Verifying Timed Security Protocols via Translation to Timed Automata*

Mirosław Kurkowski[1] and Wojciech Penczek[2]

[1] Institute of Mathematics and Computer Science, Jan Dlugosz University,
m.kurkowski@ajd.czest.pl
[2] Institute of Computer Science, PAS,
Institute of Informatics, University of Podlasie
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 the 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 VerICS.

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, 10] as well as of time dependent ones [4–8, 11]. 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 the standard model checking methods for communicating protocols, 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.

This paper extends our former verification results [10] from untimed 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. The main contribution consists in using networks of timed automata for modelling separately the behaviour of the participants and their knowledge about secrets. Thanks to that we develop a very distributed representation of the protocol executions, which is crucial for an efficient symbolic encoding and model checking. This is possible thanks to

* This research is partially supported by the Ministry of Science and Higher Education under grant 3 T11C 011 28 and TINFO: 55/E-82/BWSN-0122/2008.

the blocking multi-synchronization mechanism supported by the verification tool VERICS [13], the SAT-based BMC module of which is used for receiving the experimental results.

Related Work. There are several approaches to model checking based on timed automata. Our approach is close to the work by Corin at al. [4], where security protocols are directly modelled in terms of networks of timed automata and verified with UppAal [3]. The authors of [4] address timeouts and retransmissions, but do not show how one can model timestamps in such an approach. The approach of [8] also uses timed automata, but indirectly: timed security protocols modelled in the higher-level language *IL* are translated to timed automata. Another approach is taken in [12], where the tool TPMC, implementing a translation from Timed HLPSL to timed automata, is presented.

Outline. In Section 2 we introduce syntax for dealing with timed security protocols. A computational structure is given in Section 3. Section 4 defines a network of automata for representing the participants of a protocol and their knowledge.The experimental results and conclusions are given in Section 5.

2 Syntax of Timed Security Protocols

We start with sketching the syntax for dealing with timed security protocols. A more detailed presentation can be found in [9]. By $\mathcal{R}_+$ we denote the set of positive real numbers while $\mathcal{R}_* = \mathcal{R}_+ \cup \{0\}$ and by 2^Z_{fin} - a set of all the finite subsets of a set Z. We use the following basic sets in our model: $\mathcal{T}_P$ - the symbols representing the users of the computer network, $\mathcal{T}_I$ - the symbols representing the identifiers of the users, $\mathcal{T}_K$ - the symbols representing the cryptographic keys of the users, $\mathcal{T}_N$ - the symbols representing the users' *nonces*, $\mathcal{T}_T$ - the symbols representing the users' *time tickets*, $\mathcal{T}_L$ - the symbols representing lifetimes, $\mathcal{T}_R$ - the real (time) variables representing the times of sending the messages.

Definition 1. *By a set of* letter (terms) $\mathcal{T}$ *we mean the smallest set satisfying the following conditions:*
1. $\mathcal{T}_P \cup \mathcal{T}_I \cup \mathcal{T}_K \cup \mathcal{T}_N \cup \mathcal{T}_T \cup \mathcal{T}_L \subseteq \mathcal{T}$.
2. If $X \in \mathcal{T}$ *and* $Y \in \mathcal{T}$*, then the concatenation* $X \cdot Y \in \mathcal{T}$*,*
3. If $X \in \mathcal{T}$ *and* $\mathcal{K} \in \mathcal{T}_\mathcal{K}$*, then* $\langle X \rangle_\mathcal{K} \in \mathcal{T}$[3].
4. If $X \in \mathcal{T}$*, then the hash value of the letter term* X $h(X) \in \mathcal{T}$*,*

Next, for any finite subset $\mathcal{X} \subseteq \mathcal{T}$, the set $Comp(\mathcal{X})$ is composed of all the letters that can be constructed out of elements of $\mathcal{X}$ only[4]. In oder to deal with timed protocols, we define the set of time constraints.

Definition 2. *The set of time constraints* $\mathcal{C}$ *is given by the following grammar:*

$$\mathfrak{tc} ::= true \mid \tau_i - \tau_j \leq \mathcal{L}_\mathcal{F} \mid \mathfrak{tc} \wedge \mathfrak{tc}, \text{ where } \tau_i \in \mathcal{T}_R, \tau_j \in \mathcal{T}_T; \mathcal{L}_\mathcal{F} \in \mathcal{T}_L.$$

[3] $\langle X \rangle_\mathcal{K}$ is a term that is interpreted as a ciphertext containing the letter X encrypted with the key $\mathcal{K}$.
[4] Decription is not allowed here.

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 more complicated than in the common language as it provides an extra information to point out to additional actions of the sender like generating new secrets or composing a letter.

