Using Backward Induction Techniques in (Timed) Security Protocols Verification*

Mirosław Kurkowski, Olga Siedlecka-Lamch, and Paweł Dudek

Institute of Computer and Information Sciences
Częstochowa University of Technology,
Dąbrowskiego 73, 42-201 Częstochowa, Poland
{mkurkowski,olga.siedlecka,pdudek}@icis.pcz.pl

Abstract. This paper shows a new way of automatic verification of properties of untimed and timed security protocols. To do this we use a modified version of previously introduced formal model based on a network of synchronized (timed) automata that expreses behaviour and distributed knowledge of users during protocol executions. In our new approach we will use the backward induction method for searching of a tree of all real executions of an investigated protocol. Our approach uses additionally the boolean encoding of constructed structures and SAT solvers for searching answers to the questions about investigated properties which are expressed as reachability or unreachability of undesired states in a considered model. We exemplify all our notions and formalisms on the well known NSPK, and show experimental results for checking authentication and security properties of a few untimed and timed protocols.

Keywords: Security Protocols Verification, Backward Induction.

1 Introduction

The modern people process tens of gigabytes of information a day, most of which is transferred electronically. "Consumption" of information is a must, it gives people the knowledge, capabilities, and does not allow to fall outside the margins of society. The transfer of information must remain undisturbed, consistent with reality and not threatened by outside intruders. Such communication should be guaranteed by a number of communication protocols, secured by modern cryptographic techniques. The heart of every communication protocol is a security protocol that meets the following requirements: mutual authentication of communicating entities, confidentiality of transmitted information, integrity of this information and a new key-session distribution.

Security protocols used in practice satisfy one or more of the above requirements using either symmetric or asymmetric cryptography. Many of the developed security protocols proved to be vulnerable to the attack by malicious Intruders [21, 22], and investigators undertook the challenge of formal methods of protocol verification.

* The second and the third author acknowledge support from EU Project (European Social Fund.) PO KL *Information technologies: Research and their interdisciplinary applications*, Agreement UDA***/*-*/*POKL.04.01.01-00-051/10-00.

K. Saeed et al. (Eds.): CISIM 2013, LNCS 8104, pp. 265–276, 2013.

The process of creating the protocol is difficult, but it seems more difficult to verify. The first step is to create a valid, full formal description. The description should include all parties, states and actions occurring in the protocol: participants (sender and receiver), the information about keys that encrypt the transmitted data, actions of generating, encoding, sending, receiving and decrypting information. The Common Language is a commonly used protocol specification language, which unfortunately does not allow for a full description. It shows only the scheme of sending messages and their construction. In this case there is a need for using more complicated languages, for example presented in the papers [19, 18, 17].

The precisely specified protocol can be automatically verified in many possible ways. The field of automatic verification has been exploited for many years by both academic and commercial institutions. Basically, there are two main ways of verification: testing real and virtual systems (simulations), or modeling and formal verification. Of course after testing the already implemented systems, we can be sure that the system has worked properly only so far. Formal modeling and verification is based on building adequate mathematical structures, showing the actions that occur during the protocol execution. Of course these methods are succesfully used to model and verify various kinds of computer systems. Good examples are [11, 16]. There are numerous approaches to the verification of security protocols associated sometimes with appropriate tools [1, 2, 14, 3, 8, 9, 12, 5].

The algorithmic approach is based predominantly on model checking, but we can distinguish among formal methods also those inductive [26, 4] or deductive [16, 6, 20]. Intuitively, model checking of a protocol proceeds in checking whether a formal model of protocol executions contains an execution or a reachable state that represents an attack upon the protocol. Comparing to standard model checking techniques for communication protocols or for distributed systems, the main difficulty is caused by the need to model the Intruder which is responsible for generating attacks as well as changes of knowledge (about keys, nonces, etc.) of the participants. The Intruder can be modeled in several ways, although the Dolev-Yao model and its versions seems to be the most adequate [10]. In mentioned model the Intruder has access to all the transmitted information, can collect and use it with a precision according to its skills and resources, and is able to send it somehow and sometime through the network. Of course, the Intruder does not need to comply with the requirements of using the protocols and, in particular, does not have to use fresh information or to meet time requirements.

Using model checking we face another serious problem - the exponential explosion of states, which depends on the number of participants, sessions, or messages. The model should predict all possible scenarios with a huge amount of information generated and sent by the Intruder. In considered case the computational complexity of model checking algorithms is typically exponential in parameters of the verified protocols, particularly in the number of participants and execution steps. This problem calls for the modification of existing models and algorithms, as well as finding new ones.

