A New Effective Approach for Modelling and Verification of Security Protocols

Olga Siedlecka-Lamch[1], Mirosław Kurkowski[1], and Henryk Piech[1]

Institute of Computer and Information Sciences,
Częstochowa University of Technology ,
Dąbrowskiego 73, 42-200 Częstochowa, Poland
olga.siedlecka@icis.pcz.pl, mkurkowski@icis.pcz.pl,
h.piech@adm.pcz.czest.pl

Abstract. This paper presents a new mathematical model, and based on it a new automatic approach for verification of security protocols. This model contains a representation of the actions executed during the realization of the protocol, and the representation of changes of knowledge states of honest users and an Intruder. These possibilities are given to us by the formal definition (also shown in the article) of the runs representing the execution of protocols in real, open computer networks. Our model uses a simple and intuitive idea of chains of states for which the concept of protocol's properties verification has been given. This idea has been implemented, and the results presented in this article are compared with the results obtained by the use of the best verification tools.

Keywords: security protocols, modeling and verification, user's knowledge, Intruder's attacks

1 Introduction

Security protocols are a key point of safety in computer systems. In the open computer networks, especially in the systems which fulfill the aims of the public key infrastructure, these concurrent algorithms are often applied during an exchange of information for, generally speaking, assuring a sufficient level of security of data transmitted over networks and electronic transactions. The mutual authentication of communicating parties (users, servers) or a new session key distribution is mostly a principal task. Basically, they are most often used as vital elements of large systems, such as widely used communication protocols. Vital examples of such systems are Kerberos, SSL/TLS and Zfone. However in literature and applications, errors in the protocol's design can be found. That is why the methods of specifying and verifying their properties are developed to find faults which allow unwanted behavior and can lower the security level or even question it completely.

For automatic or semi-automatic verification systems of protocols, not only modeling protocol's executions are required, but also modeling of the user's

knowledge gained during the system performance. The full and formal description of the protocol is needed as a base for a formal verification. However, the Common Language does not fulfill this requirement. Hence, specifying of the internal and external protocol's actions are required. The methodology suggested in the following sections of this paper fulfills this requirement.

The same as communicating protocols there are several approaches to verifying of time (in)dependent security protocols. We can verify the protocols using the deductive methods (e.g., theorem proving) or algorithmic methods. The deductive methods have been exploited in many verification systems such as: Isabelle, TAPS, PVS, and NRL [6, 8, 12, 22]. The algorithmic approaches mainly consist of methods based on model checking, which have been the aim of an intensive research for several years in both academic and commercial institutions.

Intuitively, the model checking of a security protocol consists of checking if a model of the protocol accepts an execution that represents an attack upon the protocol. In comparison with the standard model checking methods for communicating protocols or for distributed systems, the main trouble is the need to model both the intruder who is responsible for generating attacks as well as changes of knowledge (about keys, nonces, etc.) of the participants.

We can define the properties expressing correctness of security protocols as reachability properties or in linear (branching) time temporal logic. Following the early achievements in model checking of cryptographic protocols by the teams of E. Clarke, C. Meadows, G. Lowe, or D. Bolignano, there has been the state-of-the-art verification system AVISPA [2] designed and implemented as the result of the EU research project. The AVISPA is composed of the following four self-complementing modules: *OFMC* applying symbolic verification on-the-fly via analysis of a transition system described in the specification language IF, *CL-AtSe* using 'constrain solving' and enables discovering of the type flaws, *SATMC* being a bounded model checker exploiting a SAT-solver and *TA4SP* applying a method based on regular tree languages and term rewriting.

On the other hand, verification systems for distributed and real time systems such as SMV, Spin, KRONOS, UppAal [1], or Verics [16] represent much longer history and experience in use. It is clearly very interesting to investigate the methods of applying the above tools to the verification of security protocols.

In this paper we present a new idea of verification of security protocols by using the chains for modeling separately the steps of many different executions of the protocol. Thanks to that we get a very distributed representation of the protocol executions, which is important for an efficient model checking. For the above we use a syntax and semantics of security protocols introduced in paper [18] and later reformulated to verification of timed protocols in [19] and [20].