Definition 3. *By a* (protocol) step α *we mean a pair* (α^1, α^2)*, where* α^1 *is a triple* $(\mathcal{P}, \mathcal{Q}, \mathcal{L}) \in \mathcal{T}_\mathcal{P} \times \mathcal{T}_\mathcal{P} \times \mathcal{T}$*, and* α^2 *is a 4-tuple* $(\tau, \mathcal{X}, \mathcal{G}, \mathfrak{tc}) \in \mathcal{T}_R \times 2^{\mathcal{T}}_{fin} \times 2^{\mathcal{T}_K \cup \mathcal{T}_N \cup \mathcal{T}_T}_{fin} \times \mathcal{C}$ *with the following intuitive meaning:* $\mathcal{P}$ *- the sender,* $\mathcal{Q}$ *- the receiver,* $\mathcal{L}$ *- the letter sent from* $\mathcal{P}$ *to* $\mathcal{Q}$*,* τ *- the time of the step execution*[5]*,* $\mathcal{X}$ *- the set of letters necessary to compose* $\mathcal{L}$*,* $\mathcal{G}$ *- the set of generated secrets necessary to compose* $\mathcal{L}$*,* $\mathfrak{tc}$ *- the step time constraint, satisfying the conditions:*
1. $\mathcal{P} \neq \mathcal{Q}$ *(no sending to itself),*
2. $\mathcal{L} \in Comp(\mathcal{X}) \wedge (\forall \mathcal{Y} \subseteq \mathcal{X})\ (\mathcal{L} \in Comp(\mathcal{Y}) \Rightarrow \mathcal{Y} = \mathcal{X})$*,*
($\mathcal{X}$ *is a minimal set from which* $\mathcal{L}$ *can be constructed)*
3. $\mathcal{G} \subseteq \mathcal{X}$ *(the secrets of* $\mathcal{G}$ *are elements of* $\mathcal{X}$*).*
By a protocol Σ *we mean a finite sequence* $(\alpha_1, \ldots, \alpha_n)$ *of* steps.

Example 1. Consider Wide Mouth Frog Protocol (WMF) as a simple working example. The syntax of WMF is as follows. $\mathcal{T}_P = \{\mathcal{A}, \mathcal{B}, \mathcal{S}\}$, $\mathcal{T}_I = \{\mathcal{I}_\mathcal{A}, \mathcal{I}_\mathcal{B}\}$, $\mathcal{T}_K = \{\mathcal{K}_{\mathcal{AS}}, \mathcal{K}_{\mathcal{BS}}, \mathcal{K}_{\mathcal{AB}}\}$, $\mathcal{T}_T = \{\tau_\mathcal{A}, \tau_\mathcal{S}\}$, $\mathcal{T}_T = \emptyset$, $\mathcal{T}_L = \{\mathcal{L}_\mathcal{F}\}$, and $\mathcal{T}_T = \{\tau_1, \tau_2\}$. WMF is given by the following sequence of steps: (α_1, α_2), where:
$\alpha_1 = (\alpha_1^1, \alpha_1^2)$, $\alpha_1^1 = \ (\mathcal{A};\ \mathcal{S};\ \ \mathcal{I}_\mathcal{A}, \langle \tau_\mathcal{A}, \mathcal{I}_\mathcal{B}, \mathcal{K}_{\mathcal{AB}} \rangle_{\mathcal{K}_{\mathcal{AS}}})$,
$\alpha_1^2 = (\tau_1; \{\mathcal{I}_\mathcal{A}, \mathcal{I}_\mathcal{B}, \tau_\mathcal{A}, \mathcal{K}_{\mathcal{AB}}, \mathcal{K}_{\mathcal{AS}}\}; \{\tau_\mathcal{A}, \mathcal{K}_{\mathcal{AB}}\}; \tau_1 - \tau_\mathcal{A} \leq \mathcal{L}_\mathcal{F})$,
$\alpha_2 = (\alpha_2^1, \alpha_2^2)$, $\alpha_2^1 = \ (\mathcal{S};\ \mathcal{B};\ \ \langle \tau_\mathcal{S}, \mathcal{I}_\mathcal{A}, \mathcal{K}_{\mathcal{AB}} \rangle_{\mathcal{K}_{\mathcal{BS}}})$.
$\alpha_2 = \ (\tau_2; \{\mathcal{I}_\mathcal{A}, \tau_\mathcal{S}, \mathcal{K}_{\mathcal{AB}}, \mathcal{K}_{\mathcal{BS}}\}; \{\tau_\mathcal{S}\}; \tau_2 - \tau_\mathcal{A} \leq \mathcal{L}_\mathcal{F}\ \wedge\ \tau_2 - \tau_\mathcal{S} \leq \mathcal{L}_\mathcal{F})$.
The time constraint $\tau_2 - \tau_\mathcal{A} \leq \mathcal{L}_\mathcal{F}$ says that S can send (B - receive) the message only under the condition that the difference between time τ_2 of sending/receiving the message and time $\tau_\mathcal{A}$ of generating the ticket is less than $\mathcal{L}_\mathcal{F}$.

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: $\mathbf{P}$- the honest participants in the network, $\mathbf{P}_\iota$- the dishonest participants, $\mathbf{I}$- the identifiers of the participants in the network, $\mathbf{K}$- the cryptographic keys of the participants, $\mathbf{N}$- the *nonces*, $\mathbf{T}$- the users' *time tickets*, $\mathbf{\Gamma}$- the *lifetimes*.

Definition 4. *By a set of* letters $\mathbf{L}$ *we mean the smallest set satisfying the following conditions:*
1. $\mathbf{P} \cup \mathbf{P}_\iota \cup \mathbf{I} \cup \mathbf{K} \cup \mathbf{N} \cup \mathbf{T} \cup \mathbf{\Gamma}\ \subseteq \mathbf{L}$*,*

[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.

2. If $x, y \in \mathbf{L}$, then the concatenation $x \cdot y \in \mathbf{L}$,
3. If $x \in \mathbf{L}$ and $k \in \mathbf{K}$, then $\langle x \rangle_k \in \mathbf{L}$, $\langle x \rangle_k$ is a ciphertext consisting of the letter x encrypted with the key k.
4. If $x \in \mathbf{L}$, then $h(x) \in \mathbf{L}$, $h(x)$ is a hash value of the letter x.

Next, for any finite $X \subseteq \mathbf{L}$ define the set $Comp(X)$, which consists of all the letters that can be composed out of elements of X only[6] and the set $Sublet(X)$, which contains all the subletters of X.