So far, our studies in this area were directed at the two models: a network of synchronized automata for modeling separately the executions of a protocol and the knowledge of the participants like in [19, 18, 17] or using the chains of states of protocol execution, see [27]. In the first case, the constructed networks of automata were translated to the

propositional boolean formulas. The security property was firstly expressed as a property of reachability of certain global states in the automata network, and subsequently as a satisfiability of the formula. SAT-solver answered the question whether satisfying valuation exists, which was equivalent to the existence of an attack or an undesirable situation. By contrast, if the formula turned out to be unsatisfiable, it meant that no attack exists.

For the searching of models of executions of concurrent systems, the backward induction technique is sometimes used. Generally speaking the method consists in constructing a tree of executions in accordance to the reversed relation to that which determines the dependence between the states in the basic model. In some situations, for example in the case of testing the reachability of certain states, this causes the reduction of size of the constructed trees. In the case of protocols this approach was sucessfully used in a few works, for example in [15].

In this article we propose the approach of the backward induction technique for the investigation of properties of protocols by building a network of automata encoding the tree of protocol execution, where the runs take place inversely to the runs proposed in [19, 18, 17]. The automata built in this approach are slightly bigger in size, which is why the experimental results confirm greater efficiency of the proposed approach in the case of protocols with a small number of steps. It is interesting that this method may also be used in the case of time-dependent protocols, and the networks of time automata which model them.

2 The Needham-Schroeder Public Key Protocol

The designing of the communication protocols is a very hard task which is connected with the possibility of appearing many problems with later use of a made protocol. As the essential example the NSPK protocol can be presented [25].

In the notation of this protocol there are designations $i(A)$ and $i(B)$, which express identifiers of the participants of the communication, respectively the A and B participants. The expression $\langle X \rangle_{K_A}$ means the X message encrypted by the public key of A participant. Similarly, the message encrypted by the public key of B participant - $\langle X \rangle_{K_B}$.

The A participant has a very important role – starts the run of the protocol. The aim of the execution of this protocol is to achieve the mutual confirmation of the identity (the authentication) between the communicating participants. The designation $A \rightarrow B : X$ refers to sending the X message from the A participant to B participant. We assume that sending the message causes the operation of receiving it by the suitable person. The concatenation of the elements in the message was determined by the operator "·".

Example 1. The scheme of NSPK protocol proposed in [25] is as follows:

$\alpha_1 \;\; A \rightarrow B \;\; : \langle N_A \cdot i(A) \rangle_{K_B}$,
$\alpha_2 \;\; B \rightarrow A \;\; : \langle N_A \cdot N_B \rangle_{K_A}$,
$\alpha_3 \;\; A \rightarrow B \;\; : \langle N_B \rangle_{K_B}$.

The A participant in first step of the protocol execution generates the random number (nonce) N_A and sends it to the B participant with A's identifier encrypted by the public key of the B user. The next step of protocol execution starts from generating by the B

participant its own random number N_B. Next it creates the message consisting of random numbers of both communicating participants, encrypts this message by the public key of the A participant and sends it to the A. In the next stage of the execution, A runs decrypting operation of the received message and the operation of comparing the N_A number, which got from B with the number N_A, which was prepared by him. If both numbers are equal, A considers B as authenticated. In the last step of the protocol execution A sends to B its random number N_B encrypted by the K_B key. After decrypting and comparing the proper numbers, the B participant can consider the A participant as authenticated. This protocol execution should guarantee both participants the identity of the communicating persons.

Example 2. Figure 1 presents the automata based model of honest execution of NSPK Protocol due to formal definitions from [19, 18, 17]. In the presented networks synchronisation of component automata is used, it consists in the fact that the global transition (in the network) labeled by the label α is enabled, if in all components if there exists transitions labeled by α, then at least one of them in each automaton is enable. It is assumed further that all states are accepting ones. As it can be seen the model consists of automata modeling the execution of external actions of the protocol (sending messages) $\mathcal{A}$ and automata that model distributed knowledge of participants ($\mathcal{A}^A_{N_A}, \mathcal{A}^A_{N_B}, \mathcal{A}^B_{N_A}, \mathcal{A}^B_{N_B}$). For example, the automaton $\mathcal{A}^A_{N_A}$ models the knowledge of the user A about the random number N_A. Executions of additional actions constituting the protocol are represented by labels on automata transitions. Transitions labeled by α_1 model execution of the first step of the protocol. During execution of that step, users A and B acquire knowledge about the number N_A. Transitions labeled α_2 model the performance of the second action of the protocol, in which users gain knowledge about the number N_B. We should pay attention on the loop in automata $\mathcal{A}^B_{N_A}$, labeled also by α_2, it models the condition to possess knowledge about number N_A by B, to execute the second step of the protocol. Accordingly, the transitions labeled α_3, model the execution of the third step of the protocol.

