_t \) is the transition relation given as follows: \((q^{i}_t, k, \text{True}, \emptyset, s^{i}_t) \in \delta^{i}_t \text{ iff } \text{Cond}_{1}(k), (s^{i}_t, k, \text{True}, \emptyset, s^{i}_t) \in \delta^{i}_t \text{ iff } \text{Cond}_{2}(k)\). The differences w.r.t \( A^{p}_t \) are as follows: the condition \( (i) \) is omitted, in \((i, ii) \) we replace \( l = Send^{(\alpha)} \text{ with } Send^{(\alpha)}_{i} \) with \( p = Resp^{(\alpha)} \text{ with } Resp^{(\alpha)}_{i} \in P_{i} \) and \( p = Resp^{(\alpha)} \text{ with } Resp^{(\alpha)}_{i} \in P_{i}. \) Moreover, we replace \((iii) \) with \((Send^{(\alpha)}_{i} \in P_{i} \land (\exists X \subseteq L)(X \vdash \text{Lett}^{(\alpha)} \land \land l \in X \land k = k_{X}^{N})\). The new condition \((iii) \) enables \( i \) to use \( l \) while composing new letters.

Recall that we are dealing with the protocol \( \Sigma \) and a finite set \( F \) of its partial interpretations. Let \( \mathcal{A}_{\Sigma} = (N, Q, s^{0}, \delta, X) \) be the product automaton of the following set of the automata

\[\{A^{p}_{\tau} | \tau \in F\} \cup \{A^{p}_{x} | l \in \text{Comp}_{T_{\text{F}}}^{p} \}.\]
Consider the state space \( C_c(A_P) = (Q, s^0, \rightarrow_c) \) of the product automaton \( A_P \). Let 
\( (k_{F, i_1}, k_{F, i_2}, \ldots, k_{F, i_k}) \) be any sequence of actions from the set \( \bigcup_{T \in F} \Sigma_f \). By a run on the word 
\( (k_{F, i_1}, k_{F, i_2}, \ldots, k_{F, i_k}) \) in the concrete space \( C_c(A_P) \) we mean the following sequence:

\[
\rho = s_0 \overset{\delta_1}{\rightarrow} s_0 \overset{a_1}{\rightarrow} \ldots \overset{\delta_{k-1}}{\rightarrow} s_{k-1} \overset{a_k}{\rightarrow} s_k,
\]

where \( a_j = k_{F, i_j} \) for all \( j = 1, \ldots, k \), \( \delta_1 = Time^{f_1(\alpha_{i_1})} \) and \( \delta_j = Time^{f_j(\alpha_{i_j})} - Time^{f_{j-1}(\alpha_{i_{j-1}})} \) for all \( j = 2, \ldots, k \). Additionally if \( s_j = (s_j, v_j) \), then \( s_j \overset{a_j, c_{j}, A} \rightarrow s_{j+1} \in \delta \), where \( c_{j} = \Sigma(T \cap Gen^{f_j(\alpha_{i_j})}) \) and \( v_j \in [c_{j}] \) and \( v_{j+1} = v_j \{ z_{T} \mid \tau \in (T \cap Gen^{f_j(\alpha_{i_j})}) \} := 0 \).

The following theorem says that for each computation in the computation structure there is a corresponding run in the concrete state space \( C_c(A_P) \) built for this structure and moreover each run of \( C_c(A_P) \) corresponds to some computation.

**Theorem 4.1.** Let \( f^i \in F \) for \( 1 \leq i \leq k \). A sequence of protocol steps \( \tau = (f^1(\alpha_{i_1}), f^2(\alpha_{i_2}), \ldots, f^k(\alpha_{i_k})) \) is a computation iff there exists a run

\[
\rho = s_0 \overset{\delta_1}{\rightarrow} s_0 \overset{a_1}{\rightarrow} \ldots \overset{\delta_{k-1}}{\rightarrow} s_{k-1} \overset{a_k}{\rightarrow} s_k,
\]

in the space \( C_c(A_P) = (Q, s^0, \rightarrow_c) \) on the word \((a_1, a_2, \ldots, a_k)\), where:

\[
a_j \in \begin{cases} 
k_{F, i_j} & \text{if } Send^{f_j(\alpha_{i_j})} \in P, \\
k^{X, i_j} & \text{if } Send^{f_j(\alpha_{i_j})} \in P', 
\end{cases}
\]

\( \delta_1 = Time^{f_1(\alpha_{i_1})} \) and for all \( j = 2, \ldots, k \) \( \delta_j = Time^{f_j(\alpha_{i_j})} - Time^{f_{j-1}(\alpha_{i_{j-1}})} \).

Proof by a straightforward induction on the length of a computation (run).

Thanks to above theorem, we can reduce an analysis of a security protocol for interpretations assumed to verification of the corresponding product automaton. Specifically, there is an attack on the protocol iff there is a run in the product automaton corresponding to some attacking execution.