Definition 5. *Let $X \subseteq \mathbf{L}$ and $K \subseteq \mathbf{K}$. Define the set $\xi_K(X) \subseteq \mathbf{L}$ 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 $\langle l \rangle_k \in \xi_K(X)$ 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 of $\mathcal{T}$, which are used for defining the runs of protocols.

Definition 6. *By a* partial interpretation *of the set of the letter terms $\mathcal{T}$ we mean any injection $\overline{f} : \mathcal{T} \to \mathbf{L}$ satisfying the following conditions:*
1. $\overline{f}(\mathcal{T}_P) \subseteq \mathbf{P} \cup \mathbf{P}_\iota$, $\overline{f}(\mathcal{T}_I) \subseteq \mathbf{I}$, $\overline{f}(\mathcal{T}_K) \subseteq \mathbf{K}$, $\overline{f}(\mathcal{T}_N) \subseteq \mathbf{N}$, $\overline{f}(\mathcal{T}_L) \subseteq \mathbf{\Gamma}$, $\overline{f}(\mathcal{T}_T) \subseteq \mathbf{T}$,
2. $(\forall X, Y \in \mathcal{T})\ \overline{f}(X \cdot Y) = \overline{f}(X) \cdot \overline{f}(Y)$ (homomorphism),
3. $(\forall X \in \mathcal{T})(\forall \mathcal{K} \in \mathcal{T}_K)\ \overline{f}(\langle X \rangle_{\mathcal{K}}) = \langle \overline{f}(X) \rangle_{\overline{f}(\mathcal{K})}$ (homomorphism),
4. $(\forall X \in \mathcal{T})\ \overline{f}(h(X)) = h(\overline{f}(X))$ (homomorphism),
5. If $\overline{f}(\mathcal{P}) = p$ for $p \in \mathbf{P}$, then $\overline{f}(\mathcal{I}_\mathcal{P}) = i_p$, $\overline{f}(\mathcal{N}_\mathcal{P}) \in \{n_p^1, \ldots, n_p^{k_S}\}$, $\overline{f}(\mathcal{K}_\mathcal{P}) = k_p$ and $\overline{f}(\mathcal{K}_\mathcal{P}^{-1}) = k_p^{-1}$.
6. If $\overline{f}(\mathcal{P}) = \iota$, then $\overline{f}(\mathcal{I}_\mathcal{P}) = i_\iota$, $\overline{f}(\mathcal{K}_\mathcal{P}) = k_\iota$ and $\overline{f}(\mathcal{K}_\mathcal{P}^{-1}) = k_\iota^{-1}$.
7. If $\overline{f}(\mathcal{P}) = \iota(p)$, then $\overline{f}(\mathcal{I}_\mathcal{P}) = i_p$, $\overline{f}(\mathcal{K}_\mathcal{P}) = k_p$ and $\overline{f}(\mathcal{K}_\mathcal{P}^{-1}) = k_p^{-1}$,
8. $\overline{f}(\mathcal{T}_P) \setminus \mathbf{P}_\iota \neq \emptyset$.

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 homomorfical 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 ι 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 ι impersonates another participant p in some interpretation, then in any execution under this interpretation p's keys and p's identifier need to be used by ι. 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.

[6] Decription is not allowed here.

Definition 7. *For a given partial interpretation $\overline{f}$ by an* interpretation *(associated with $\overline{f}$) of the set of the letter terms $\mathcal{T}$ and the set of time variables $\mathcal{T}_R$ we mean any injection $f : \mathcal{T} \cup \mathcal{T}_R \to \mathbf{L} \cup \mathcal{R}_*$ such that $f \mid_{\mathcal{T} \setminus \mathcal{T}_T} = \overline{f}$ and $f(\mathcal{T}_T \cup \mathcal{T}_R) \subseteq \mathcal{R}_*$.*

The time tickets and the times of sending/receiving of messages are finally mapped into time moments represented by positive 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 8. *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. $X \subseteq Sublet(\{l\})$,
2. $l \in Comp(X)$,
3. $(\forall m \in X)(m \notin Comp(X \setminus \{m\})$,
4. $(\forall m \in X)(l \notin Comp(X \setminus \{m\})$.

Intuitively, $X \vdash l$ if all the elements of X are subletters of l, l can be composed out of the elements of X, and X is a minimal such a set. We extend an interpretation f to the time constraints of $\mathcal{C}$, which is defined inductively as follows:
– $f(true) := true$,
– $f(\tau_i - \tau_j \leq \mathcal{L}_{\mathcal{F}}) := f(\tau_i) - f(\tau_j) \leq f(\mathcal{L}_{\mathcal{F}})$,
– $f(\mathfrak{tc}_1 \wedge \mathfrak{tc}_2) := f(\mathfrak{tc}_1) \wedge f(\mathfrak{tc}_2)$.

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 9. *Consider a step $\alpha = (\alpha^1, \alpha^2) = ((\mathcal{P}, \mathcal{Q}, \mathcal{L}), (\tau_\alpha, \mathcal{X}, \mathcal{G}, \mathfrak{tc}))$ of a given protocol Σ and an interpretation f of $\mathcal{T} \cup \mathcal{T}_R$, which satisfies the following condition $\bigwedge_{\tau \in \mathcal{G} \cap \mathcal{T}_T}(f(\tau) = f(\tau_\alpha))$, 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 α (denoted by $f(\alpha)$) we mean the following tuple:*
– $((f(\mathcal{P}), f(\mathcal{Q}), f(\mathcal{L})), (f(\tau_\alpha), f(\mathcal{X}), f(\mathcal{G}), f(\mathfrak{tc})))$, *if* $f(\mathcal{P}) \in \mathbf{P}$,
– $((f(\mathcal{P}), f(\mathcal{Q}), f(\mathcal{L})), (f(\tau_\alpha), \{X \mid X \vdash f(\mathcal{L})\}, \emptyset, f(\mathfrak{tc})))$, *if* $f(\mathcal{P}) \in \mathbf{P}_\iota$.