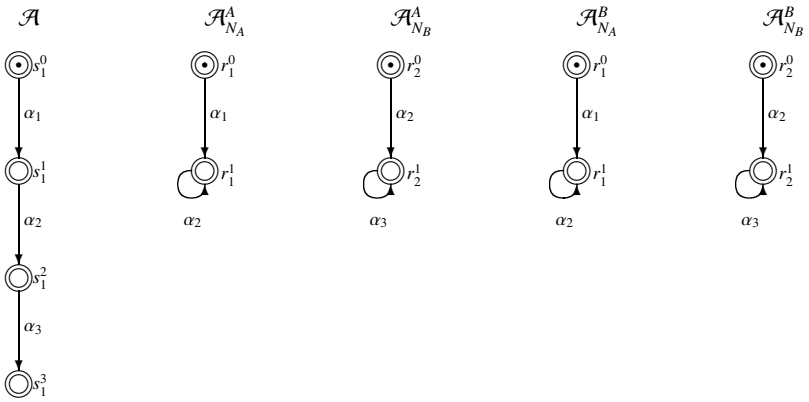

Fig. 1. Automata model of execution of the NSPK Protocol

The NSPK protocol was used through 17 years. In 1995 Gavin Lowe discovered the version of the protocol execution which consisted a possible attack, showing that the

NSPK protocol is susceptible to a break-in [21]. The Intruder is the additional participant with its own identifier and the keys, determined by ι symbol. The Intruder does not do the protocol according to the scheme but uses own ways, impersonates other users and cheats them.

Example 3. The attack presented by Gavin Lowe is as follows:

$\alpha_1^1\ A \rightarrow \iota : \langle N_A \cdot i(A)\rangle_{K_\iota}$,
$\alpha_1^2\ \iota(A) \rightarrow B : \langle N_A \cdot i(A)\rangle_{K_B}$,
$\alpha_2^2\ B \rightarrow \iota(A) : \langle N_A \cdot N_B\rangle_{K_A}$,
$\alpha_2^1\ \iota \rightarrow A : \langle N_A \cdot N_B\rangle_{K_A}$,
$\alpha_3^1\ A \rightarrow \iota : \langle N_B\rangle_{K_\iota}$,
$\alpha_3^2\ \iota(A) \rightarrow B : \langle N_B\rangle_{K_B}$.

The Lowe scheme shows two simultaneously NSPK protocol executions. The α^1 execution refers to the communication between the A participant and the Intruder. The α^2 execution refers to the situation where the Intruder pretends to be the A participant and in his name communicates with B. To have correct protocol which cannot be broken it is enough to add the identifier of the sender to sent message in the second step of the original NSPK protocol execution: $\langle N_A \cdot N_B \cdot i(B)\rangle_{K_A}$. All known verification methods and tools confirm safety of this version of NSPK.

3 Backward Induction in Modeling and Verification of Untimed Protocols

As mentioned earlier, in the verification of systems modeled by transitional structures, backward induction is sometimes used. This takes place when there is hope that it is easier to build an execution tree of a program or system by constructing it, so to say, from the end, moving backwards from a certain state in accordance to a specific relation of a passage in the structure. Examples indicate that sometimes this method is very efficient, especially in the case of stating unreachability of certain states (see [15]).

In order to verify security protocols using the backward induction method, we will now define the product automaton which models inversely the runs which take place in the product automaton constructed in previous sections. An appropriate theorem of the adequacy of the calculations done in both product automata will allow the searching of the tree using the backward induction method.

Let $\mathfrak{A}$ be a family of automata $\mathcal{A}_i$, $i \in I$ meeting the following conditions. Let $\mathcal{A}_i = \{Q_i, \Sigma_i, \delta_i, s_i^0, F_i\}$ be an automaton, where:

- $Q_i = \{s_i^0, s_i^1, \ldots, s_i^t\}$, for all $i \in I$ and some $t \in N$ is the set of states of automata $\mathcal{A}_i$,
- $\Sigma_i = \{k_i^1, k_i^2, \ldots, k_i^s\}$ is the set of labels, where ($\Sigma_i \cap \Sigma_j = \emptyset$, dla $i \neq j$) for all automata from $\mathfrak{A}$,
- $\delta_i \subseteq Q_i \times \Sigma_i \times Q_i$ is a transition relation that meets the following conditions:
  - $(s_i^l, k, s_i^j) \in \delta_i$ iff $l = j - 1$, for any $l = 0, 1, \ldots, s - 1$ oraz $j = 1, 2, \ldots, s$,
  - if $(s_i^l, k, s_i^{l+1}) \in \delta_i$ and $(s_i^p, k, s_i^{p+1}) \in \delta_i$, then $l = p$.
- s_i^0 is an initial state, $F_i = Q_i$ is the set of finite states.