In the following, a method is given for representing the executions of a security protocol (within a computational structure for a bounded number of sessions) by the chains of states and it shows how the attacks on authentication and security can be found.Our model allows for specification and verification of untimed cryptographic protocols realizing the well-known *challenge-response* idea. The Needham-Schroeder public key protocol [23] is the best known ex-

ample here, however NSPK-Lowe, Untimed WMF, Andrew, TMN, Otway-Rees, and Yahalom [7] are more complicated problems.

Our model of the Intruder's behavior follows the well known Dolev-Yao model [11] however our Intruder can get letters sent only to him. A limitation of the Intruder is visible because we need to find attacks in a more efficient way as the size of a state space is then very limited.

The rest of the paper is organized as follows. In the second section we show the NSPK protocol as the basic example of protocols. In the third section we introduce a formal language that allows you to intuitively describe in detail the steps of security protocols. In the next section we present a new, simple, mathematical model representing executions of protocol's steps (introducing the Intruder). Finally, we show the experimental results for a few protocols.

2 The Needham Schroeder Public Key Protocol

If we want to understand the problems of design and the practical use of protocols in this section, there is one example of them presented: the Needham Schroeder Public Key Authentication Protocol (NSPK). The protocol is written in the so called Common Language – a protocol's specification language widely used in the literature. Unfortunately, as we can see, this language does not fully describe the protocol. A scheme of sending messages during the protocol's execution and constructing the messages is presented, i.e., the external actions of the protocol. The description of the internal actions and the other conditions of the protocol's execution is usually written in a natural language.

Now, we show one of the protocols proposed in [23]. In the notation presented below, the symbols $i(A)$ and $i(B)$ denote the identifiers of the users A and B, which want to communicate safely with each other. By $\langle X \rangle_{K_A}$ we mean a ciphertext containing a message X encrypted with the public key of the user A. Analogously, by $\langle X \rangle_{K_B}$ we mean a ciphertext encrypted by B's public key.

The user A initiates the protocol. The objective, which should be achieved after the protocol's execution, is a mutual authentication of the parties A and B, i.e., the mutual confirmation of their identities. The notation $A \rightarrow B : X$ means that the message X is transmitted from A to B. In our approach it is often assumed that sending information implies its receipt by the receiver. The notation $X \cdot Y$ means concatenation of X and Y.

Example 1. The paper [23] proposed a protocol with public keys in a server environment. Nowadays, when the Public Key Infrastructure is widely used, NSPK without key servers is particularly interesting. The scheme of this version of the protocol is the following:

$$\begin{aligned} &\alpha_1 \quad A \rightarrow B \;:\; \langle N_A \cdot i(A) \rangle_{K_B}, \\ &\alpha_2 \quad B \rightarrow A \;:\; \langle N_A \cdot N_B \rangle_{K_A}, \\ &\alpha_3 \quad A \rightarrow B \;:\; \langle N_B \rangle_{K_B}. \end{aligned} \tag{1}$$

In the first step of the protocol, the user A generates its random number N_A and sends it together with its identifier $i(A)$ to B, encrypting everything with the public key belonging to B. In the second step, B generates its own number N_B and encrypts it together with the obtained number N_A with A's public key.

Then, B sends the so prepared ciphertext to A, which can decrypt it, A compares the number received from B with its nonce N_A and at this moment A acknowledges B to be authenticated, since B sent to A the nonce N_A. A encrypts this number with B's public key and sends it to B. B decrypts the ciphertext (only it is able to do that) and compares the number sent by A with its own N_B. At this moment A is authenticated by B. Due to the structure of transferring the data used in the protocol and properties of the asymmetric cryptography, the users A and B should be sure of their identity after executing the protocol.

A very interesting and instructive history of this protocol has been shown and used in practice in its original version for 17 years. However, in 1995 it turned out that the protocol can be broken. Gavin Lowe presented an attack upon the protocol in [21]. The attack is performed by the Intruder denoted by ι. The Intruder is a user of the computer network which has its own identifier and asymmetric keys. However, ι does not need to execute the protocol according to its scheme. In many ways it may cheat other parties communicating with them.