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 $f(\alpha_i) = ((p, q, l), (\tau, X, G, \mathfrak{c}))$, for some $p, q \in \mathbf{P} \cup \mathbf{P}_\iota$, $X \in 2^{\mathbf{L}}_{fin}$, $G \in 2^{\mathbf{K} \cup \mathbf{N}}_{fin}$, and $l \in \mathbf{L}$, then we use the following notations: $Send^{f(\alpha_i)} = p$, $Lett^{f(\alpha_i)} = l$, $Gen^{f(\alpha_i)} = G$, $Resp^{f(\alpha_i)} = q$, $Part^{f(\alpha_i)} = \{Send^{f(\alpha_i)}, Resp^{f(\alpha_i)}\}$, $Time^{f(\alpha_i)} = \tau$, and $TConstr^{f(\alpha_i)} = \mathfrak{c}$.
In addition if $Send^{f(\alpha_i)} \in \mathbf{P}$, then let $Comp^{f(\alpha_i)} = X$ (the set of letters that are sufficient to compose $Lett^{f(\alpha_i)}$) and if $Send^{f(\alpha_i)} \in \mathbf{P}_\iota$, then let $Comp^{f(\alpha_i)} = \bigcup_{X \vdash Lett^{f(\alpha_i)}} X$ (the union of sets generating $Lett^{f(\alpha_i)}$). Similarly, for a partial interpretation $\overline{f}$ the interpretation f is associated with, we use notations: $Send^{\overline{f}(\alpha_i)}, Lett^{\overline{f}(\alpha_i)}, Gen^{\overline{f}(\alpha_i)}, Resp^{f(\alpha_i)}$ for $\overline{f}(\mathcal{P}), \overline{f}(\mathcal{L})), \overline{f}(\mathcal{G}), \overline{f}(\mathcal{Q})$, respectively.

Definition 10. *Let f be an interpretation satisfying the following two 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 $\bigwedge_{i=1,\ldots,n} TConstr^{f(\alpha_i)} \equiv true$. Then, by* the f-execution of a protocol Σ *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 $Comp^p_{\mathcal{F}}$ ($Comp^\iota_{\mathcal{F}}$) of the letters, which the participant $p \in \bigcup_{f\in\mathcal{F}} f(\mathcal{T}_P) \setminus \mathbf{P}_\iota$ (Intruder ι, resp.) needs in order to compose all the letters sent in an execution under any interpretation $f \in \mathcal{F}$.

Definition 11. *The set $Comp^p_{\mathcal{F}} = \bigcup_{1\le i\le n} \bigcup_{\{f\in\mathcal{F} | Send^{f(\alpha_i)}=p\}} Comp^{f(\alpha_i)}$ for an honest user p is the union of all the sets $Comp^{f(\alpha_i)}$ for all $i \le n$ and $f \in \mathcal{F}$, where $Send^{f(\alpha_i)} = p$. The set $Comp^\iota_{\mathcal{F}} = \bigcup_{1\le i\le n} \bigcup_{\{f\in\mathcal{F} | Send^{f(\alpha_i)}\in\mathbf{P}_\iota\}} Comp^{f(\alpha_i)}$ is the union of all the sets $Comp^{f(\alpha_i)}$ for all $i \le n$ and $f \in \mathcal{F}$, where $Send^{f(\alpha_i)} \in \mathbf{P}_\iota$.*

Consider any finite sequence of interpretations of k protocol steps $\mathfrak{r} = (f^1(\alpha_{i_1}), f^2(\alpha_{i_2}), \ldots, f^k(\alpha_{i_k}))$. For every $p \in \bigcup_{i=1}^k f^i(\mathcal{T}_P)$ we define a sequence of the participant's knowledge $(\kappa^j_p)_{j=1,\ldots,k}$ at the steps of the protocol.

Definition 12. *For an honest user $p \in \bigcup_{f\in\mathcal{F}} f(\mathcal{T}_P) \setminus \mathbf{P}_\iota$ his knowledge at the step j is given inductively as follows:*

$$\kappa^0_p = \mathbf{I} \cup \{k_p^{-1}\} \cup \{k_q \mid q \in \mathbf{P}\} \cup \{k_\iota\},$$

$$\kappa^{j+1}_p = \begin{cases} \kappa^j_p & \text{if } p \notin Part^{f^{j+1}(\alpha_{i_{j+1}})}, \\ \kappa^j_p \cup Gen^{f^{j+1}(\alpha_{i_{j+1}})} & \text{if } p = Send^{f^{j+1}(\alpha_{i_{j+1}})}, \\ Comp^p_{\mathcal{F}} \cap \xi_{\{k_p^{-1}\}}(\kappa^j_p \cup \{Lett^{f^{j+1}(\alpha_{i_{j+1}})}\}) & \text{if } p = Resp^{f^{j+1}(\alpha_{i_{j+1}})}. \end{cases}$$

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 $Comp^p_{\mathcal{F}}$, i.e., to the letters which the participant needs in order to compose any letter in any execution determined by $\mathcal{F}$.