It is noticeable that this family defines the automata which have the same structure as automata modeling the execution of protocols defined in [17–19]. The following conditions have been met:

- the sets of states of all the automata are equally numerous.
- the relation of the passage in each automaton is determined only on a pair of states directly following one another.
- labels do not duplicate in different automata, nor do they duplicate within the scope of one automaton,
- all states of the automata are accepting ones.

We denote by Σ the set of all labels from automata from the family $\mathfrak{A}$ ($\Sigma = \bigcup_{i \in I} \Sigma_i$).

Now the family of automata corresponding to the knowledge automata will be defined.

Let $\mathfrak{B}$ be a family of automata $\mathcal{B}_j$, $j \in J$ that meets the following conditions. $\mathcal{B}_j = \{\mathcal{R}_j, \Sigma, \rho_j, r_j^0, T_j\}$ where:

- $\mathcal{R}_j = \{r_j^0, r_j^1\}$, for all $j \in J$,
- Σ is the set of labels constructed before,
- $\rho_j \subseteq \mathcal{R}_j \times \Sigma \times \mathcal{R}_j$ is the transition relation that meets the condition: $(r_j^l, k, r_j^p) \in \rho_j$ iff $l = 0 \wedge p = 1$ or $l = p = 1$,
- r_j^0 is an initial state, $T_j = \mathcal{R}_j$ is a set of finite states.

It is noticeable here that these automata have the same structure as knowledge automata constructed in the previous section:

- each have two states,
- transitions are defined only from the first state to the second, and from the second state to itself,
- labels are taken from the set of all labels from the automata of the $\mathfrak{A}$ family,
- all states are accepting ones.

By $\underline{\mathfrak{A}\mathfrak{B}}$ we denote the product automaton (network of automata) defined over the family $\mathfrak{A} \cup \mathfrak{B}$ due to definition of product automaton given in [17].

We will now define the product automaton modeling the runs of automaton $\underline{\mathfrak{A}\mathfrak{B}}$ executed inversely.

For all automata $\mathcal{A}_i \in \mathfrak{A}$ let $\overline{\mathcal{A}_i}$ be a smallest automaton that meets the following conditions:

$\overline{\mathcal{A}_i} = \{Q_i, \Sigma_i, \overline{\delta_i}, s_i^t, \{s_i^t, s_i^0\}\}$ where:

- $\overline{\delta_i} \subseteq Q_i \times \Sigma_i \times Q_i$ is a transition relation that meets the condition: if $(s_i^l, k, s_i^{l+1}) \in \delta_i$, then $(s_i^{l+1}, k, s_i^l) \in \overline{\delta_i}$ and $(s_i^{l+1}, k, s_i^0) \in \overline{\delta_i}$.
- s_i^t is an initial state, $\{s_i^t, s_i^0\}$ is a set of accepting states.

One should notice that the sets of states and labels are the same as in corresponding automata of the family $\mathfrak{A}$. By $\overline{\mathfrak{A}}$ we define the family of all such constructed automata.

Moreover for all automata $\mathcal{B}_j \in \mathfrak{B}$ we define automata $\overline{\mathcal{B}_j}$ as the smallest automata that meet for every $j \in J$:

$\overline{\mathcal{B}_j} = \{\overline{\mathcal{R}_j}, \Sigma, \overline{\rho_j}, r_j^2, \{r_j^0, r_j^2\}\}$ where:

- $\overline{\mathcal{R}_j} = \mathcal{R}_j \cup \{r_j^2\} = \{r_j^0, r_j^1, r_j^2\}$,
- Σ is the set of labels constructed before,
- $\overline{\rho_j} \subseteq \overline{\mathcal{R}_j} \times \Sigma \times \overline{\mathcal{R}_j}$ is the transition relation that meets the following conditions: if $(r_j^l, k, r_j^p) \in \rho_j$, then $(r_j^p, k, r_j^l) \in \overline{\rho_j}$ and $(r_j^2, k, r_j^l) \in \overline{\rho_j}$.
- r_j^2 is an initial state, $\{r_j^0, r_j^2\}$ is a set of accepting states.

By $\overline{\mathfrak{B}}$ we denote the family of all automata constructed before.