Example 2. Below we present the attack given by Lowe.

$$\begin{array}{lll} \alpha_1^1 & A \to \iota & : \langle N_A \cdot i(A)\rangle_{K_\iota}, \\ & \alpha_1^2 \ \iota(A) \to B & : \langle N_A \cdot i(A)\rangle_{K_B}, \\ & \alpha_2^2 \ B \to \iota(A) & : \langle N_A \cdot N_B\rangle_{K_A}, \\ \alpha_2^1 & \iota \to A & : \langle N_A \cdot N_B\rangle_{K_A}, \\ \alpha_3^1 & A \to \iota & : \langle N_B\rangle_{K_\iota}, \\ & \alpha_3^2 \ \iota(A) \to B & : \langle N_B\rangle_{K_B}. \end{array} \tag{2}$$

The above scheme shows two simultaneous runs of the protocol. The execution α^1 corresponds to communication of the user A with the Intruder ι, who impersonates A (we denote it by $\iota(A)$) during the execution α^2.

The second step of the fixed version of the protocol, also developed by Lowe, is as follows:

$$\alpha_2 \ B \to A \ : \langle N_A \cdot N_B \cdot i(B)\rangle_{K_A}, \tag{3}$$

Simply adding the responder's identifier $i(B)$ to the ciphertext $\langle N_A \cdot N_B \cdot i(B)\rangle_{K_A}$ excludes the possibility of its deceptive use by the potential Intruder.

3 Formal model of executions of the protocol

A security protocol can be described intuitively without a formal definition. There is one of the common methods to write such a protocol in the so-called

Common Language, in which there are external protocol's actions and this is the scheme of sending messages. Unfortunately this is not a complete description of the protocol, since it is necessary to add the information about the internal actions such as generating new confidential information or the process of the message encryption/decryption. A formal definition which provides all the information about the protocol can be found in the papers [18–20]. In these works the protocol is defined as an abstract object (algorithm) and is differed from the execution. The executions of the protocol are certain substitutions of the abstract protocol in a defined structure which reflects the actual work of the network [18–20]. Note that in such approach we can also consider the execution, which cannot be done alone (see *Example 3*). Such execution is treated as the hypothetical execution that may be made as parts of the interleaves of several executions which are assembled as the attack. In this paper we use intuitively described executions of the protocol as these substitutions.

Example 3. We consider previously discussed Needham Schroeder protocol. Below there are two different hypothetical executions. In the first one, the honest users B and A communicate with each other. In the second: the user A communicates with the Intruder ι who impersonates the user B. Such behavior of the Intruder is marked by $\iota(B)$.

Note that the second execution cannot be done alone. However, it can be considered as a potential part of the interleave of several executions are assembled as the attack (see *Example* 2).

$$\begin{array}{lcll} \alpha_1^3 & = & B \to A: & \langle N_B \cdot i(B)\rangle_{K_A}, \\ \alpha_2^3 & = & A \to B: & \langle N_B \cdot N_A\rangle_{K_B}, \\ \alpha_3^3 & = & B \to A: & \langle N_A\rangle_{K_A}, \end{array} \qquad \begin{array}{lcll} \alpha_1^4 & = & A \to \iota(B): & \langle N_A \cdot i(A)\rangle_{K_B}, \\ \alpha_2^4 & = & \iota(B) \to A: & \langle N_A \cdot N_\iota\rangle_{K_A}, \\ \alpha_3^4 & = & A \to \iota(B): & \langle N_\iota\rangle_{K_B}. \end{array}$$

The basic notations used in the following part of the paper are given below. We begin with defining the following sets:

- $\mathbf{P} = \{P_1, P_2, \ldots, P_{n_P}\}$ - a set of the honest participants in the network,
- $\mathbf{P}_\iota = \{\iota, \iota(P_1), \iota(P_2), \ldots, \iota(P_{n_P})\}$ - a set of the dishonest participants containing the Intruder and the Intruder impersonating the participant P_i for $1 \leq i \leq n_P$,
- $\mathbf{I} = \{i(P_1), \ldots, i(P_{n_P}), i_\iota\}$ - a set of the identifiers of the participants in the network,
- $\mathbf{K} = \bigcup_{i=1}^{n_P}\{K_{P_i}, K_{P_i}^{-1}\} \cup \{K_\iota, K_\iota^{-1}\}$ - a set of the public and private cryptographic keys (already existing or possible to be generated) of the participants,
- $\mathbf{N} = \bigcup_{i=1}^{N_P}\{N_{P_i}^1, \ldots, N_{P_i}^{k_N}\} \cup \{N_\iota^1, \ldots, N_\iota^{k_N}\}$ - a set of the *nonces*[1].

By *the set of letters* $\mathbf{L}$ we mean the smallest set fulfilling the following conditions:

1. $\mathbf{P} \cup \mathbf{P}_\iota \cup \mathbf{I} \cup \mathbf{K} \cup \mathbf{N} \subseteq \mathbf{L}$,

[1] As before, we assume that N_P and K_N are some fixed natural numbers. For simplicity, we take the same number of nonces for each user.

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.

The same as in the papers [18, 19] the protocol Π is a sequence of steps defined as ordered five-tuples $\alpha = (P, Q, M, G, K)$. In such step P is the step initiator (sending part), Q is a message recipient, M is a sent message, G is a set of information required in order to be generated by P for the execution of the step α and K is a set of information required for P in order to send M. Assume the following notation: if $\alpha = (P, Q, M, G, K)$, then by $Send(\alpha)$, $Rec(\alpha)$, $Mess(\alpha)$, $Gen(\alpha)$, $Know(\alpha)$ we mean the following elements: P, Q, M, G, K. Consider the following example:

Example 4. One of the execution of the protocol NSPK described previously can be defined as a sequence of three steps:

$$\begin{aligned}
\alpha_1 &= (A, B, \langle N_A \cdot i(A) \rangle_{K_B}, \{N_A\}, \{i(A), N_A, K_B\}),\\
\alpha_2 &= (B, A, \langle N_A \cdot N_B \rangle_{K_A}, \{N_B\}, \{N_A, N_B, K_A\}),\\
\alpha_3 &= (A, B, \langle N_B \rangle_{K_B}, \emptyset, \{N_B, K_B\}).
\end{aligned}$$

In this way we can describe any number of executions of the tested protocol. During the verification the considered space is limited to two honest users and the Intruder (more information can be obtained in [18, 19]).

The intuition of the following definition is as follows: constructed runs correspond with different true interleaves of the protocol. Due to the fact that in reality, the ability to execute the steps of the protocol is dependent on the knowledge of the individual members in order to define the conditions which enable to construct the necessary runs, the knowledge of the users will be needed during the execution of the protocol.

Consider the following finite sequence of the execution of the protocol's steps: $\mathcal{R} = \alpha_{k_1}^{i_1}, \alpha_{k_2}^{i_2}, \alpha_{k_3}^{i_3}, \ldots, \alpha_{k_s}^{i_s}$, where in denoting a step α the superscript indicates the number of execution, and the subscript indicates the number of the step in the given execution.

Example 5. If we consider the two different executions of the same protocol, which consist of three steps, $\alpha_1^1, \alpha_2^1, \alpha_3^1$ and $\alpha_1^2, \alpha_2^2, \alpha_3^2$, the possible sequence is: $\mathcal{R} = \alpha_1^1, \alpha_2^1, \alpha_1^2, \alpha_3^1, \alpha_2^2, \alpha_3^2$. Observe that in such sequences it is necessary to keep the chronology of the execution of each step, i.e., no step of the execution can be found in the sequence before the preceding step in the protocol.

For each such sequence $\mathcal{R}$ we define the set sequence of the knowledge of users who take part in the executions which constitute the $\mathcal{R}$. The initiators of given steps α are denoted by $Send(\alpha)$, and the recipients of the information sent during the step's execution α by $Rec(\alpha)$. Denote by $\mathbf{U}$ the set of all the users involved in the steps which constitute the sequence $\mathcal{R}$.