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

Definition 13. *For the Intruder the knowledge at each step j of the protocol is common for all $p \in \bigcup_{f\in\mathcal{F}} f(\mathcal{T}_P) \cap \mathbf{P}_\iota$ and it is given inductively as follows:*

$$\kappa^0_\iota = \mathbf{I} \cup \{k_\iota^{-1}, k_\iota\} \cup \{k_q \mid q \in \mathbf{P}\} \cup \{n^1_\iota, \ldots, n^{k_N}_\iota\},$$

$$\kappa^{j+1}_\iota = \begin{cases} \kappa^j_\iota & \text{if } Resp^{f^{j+1}(\alpha_{i_{j+1}})} \notin P_\iota, \\ Comp^\iota_{\mathcal{F}} \cap \xi_{\{k_\iota^{-1}\}}(\kappa^j_\iota \cup \{Lett^{f^{j+1}(\alpha_{i_{j+1}})}\}) & \text{if } Resp^{f^{j+1}(\alpha_{i_{j+1}})} \in P_\iota. \end{cases}$$

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 $Comp^{\iota}_{\mathcal{F}}$ 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 guaranteeing that a sequence of protocol step interpretations is a computation of the protocol.

Definition 14. *By a computation of the protocol Σ we mean any finite sequence of protocol step interpretations:* $\mathfrak{r} = (f^1(\alpha_{i_1}), f^2(\alpha_{i_2}), \ldots, f^k(\alpha_{i_k}))$ *which meets the following conditions:*

1. $(\forall k \in \mathcal{N}_+)[i_k > 1 \Rightarrow (\exists j < k)(f^j = f^k \wedge i_j = i_k - 1)]$,
2. $(\forall k, j \in \mathcal{N}_+)[k \neq j \Rightarrow Gen^{f^k(\alpha_{i_k})} \cap Gen^{f^j(\alpha_{i_j})} = \emptyset]$,
3. $(\forall j \in \mathbf{N}_+)[Lett^{f^j(\alpha_{i_j})} \in Comp(\kappa^{j-1}_{Send^{f^j(\alpha_{i_j})}} \cup Gen^{f^j(\alpha_{i_j})})]$,
4. $\forall_{n=1,\ldots,k-1}(Time^{f^n(\alpha_{i_n})} < Time^{f^{n+1}(\alpha_{i_{n+1}})})$,
5. $\forall_{n=1,\ldots,k} TConstr^{f^n(\alpha_{i_n})} = true$.

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 $Lett(f^j(\alpha_{i_j}))$ can be sent by $Send(f^j(\alpha_{i_j}))$ only if it can be composed from the set of currently generated nonces and the knowledge of the participant $Send(f^j(\alpha_{i_j}))$ 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.

Security protocols are used in order to establish a secure communication channel between two parties involved in the communication. Here we focus on checking authentication and secrecy only. We call a given protocol *correct* if it cannot be executed such that identifiers or keys of one participant are used by someone else. Having this in mind, we give the following definition of an attack.

Definition 15. *By* an attacking execution *we mean any execution under an interpretation f, where $f(\mathcal{P}) = \iota(p)$, for some $\mathcal{P} \in \mathcal{T}_P$ and $p \in \mathbf{P}$.*

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 $\mathcal{C}$ be a set of time constraints, defined in a similar way to Def. 2, but using the clocks $\mathcal{X}$ and integers.

Definition 16. *A* timed automaton *(TA, for short) is a five-tuple* $\mathcal{A} = (A, L, l^0, E, \mathcal{X})$*, 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,*
– $E \subseteq L \times A \times \mathcal{C} \times 2^{\mathcal{X}} \times L$ *is a* transition relation*,*
Each element e *of* E *is denoted by* $l \xrightarrow{a,\mathfrak{cc},X} l'$*, 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* $\mathfrak{cc} \in \mathcal{C}$ *defining the enabling condition (guard) for* e*.*

Given a transition $e := l \xrightarrow{a,\mathfrak{cc},X} l'$, we write $source(e)$, $target(e)$, $action(e)$, $guard(e)$ and $reset(e)$ for l, l', a, $\mathfrak{cc}$ 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.

A global state (l, v) of a timed automaton is composed of a location l and a valuation of the clocks v. A timed automaton can execute two types of transitions: either an action successor $((l, v) \xrightarrow{a}_c (l', v'))$ over an action a (if the guard and the invariants are satisfied) or a timed successor $((l, v) \xrightarrow{\delta}_c (l, v + \delta))$, which cannot invalidate guards and satisfies the invariants. For a global state (l, v) and $\delta \in \mathcal{R}_*$, let $(l, v) + \delta$ denote $(l, v + \delta)$. A $\mathfrak{s}_0$-*run* ρ of $\mathcal{A}$ is a maximal sequence

$$\rho = \mathfrak{s}_0 \xrightarrow{\delta_0}_c \mathfrak{s}_0 + \delta_0 \xrightarrow{a_0}_c \mathfrak{s}_1 \xrightarrow{\delta_1}_c \mathfrak{s}_1 + \delta_1 \xrightarrow{a_1}_c \mathfrak{s}_2 \xrightarrow{\delta_2}_c \ldots,$$

where $a_i \in A$ and $\delta_i \in \mathcal{R}_{0+}$, for each $i \geq \mathcal{N}$.

We use networks (sets) of timed automata for modelling executions of the protocol as well as for modelling the knowledge of the participants. The product of a network of timed automata is also a timed automata. Because of the lack of space we omit here the definition of formal semantics of a timed automaton and the product timed automaton of a network of timed automata. These definitions can be found in the survey [PP07][7] as well as in [9].