Example 4. Figures 2 and 3 present the intuition of the constructions implemented above. According to the above mentioned definitions of the beginning states of automata $\mathcal{A}, \mathcal{B}_1, \mathcal{B}_2$ are accordingly: s_1^0, r_1^0, r_2^0, whereas accepting ones are all states of the automata. In the case of automata $\overline{\mathcal{A}}, \overline{\mathcal{B}_1}, \overline{\mathcal{B}_2}$, beginning states are accordingly: s_1^3, r_1^2, r_2^2, whereas accepting ones are states $s_1^3, s_1^0, r_1^2, r_1^0, r_2^2, r_2^0$. One should notice that in the first network runs over the words: q_1, q_1q_2 and $q_1q_2q_4$ exist. In the second product automaton there are accordingly runs over the words: q_1, q_2q_1 and $q_4q_2q_1$.

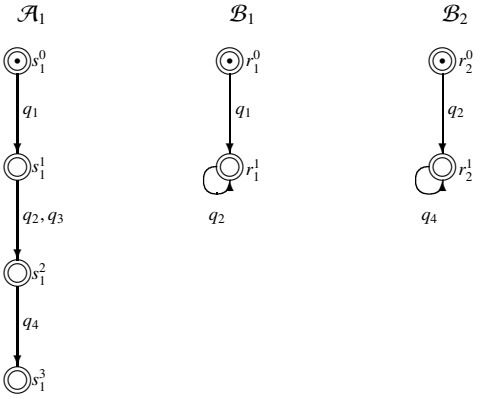

Fig. 2. Direct automata model

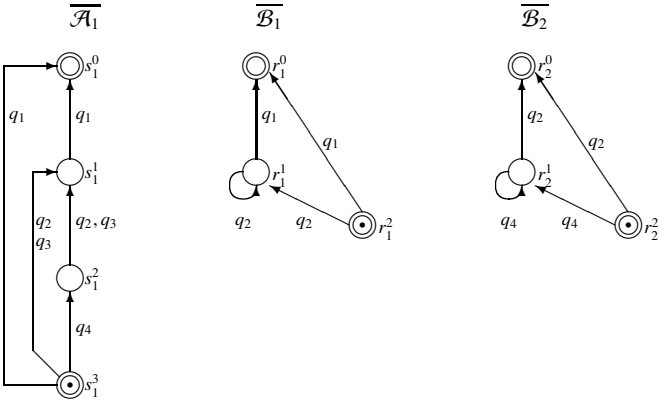

Fig. 3. Reverse automata model

One can see from the previous example that suitable accepting runs exist in the product automaton of the first type only if in the product automaton of the second type exist accepting runs which are mirror-reversed. The suitable example of network of automata that models reverse execution of NSPK is presented in Example 5.

Example 5. Figure 4 show the reverse automata model for execution of NSPK Protocol given in *Example 2*.

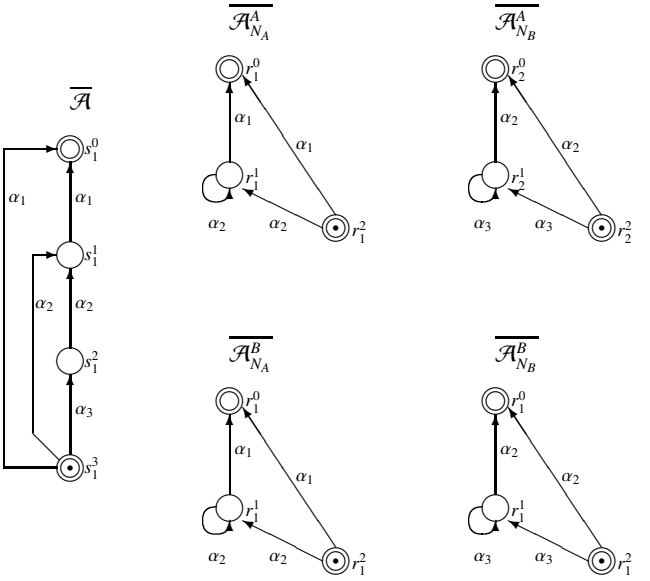

Fig. 4. Reverse automata model for NSPK execution

As done so previously, let's consider the product automaton $\overline{\mathfrak{AB}}$. For such constructed families of automata the following theorem holds.

Theorem 1. *In the product automaton* $\underline{\mathfrak{AB}}$ *there exists the run over the word* $q = q_1, q_2, \ldots, q_w$ *iff in the product automaton* $\overline{\mathfrak{AB}}$ *there exists the run over the word* $\overline{q} = q_w, \ldots, q_2, q_1$.

Proof of Theorem 1 proceeds by induction considering the length of the word q.