## 5. Experimental Results

We have tested susceptibility to authentication (e.g., reflection, Man-in-the-middle) and secrecy attacks for several timed security protocols: CCITT(1), WMF, Denning-Sacco, Kerberos, and the timed version of NSPK [16]. In our experiments we have tested reachability of states which correspond to the attack considered. The space considered consists of the two honest participants \( A \) and \( B \), the honest server \( S \) (if needed), and the Intruder \( I \). The experiments show that our method captures all the mentioned above types of attacks upon the protocols as well as reports correctness of flawless protocols like NSPK \(_{F, i_2}\).

In case of timed protocols, it is difficult to make a complete comparison with other approaches mentioned before. This is mainly due to the lack of available experimental results (see for example [5]) or a very limited number of sessions considered like in [4] or [11]. Therefore, we present all our results in Table 1 and only then we compare some results with those of TPMC [11] and SATMC [2] - a SAT-based model checker module of the state-of-the-art tool AVISPA [1].
Since SATMC supports verification of untimed security protocols only, for comparing SATMC with VerICS we consider untimed versions of the timed protocols verified with our method. The computer used to perform the experiments was equipped with the processor Intel Pentium D (3000 MHz), 2 GB main memory, the operating system Linux, and the SAT-solver MiniSat. For each protocol we give the number of sessions of the test, the number of clauses in the SAT formula, the total memory consumed, and the total time of generating and solving the formula by the SAT-solver. The comparison shows that our tool correctly reports the expected attacks.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Sessions</th>
<th>Clauses</th>
<th>Memory (MB)</th>
<th>Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSPK</td>
<td>3</td>
<td>15124</td>
<td>2.56</td>
<td>0.04</td>
</tr>
<tr>
<td>NSPK_Fix</td>
<td>3</td>
<td>16496</td>
<td>2.72</td>
<td>0.30</td>
</tr>
<tr>
<td>NSPK_Time</td>
<td>3</td>
<td>52816</td>
<td>4.60</td>
<td>1.78</td>
</tr>
<tr>
<td>NSPK_Time,Fix</td>
<td>6</td>
<td>1187363</td>
<td>134</td>
<td>87</td>
</tr>
<tr>
<td>NSPK_Time,Fix</td>
<td>6</td>
<td>924062</td>
<td>509</td>
<td>6123</td>
</tr>
<tr>
<td>WMF</td>
<td>3</td>
<td>14596</td>
<td>2.47</td>
<td>0.05</td>
</tr>
<tr>
<td>WMF</td>
<td>9</td>
<td>439310</td>
<td>22.3</td>
<td>1.62</td>
</tr>
<tr>
<td>CCITT (1)</td>
<td>3</td>
<td>5146</td>
<td>2.05</td>
<td>0.02</td>
</tr>
<tr>
<td>CCITT (1)</td>
<td>9</td>
<td>388777</td>
<td>18.74</td>
<td>1.10</td>
</tr>
<tr>
<td>Denning-Sacco</td>
<td>3</td>
<td>12440</td>
<td>2.33</td>
<td>0.06</td>
</tr>
<tr>
<td>Denning-Sacco</td>
<td>9</td>
<td>475391</td>
<td>24.57</td>
<td>1.75</td>
</tr>
<tr>
<td>Kerberos</td>
<td>3</td>
<td>38119</td>
<td>3.79</td>
<td>0.26</td>
</tr>
<tr>
<td>Kerberos</td>
<td>9</td>
<td>567128</td>
<td>34.71</td>
<td>3.46</td>
</tr>
</tbody>
</table>

Table 1. Experimental results

The meaning of symbols: Clauses - the number of clauses in the SAT formula, Memory - the memory consumed, Time - the time of verification.

The available times of verification with SATMC (linear encoding) are as follows: for NSPK (3 sessions) - 0.10 sec., and for Kerberos (3 sessions) - 6.24 sec. In case of TPMC, the times available are for NSPK (3 sessions) - 0.86 sec., NSPK\_Fix (3 sessions) - 0.10 sec., and Kerberos (3 sessions) - 6.24 sec. So, VerICS is faster than SATMC and TPMC for most of the protocols verified. Notice that our method often allows for verifying protocol runs composed even of 9 sessions, which is quite seldom for the available tools based on model checking.

6. Conclusions and Perspectives

In this paper we proposed a novel method for verifying secrecy and authentication of timed security protocols. The method consists in modelling a finite number of participants (including the intruder) and their knowledge about secrets, by a network of timed automata. Due to a very distributed representation our approach seems to be quite efficient. Our experimental results also look very promising in comparison with SATMC (linear encoding) and TPMC. The next step is to extend our model to all kinds of attacks and to investigate the computation limits of our method in terms of the number of sessions covered. The authors wish to thank Paweł Dudek and Andrzej Zbrzezny for their help in implementing our approach.
References