4.1 Automata for executions and knowledge 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 $A_{\overline{f}} = (\Sigma_{\overline{f}}, Q_{\overline{f}}, s_0^{\overline{f}}, \delta_{\overline{f}}, \mathcal{X}_{\overline{f}})$, where:
- $\Sigma_{\overline{f}} = \{k_{\overline{f},i} \mid 1 \leq i \leq n \ \wedge \ Send^{f(\alpha_i)} \in \mathbf{P}\} \cup$
  $\bigcup_{i=1}^{n} \bigcup_{X \subseteq \mathcal{L}} \{k_{\overline{f},i}^{X}\} \mid Send^{f(\alpha_i)} \in \mathbf{P}_\iota \wedge X \vdash Lett^{f(\alpha_i)}$,
- $Q_{\overline{f}} = \{s_0^{\overline{f}}, s_1^{\overline{f}}, s_2^{\overline{f}}, \ldots, s_n^{\overline{f}}\}$ is the set of states, where $s_0^{\overline{f}}$ is the initial state,
- $\mathcal{X}_{\overline{f}} = \bigcup_{i=0,1,\ldots,n} \{z_\tau \mid \tau \in (\mathcal{T}_T \cap Gen^{f(\alpha_i)})\}$,

[7] W. Penczek and A. Półrola, Advances in Verification of Time Petri Nets and Timed Automata, Springer, 2007.

- $\delta_{\overline{f}} = \{(s^{\overline{f}}_{i-1}, k_{\overline{f},i}, Z(TConstr^{\overline{f}(\alpha_i)}), \{z_\tau \mid \tau \in (\mathbf{T} \cap Gen^{f(\alpha_{i-1})})\}, s^{\overline{f}}_i) \mid 1 \leq i \leq n \wedge k_{\overline{f},i} \in \Sigma_{\overline{f}}\} \cup \{(s^{\overline{f}}_{i-1}, k^X_{\overline{f},i}, Z(TConstr^{\overline{f}(\alpha_i)}), \{z_\tau \mid \tau \in (\mathcal{T}_T \cap Gen^{f(\alpha_{i-1})})\}, s^{\overline{f}}_i) \mid 1 \leq i \leq n \wedge k^X_{\overline{f},i} \in \Sigma_{\overline{f}}\}$,

The time constraints $Z(TConstr^{\overline{f}(\alpha_i)})$ are defined inductively as follows:
– $Z(true) = true$;
– $Z(\tau_i - \tau_P \leq l) = z_{\tau_P} \leq l$;
– $Z(TConstr_1 \wedge TConstr_2) = Z(TConstr_1) \wedge Z(TConstr_2)$.
In the above definition each state $s^{\overline{f}}_i$ of the automaton is reached after executing one of the steps $f(\alpha_i)$ of the execution $f(\Sigma)$ for some f associated with $\overline{f}$. Additionally the state $s^{\overline{f}}_i$ can be reached iff $TConstr^{f(\alpha_i)} = true$ and all the clocks $\{z_\tau \mid \tau \in (\mathcal{T}_T \cap Gen^{\overline{f}(\alpha_{i-1})})\}$ 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 $k^X_{\overline{f},i}$.

Consider a finite set of protocol partial interpretations $\mathcal{F}$. For each honest participant $p \in (\bigcup_{\overline{f} \in \mathcal{F}} \overline{f}(\mathcal{T}_P) \setminus \mathbf{P}_\iota)$ and each element $l \in Comp^p_{\mathcal{F}} \setminus \kappa^0_p$, we define the following *knowledge* automaton $A^p_l = (\Sigma^p_l, Q^p_l, q^p_l, \delta^p_l, \mathcal{X}^p_l)$, where

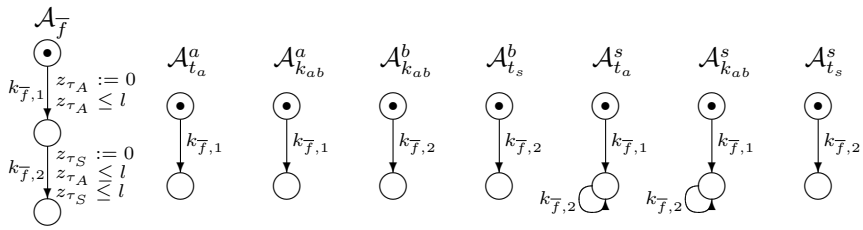
– $\Sigma^p_l \stackrel{def}{=} \{k \in \bigcup_{\overline{f} \in \mathcal{F}} \Sigma_{\overline{f}} \mid Cond_1(k) \vee Cond_2(k)\}$ with
$Cond_1(k) := (\exists \overline{f} \in \mathcal{F})(\exists \sigma \in \delta_{\overline{f}})$
$\sigma = (s^{\overline{f}}_{i-1}, k, TConstr^{\overline{f}(\alpha_i)}, \{z_\tau \mid \tau \in (\mathcal{T}_T \cap Gen^{f(\alpha_{i-1})})\}, s^{\overline{f}}_i) \wedge$
(i) $(p = Send^{f(\alpha_i)} \wedge l \in Gen^{f(\alpha_i)}) \vee$
(ii) $(p = Resp^{f(\alpha_i)} \wedge l \in \xi_{\{k_p^{-1}\}}(\{Lett^{f(\alpha_i)}\}) \wedge$
$\wedge (\forall j \in \{1, \ldots, i-1\})((p = Resp^{f(\alpha_j)} \Rightarrow l \notin \xi_{\{k_p^{-1}\}}(Lett^{f(\alpha_j)})) \wedge$
$\wedge (p = Send^{f(\alpha_j)} \Rightarrow l \notin Gen^{f(\alpha_j)}))$
$Cond_2(k) := (\exists \overline{f} \in \mathcal{F})(\exists \sigma \in \delta_{\overline{f}})$
$\sigma = (s^{\overline{f}}_{i-1}, k, TConstr^{\overline{f}(\alpha_i)}, \{z_\tau \mid \tau \in (\mathcal{T}_T \cap Gen^{f(\alpha_{i-1})})\}, s^{\overline{f}}_i) \wedge$
(iii) $(p = Send^{f(\alpha_i)} \wedge l \in Comp^{f(\alpha_i)} \setminus Gen^{f(\alpha_i)}) \vee$
(iv) $(p = Resp^{f(\alpha_i)} \wedge l \in \xi_{\{k_p^{-1}\}}(\{Lett^{f(\alpha_i)}\}) \wedge$
$\wedge (\forall j \in \{1, \ldots, i-1\})((p = Resp^{f(\alpha_j)} \Rightarrow l \notin \xi_{\{k_p^{-1}\}}(Lett^{f(\alpha_j)})) \wedge$
$\wedge (p = Send^{f(\alpha_j)} \Rightarrow l \notin Gen^{f(\alpha_j)}))$
– $Q^p_l = \{q^p_l, s^p_l\}$ is the set of states (q^p_l is the initial state),
– δ^p_l is the transition relation given as follows
$(q^p_l, k, True, \emptyset, s^p_l) \in \delta^p_l$ iff $Cond_1(k)$, $(s^p_l, k, True, \emptyset, s^p_l) \in \delta^p_l$ iff $Cond_2(k)$.