Proof of the right implication. In order to prove the inductive base, one should notice that if in product automaton $\underline{\mathfrak{AB}}$ there exists a run over the word $q = q_1$, then there exists exactly one automaton from the family $\mathfrak{A}$ in which the transition labelled q_1 is the transition from the beginning state to its successor. Therefore it is fulfilled that for a given $i \in I$ we have $(s_i^0, q_1, s_i^1) \in \delta_i$. According to the definition of the automaton $\overline{\mathfrak{AB}}$ in automaton $\overline{\mathcal{A}_i}$ there exists the transition $(s_i^t, q_1, s_i^0) \in \overline{\delta_i}$. It is noticeable that according to the definitions of knowledge automaton from the family $\underline{\mathfrak{AB}}$, in which any transition is labelled with label q_1, then there also exists an appropriate transition labelled q_1 leading from the beginning state r_j^2 to state r_j^0. For automata where in their transitions the label

$\underline{q_1}$ does not appear, an apropriate definition of ending states of the product automaton $\overline{\mathfrak{AB}}$ should be reminded.

In order to prove the inductive step, it is assumed that the right implication is true for words of length k, therefore it is true that if in the product automaton $\underline{\mathfrak{AB}}$ there is a run over the word $q = q_1, q_2, \ldots, q_k$, then in the product automaton $\overline{\mathfrak{AB}}$ there is a run over the word $\overline{q} = q_k, \ldots, q_2, q_1$. It now should be proved that the implication is true for words of length $k + 1$.

Any given word $q = q_1, q_2, \ldots, q_k, q_{k+1}$ should now be considered. According to the definition of the product automaton $\underline{\mathfrak{AB}}$ there exists exactly one automaton from the family $\mathfrak{A}$ in which there exists a transition labelled q_{k+1}. So it is fulfilled that for a given $i \in I, (s_i^j, q_{k+1}, s_i^{j+1}) \in \delta_i$. According to the definition of automaton $\overline{\mathfrak{AB}}$ in automaton $\overline{\mathcal{A}_i}$ there exists a transition $(s_i^t, q_1, s_i^j) \in \overline{\delta_i}$. It is also noticeable that according to definitions of knowledge automata from the family $\underline{\mathfrak{AB}}$, in which any transition is labelled with label q_{k+1}, there also exists an appropriate transition labelled q_{k+1}. Proof of left implication is analogous.

It should also be noticed that the same property concerns the accepting runs. The reason for this results directly from the above statement and in the definition of accepting states of the components of families of both types.

4 Backward Induction in Modeling and Verification of Timed Protocols

Just like in the previous section, a model automata of time-dependent protocol executions which allows verification using the backward induction method will now be presented. Because the knowledge automata in this chapter do not include time aspects, only the construction of timed automata representing protocol execution will be introduced.

Let C be a set of time constraints defined in [19, 18]. Aditionally let $\mathfrak{TA}$ be a family of timed automata $\mathcal{TA}_i$ ($i \in I$ for some indices from I) that meet the following conditions.

Let $\mathcal{TA}_i = \{Q_i, \Sigma_i, \delta_i, s_i^0, F_i, \mathcal{X}_i\}$ be a timed automaton, where:

- let $Q_i = \{s_i^0, s_i^1, \ldots, s_i^t\}$, for all $i \in I$ and some $t \in N$ be a set of states in $\mathcal{A}_i$,
- let $\Sigma_i = \{k_i^1, k_i^2, \ldots, k_i^s\}$ be the set of labels where $\mathfrak{A}$ ($\Sigma_i \cap \Sigma_j = \emptyset$, dla $i \neq j$),
- $\delta_i \subseteq Q_i \times \Sigma_i \times C \times 2^{\mathcal{X}_i} \times Q_i$ be a transition relation that satisfies:
  - $(s_i^l, k, cc, X, s_i^j) \in \delta_i$ iff $l = j - 1$, for all $l = 0, 1, \ldots, s - 1$ and $j = 1, 2, \ldots, s$,
  - if $(s_i^l, k, cc, X, s_i^{l+1}) \in \delta_i$ and $(s_i^p, k, cc, X, s_i^{p+1}) \in \delta_i$, then $l = p$.
- s_i^0 is an initial state,
- $F_i = Q_i$ is the set of accepting states,
- $\mathcal{X}_i$ is the set of clocks.

This family defines automata which have the same structure as those defined in [19, 18] timed automata modeling the execution of time-dependent protocols. A timed automaton modeling reverse protocol execution will now be constructed.