If the automaton A^p_l is in the state q^p_l, then this means that the participant p does not know l. If the automaton A^p_l moves to the state s^p_l, 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 $\overline{f}(\alpha_i)$. The condition (ii) says that p learns about

l at the step $\overline{f}(\alpha_i)$. 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 ι and each letter $l \in Comp^{\iota}_{\mathcal{F}} \setminus \kappa^0_{\iota}$ we define the knowledge automaton $A^{\iota}_l = (\Sigma^{\iota}_l, Q^{\iota}_l, q^{\iota}_l, \delta^{\iota}_l, \mathcal{X}^{\iota}_l)$ in a similar way to the definition of A^p_l. So, we have: $Q^{\iota}_l = \{q^{\iota}_l, s^{\iota}_l\}$ is a set of the states, q^{ι}_l is the initial state, δ^{ι}_l is the transition relation given as follows: $(q^{\iota}_l, k, True, \emptyset, s^{\iota}_l) \in \delta^{\iota}_l$ iff $Cond^{\iota}_1(k)$, $(s^{\iota}_l, k, True, \emptyset, s^{\iota}_l) \in \delta^{\iota}_l$ iff $Cond^{\iota}_2(k)$. The differences w.r.t A^p_l are as follows: the condition (i) is omitted, in (ii, iv) we replace $p = Send^{f(\alpha_i)}$ with $Send^{f(\alpha_i)} \in \mathbf{P}_{\iota}$ and $p = Resp^{f(\alpha_i)}$ with $Resp^{f(\alpha_i)} \in \mathbf{P}_{\iota}$. Moreover, we replace (iii) with $(Send^{f(\alpha_i)} \in \mathbf{P}_{\iota} \ \wedge (\exists X \subseteq \mathbf{L})(X \vdash Lett^{f(\alpha_i)} \ \wedge \ l \in X \ \wedge k = k^X_{f,i}))$. The new condition (iii) enables ι to use l while composing new letters.

Example 2. The network of automata for the WMF Protocol given in Example 1 with the partial interpretation $\overline{f}$, where $\overline{f}(\mathcal{A}) = a$, $\overline{f}(\mathcal{B}) = b$, $\overline{f}(\mathcal{S}) = s$, $\overline{f}(\tau_A) = t_a$, $\overline{f}(\tau_S) = t_s$, $\overline{f}(\mathcal{L}_{\mathcal{F}}) = l$, etc. is as follows.



Execution automaton *Knowledge automata*

Recall that we are dealing with the protocol Σ and a finite set $\mathcal{F}$ of its partial interpretations. Let $\mathcal{A}_{\mathcal{P}} = (N, Q, s^0, \delta, X)$ be the product automaton of the following set of the automata
$\{A_{\overline{f}} \mid \overline{f} \in \mathcal{F}\} \cup \{A^p_l \mid p \in \bigcup_{\overline{f} \in \mathcal{F}} \overline{f}(\mathcal{T}_P) \cap (\mathcal{P} \cup \{\iota\}) \wedge l \in Comp^p_{\mathcal{F}}\}$.

Consider the *state space* $C_c(\mathcal{A}_{\mathcal{P}}) = (Q, \mathfrak{s}^0, \rightarrow_c)$ of the product automaton $\mathcal{A}_{\mathcal{P}}$. Let $(k_{\overline{f^1},i_1}, k_{\overline{f^2},i_2}, \ldots, k_{\overline{f^k},i_k})$ be any sequence of actions from the set $\bigcup_{\overline{f} \in \mathcal{F}} \Sigma_f$. By a run on the word $(k_{\overline{f^1},i_1}, k_{\overline{f^2},i_2}, \ldots, k_{\overline{f^k},i_k})$ in the concrete space $C_c(\mathcal{A}_{\mathcal{P}})$ we mean the following sequence: $\rho = \mathfrak{s}_0 \xrightarrow{\delta_1}_c \mathfrak{s}_0 + \delta_1 \xrightarrow{a_1}_c \ldots \mathfrak{s}_{k-1} \xrightarrow{\delta_k}_c \mathfrak{s}_{k-1} + \delta_k \xrightarrow{a_k}_c \mathfrak{s}_k$, where $a_j = k_{\overline{f^j},i_j}$ 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 $j = 2, \ldots, k$ and for some interpretations f^j associated with partial interpretations $\overline{f^j}$ (for $j = 1, \ldots, k$). Additionally if $\mathfrak{s}_j = (s_j, v_j)$, then $s_j \xrightarrow{a_j, \mathfrak{cc}_j, \mathcal{X}_j} s_{j+1} \in \delta$, where $\mathfrak{cc}_j = Z(TConstr^{f^j(\alpha_{i_j})})$ and $v_j \in [\![\mathfrak{cc}_j]\!]$ and $v_{j+1} = v_j[\{z_\tau \mid \tau \in (\mathbf{T} \cap Gen^{\overline{f}(\alpha_{i_j})})\} := 0]$.

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

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

$$\rho = \mathfrak{s}_0 \xrightarrow{\delta_1}_c \mathfrak{s}_0 + \delta_1 \xrightarrow{a_1}_c \ldots \mathfrak{s}_{k-1} \xrightarrow{\delta_k}_c \mathfrak{s}_{k-1} + \delta_k \xrightarrow{a_k}_c \mathfrak{s}_k,$$

in the space $C_c(\mathcal{A}_\mathcal{P}) = (Q, \mathfrak{s}^0, \rightarrow_c)$ on the word $(a_1, a_2, \ldots, a_k)$, where:

$$a_j \in \begin{cases} \{k_{\overline{f^j}, i_j}\} & \text{if} \quad Send^{f^j(\alpha_{i_j})} \in \mathbf{P}, \\ \{k^X_{\overline{f^j}, i_j} \mid X \vdash Lett^{f^j(\alpha_{i_j})}\} & \text{if} \quad Send^{f^j(\alpha_{i_j})} \in \mathbf{P}_\iota, \end{cases}$$

$\delta_1 = Time^{f^1(\alpha_{i_1})}$ *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 induction on the length of a computation (run) Omitted because of a lack of space (see [9]).

Thanks to above theorem, we can reduce the 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 and Conclusions

We have tested susceptibility to authentication and secrecy attacks for several timed security protocols: CCITT(1), WMF, Denning-Sacco, Kerberos, and the timed version of NSPK. The space considered consists of the two honest participants A and B, the honest server S (if needed), and the Intruder I. Additionally, we assume that every participant has only one time ticket. The experiments show that our method captures all the mentioned above types of attacks upon the protocols as well as reports correctness for flawless protocols like NSPK_{Fix} In Table 1. our results are compared with these of TPMC [12] 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.

The experimental results show that for some protocols our tool behaves much better than SATMC and TPMC, which is a result of our distributed representation favouring symbolic verification.

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.

Protocol	Sessions	VeriCS			SATMC*	TPMC
		Clauses	Memory (MB)	Time (sec)	Time (sec)	Time (sec)
NSPK	3	15124	2,56	0,04	0,10	0,86
NSPK_{Fix}	3	16496	2,72	0,30	NA	0,10
WMF	3	14596	2,47	0,05	NS	0,44
CCITT (1)	3	5146	2,05	0,02	NS	NA
Denning-Sacco	3	12440	2,33	0,06	NS	NA
Kerberos	3	38119	3,79	0,26	6,24**	NA
NSPK_{Time}	3	52816	4,60	1,78	NS	NA

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, *) - linear encoding, **) - for the untimed version, NS - not supported by the tool, NA - not available to us.

References

1. A. Armando, et all, The AVISPA tool for the automated validation of internet security protocols and applications. In Proc. of *CAV'05*, vol. 3576 of *LNCS*, pp. 281–285, 2005.
2. A. Armando and L. Compagna. SAT-based model-checking for security protocols analysis. *Int. J. Inf. Secur.*, 7(1):3–32, 2008.
3. J. Bengtsson, K. G. Larsen, F. Larsson andP. Pettersson, W. Yi, and C. Weise. New generation of Uppaal. In *Proc. of the Int. Workshop on Software Tools for Technology Transfer*, 1998.
4. R. Corin, S. Etalle, P. H. Hartel, and A. Mader. Timed model checking of security protocols. In Proc. of *FMSE'04*, pp. 23–32. ACM, 2004.
5. G. Delzanno and P. Ganty. Automatic verification of time sensitive cryptographic protocols. In Proc. of *TACAS'04*, vol. 2988 of *LNCS*, pp. 342–356, 2004.
6. N. Evans and S. Schneider. Analysing time dependent security properties in CSP using PVS. In Proc. of *ESORICS'00*, vol. 1895 of *LNCS*, pp. 222–237, 2000.
7. R. Gorrieri, E. Locatelli, and F. Martinelli. A simple language for real-time cryptographic protocol analysis. In Proc. of *ESOP'03*, vol. 2618 of *LNCS*, pp. 114–128, 2003.
8. G. Jakubowska and W. Penczek. Is your security protocol on time ? In *Proc. FSEN 2007* Tehran, vol. 4767 of *LNCS*, pages 65–80. Springer, 2007.
9. M. Kurkowski and W. Penczek Timed automata based model checking of timed security protocols. ICS PAS Technical Report, 2008, http://www.ipipan.waw.pl/~penczek/WPenczek/2008/TR1010.pdf.
10. M. Kurkowski, W. Penczek, and A. Zbrzezny. SAT-based verification of security protocols via translation to networks of automata. In Proc. of *MoChart IV*, vol. 4428 of *LNAI*, pp. 146–165. Springer-Verlag, 2007.
11. G. Lowe. CASPER: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6(1-2):53–84, 1998.
12. M. Benerecetti, N. Cuomo, and A. Peron. TPMC: A model checker for time-sensitive security protocols. In Proc. of *HPCS 2007*, Prague, pp. 742-749, 2007.
13. Verics Team. Verics 2007 - a model checker for real time and multi-agent systems. http://pegaz.ipipan.waw.pl/verics/