For each automaton $\mathcal{TA}_i \in \mathfrak{A}$ let $\overline{\mathcal{TA}_i}$ be a smallest automaton that satisfies the following conditions:

$\overline{\mathcal{A}_i} = \{Q_i, \Sigma_i, \overline{\delta_i}, s_i^t, \{s_i^t, s_i^0\}, \mathcal{X}_i\}$ where:

- $\overline{\delta_i} \subseteq Q_i \times \Sigma_i \times C \times 2^{\mathcal{X}_i} \times Q_i$ be a transition relation that satisfies: if $(s_i^l, k, cc, X, s_i^{l+1}) \in \delta_i$, then $(s_i^{l+1}, k, cc, X, s_i^l) \in \overline{\delta_i}$ and $(s_i^{l+1}, k, cc, X, s_i^0) \in \overline{\delta_i}$.
- s_i^t is an initial state,
- $\{s_i^t, s_i^0\}$ is a set of accepting states.

In the case presented below, there holds a proper and adequate theorem concerning the existence of accepting runs in product automata including families execution automata as well as knowledge automata. The proof is analogous like in the previous one. For it should be noticed that only execution automata include nonempty time constraints, and these are accordingly modeled in a reverse product automata.

5 Experimental Results

In this section the results of experiments performed according to methodology discussed above will be presented and described. The results were obtained with the original implementation done in C++. For testing we used untimed and timed versions of four of well known protocols: NSPK (mentioned above), Lowe's version of NSPK (NSPKL), Wide Mouth Frog (WMF), and CCITT. The implementation generates for a given (in ProToc language [17, 18]) protocol and defined in it the space of considerations, different interpretations - executions of the tested protocol. These executions, in turn, are automatically translated to the network of untimed or timed automata. Then automata are translated to the boolean propositional formula in CNF (conjunction normal form) format. Formulas are optimized and tested using SAT-solver MiniSAT. The existence of a valuation satisfying the formula is equivalent to the existence of an attack on the protocol. Lack of satisfying valuation shows that there was no attack in the considered finite space. The module enables testing the previously described attacks. The performed calculations found the known attacks upon protocols. The tables presented below mainly focus on verification parameters: memory used by the CPU and computation time.

The verification method that uses the backward induction approach gave better results in the case of the WMF and CCITT protocols in both cases of untimed and timed versions. This may be caused by a small number of protocol steps, resulting in slight differences in the constructed model and the size of the formula. In the case of protocols with a larger number of steps automata model was a little biger and encoding gave larger formula. In this case searching by the solver took a bit more time than in the direct method.

Table 1 and 2 show a comparison of the direct verification and the backward induction technique in the case of untimed and time dependent protocols. After conducting a series of calculations for these protocols and their analyzis, the following conclusions can be made: the backward induction method gave somewhat better results in the case of protocols with a small number of steps. In the case of long protocols the method did not quicken obtained verification results.

Table 1. Comparison of direct and backward methods of untimed protocols verification

	Direct	Direct	Backward	Backward
Protocol	Memory (MB)	Time (s.)	Memory (MB)	Time (s.)
$WMF_{Untimed}$	8,56	0,004	7,78	0,002
$CCITT_{Untimed}$	9,13	0,041	8,84	0,031
NSPK	9,57	0,102	13,5	0,137
NSPKL	11,3	0,156	15,61	0,171

Table 2. Comparison of direct and backward methods for time dependent protocols

	Direct	Direct	Backward	Backward
Protocol	Memory (MB)	Time (s.)	Memory (MB)	Time (s.)
WMF	10,32	0,004	9,41	0,002
CCITT	10,89	0,063	10,24	0,045
$NSPK_{Timed}$	11,56	0,110	16,32	0,166
$NSPKL_{Timed}$	13,72	0,210	18,77	0,231

6 Summary

Research carried out by various international teams show that there is no single, "ideal" method of verification for all protocols. Often we must select the method for the given protocol. During the searching for the next models and methods we developed and implemented the backward induction method for constructing automata that model executions of security protocols. After a series of computation and their analysis we can draw the following conclusions: the backward induction method in the case of automata based modeling of executions and verification of security protocols due to methodology given in [17, 18] may give better results only in the case of protocols with a small number of steps and there it can be successfully used. For long protocol method does not speed up the computations.

References

1. Amnell, T., et al.: UPPAAL - Now, Next, and Future. In: Cassez, F., Jard, C., Rozoy, B., Dermot, M. (eds.) MOVEP 2000. LNCS, vol. 2067, pp. 99–124. Springer, Heidelberg (2001)
2. Armando, A., et al.: The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 281–285. Springer, Heidelberg (2005)
3. Armando, A., Compagna, L.: Sat-based model-checking for security protocols analysis. International Journal of Information Security 7(1), 3–32 (2008)
4. Bella, G., Massacci, F., Paulson, L.C.: Verifying the set registration protocols. IEEE Journal on Selected Areas in Communications 20(1), 77–87 (2003)
5. Bella, G., Paulson, L.C.: Using Isabelle to prove properties of the kerberos authentication system. In: Orman, H., Meadows, C. (eds.) Proc. of the DIMACS Workshop on Design and Formal Verification of Security Protocols (1997)
6. Burrows, M., Abadi, M., Needham, R.M.: A logic of authentication. ACM Trans. Comput. Syst. 8(1), 18–36 (1990)
7. Cohen, E.: TAPS: A first-order verifier for cryptographic protocols. In: CSFW 2000: Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW 2000), p. 144. IEEE Computer Society, Washington, DC (2000)

8. Corin, R., Etalle, S., Hartel, P.H., Mader, A.: Timed model checking of security protocols. In: Proc. of the 2004 ACM Workshop FMSE, pp. 23–32. ACM (2004)
9. Delzanno, G., Ganty, P.: Automatic verification of time sensitive cryptographic protocols. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 342–356. Springer, Heidelberg (2004)
10. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on Information Theory 29(2), 198–207 (1983)
11. El Fray, I., Kurkowski, M., Pejaś, J., Maćków, W.: A New Mathematical Model for the Analytical Risk Assessment and Prediction in IT Systems. Control & Cybernetics 41, 241–268 (2012); Systems Research Institute PAS, Warsaw
12. Evans, N., Schneider, S.: Analysing time dependent security properties in CSP using PVS. In: Cuppens, F., Deswarte, Y., Gollmann, D., Waidner, M. (eds.) ESORICS 2000. LNCS, vol. 1895. Springer, Heidelberg (2000)
13. Jakubowska, G., Penczek, W., Srebrny, M.: Verifying security protocols with timestamps via translation to timed automata. In: Proc. of the International Workshop on Concurrency, Specification and Programming (CS&P 2005), pp. 100–115. Warsaw University (2005)
14. Kacprzak, M., et al.: Verics 2007 - a model checker for knowledge and real-time. Fundam. Inform. 85(1-4), 313–328 (2008)
15. Kurkowski, M., Maćków, W.: Using Backward Strategy to the Needham-Schroeder Public Key Protocol Verification. In: Artificial Intelligence and Security in Computing Systems, pp. 249–260. Kluwer Academic Publishers, Boston (2003)
16. Kurkowski, M., Pejaś, J.: A Propositional Logic for Access Control Policy in Distributed Systems. In: Artificial Intelligence and Security in Computing Systems, pp. 157–191. Kluwer Academic Publishers, Boston (2003)
17. Kurkowski, M., Penczek, W.: Verifying Security Protocols Modeled by Networks of Automata. Fund. Inform. 79(3-4), 453–471 (2007)
18. Kurkowski, M., Penczek, W.: Verifying Timed Security Protocols via Translation to Timed Automata. Fund. Inform. 93(1-3), 245–259 (2009)
19. Kurkowski, M., Penczek, W.: Applying Timed Automata to Model Checking of Security Protocols. In: Wang, J. (ed.) Handbook of Finite State Based Models and Applications, pp. 223–254. Chapman&Hall/CRC Press, Boca Raton (2012)
20. Kurkowski, M., Srebrny, M.: A Quantifier-free First-order Knowledge Logic of Authentication. Fund. Inform. 72, 263–282 (2006)
21. Lowe, G.: Breaking and Fixing the Needham-Schroeder Public-key Protocol Using fdr. In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055, pp. 147–166. Springer, Heidelberg (1996)
22. Lowe, G.: Some new attacks upon security protocols. In: Proceedings of the Computer Security Foundations Workshop VIII. IEEE Computer Society Press (1996)
23. Meadows, C.: The NRL protocol analyzer: An overview. Journal of Logic Programming 26(2), 13–131 (1996)
24. Moore, J.H.: Protocol failures in cryptosystems. Proceedings of the IEEE 76(5) (1988)
25. Needham, R.M., Schroeder, M.D.: Using encryption for authentication in large networks of computers. Commun. ACM 21(12), 993–999 (1978)
26. Paulson, L.C.: Inductive analysis of the internet protocol tls. ACM Trans. Inf. Syst. Secur. 2(3), 332–351 (1999)
27. Siedlecka-Lamch, O., Kurkowski, M., Piech, H.: A New Effective Approach for Modeling and Verification of Security Protocols. In: Proceedings of 21st International Workshop on Concurrency, Specification and Programming (CS&P 2012), pp. 191–202. Humboldt University Press, Berlin (2012)