We have: $\mathbf{U} = \bigcup_{j=1,\ldots,s}\{Send(\alpha_{k_j}^{i_j})\} \cup \bigcup_{j=1,\ldots,s}\{Rec(\alpha_{k_j}^{i_j})\}$. Assume that each user has a basic knowledge and it means that the user knows his own encryption keys (public and private), user IDs and public keys of the users.

In the following definition we will use the operator κ,[2] that means to acquire the knowledge of the encrypted messages. This acquisition is of course dependent on the possession of the appropriate encryption keys.

Next definition models the knowledge of the users during the execution of the protocol's steps, taking into account the increase of the knowledge through the messages and their contents (if the user has the appropriate encryption keys for the decryption) and the generated information. Defined sequences are the ascending sequences.

Consider any $j = 1, \ldots s-1$ for any sequence $\mathcal{R} = \alpha_{k_1}^{i_1}, \alpha_{k_2}^{i_2}, \alpha_{k_3}^{i_3}, \ldots, \alpha_{k_s}^{i_s}$. For every user $p \in \mathbf{P}$ we have $Know_p^1 = \bigcup_{s=1}^{n_P} \{i(P_s), K_{P_s}\} \cup \{\iota, K_\iota, K_p^{-1}\}$ and:

$$Know_p^{j+1} = \begin{cases} Know_p^j \ \cup \ Gen(\alpha_{k_{j+1}}^{i_{j+1}}) & \text{if} \quad p = Send(\alpha_{k_{j+1}}^{i_{j+1}}), \\ \kappa(Know_p^j \cup \{Mess(\alpha_{k_{j+1}}^{i_{j+1}}))\}) & \text{if} \quad p = Resp(\alpha_{k_{j+1}}^{i_{j+1}}), \\ Know_p^j & \text{otherwise.} \end{cases}$$

Definition 1. *By the run we call any finite sequence of the steps of protocol's executions which fulfills the following conditions:* $\mathfrak{r} = \alpha_{k_1}^{i_1}, \alpha_{k_2}^{i_2}, \alpha_{k_3}^{i_3}, \ldots, \alpha_{k_s}^{i_s}$ *that meets the following conditions:*

1. $\forall_{j=1,\ldots,s}[k_j = 1 \vee \exists_{t<j}(i_t = i_j \ \wedge \ k_t = k_j - 1)]$,
2. $\forall_{j=2,\ldots,s}[Know(\alpha_{k_j}^{i_j}) \subseteq Know^{j-1}_{Send(\alpha_{k_j}^{i_j})} \cup Gen(\alpha_{k_j}^{i_j}))]$.

The first condition defines the appropriate dependence between the subsequent steps of the same execution of the protocol which are in the given run. The second condition ensures the proper dependence between the knowledge of the users who send messages in the next steps - parts of the run.

This definition allows to express the actual interleaves of different protocol's execution which reflects the actual executions. For such runs in the fixed base space we define the chains encoding below.

4 Chains of states representing executions of protocol's steps

In the paper [18], a mathematical model of a protocol's executions (correct runs) is translated into a network of synchronous automata. The runs were expressed as the computations in this network. The security problems were modeled as a problem of the reachability of desired (unsecured) states in the network. The computations in the network were subsequently encoded as propositional boolean formulas. SAT-solvers answered the question whether a valuation fulfilling the formula exists , and therefore whether an attack on the protocol exists. In the paper [18], several experimental results for untimed protocols were given. In some

[2] The formal definition of this operator can be found in the paper [19].

cases they were better than those obtained during verification with the AVISPA Tool.

Now, we introduce a new mathematical model of the protocol's executions. The double translation of the mathematical model of executions proposed in [18] brought along some redundancies, of which the following approach is free. The obtained and described experimental results in the last section of the article are very good, and are promising for the later development of this approach.

In the proposed model, we code each step of every execution of a protocol as a simple chain of the states representing individual actions carried out during the step, or the conditions for making them possible.

We distinguish four types of states:

1. states representing the execution of steps, those will be labeled S_j^i to mark the execution of i-th step in the j-th execution,
2. states representing the generation of new confidential information by users (nonces, keys) labeled onwards e.g. $G_A^{N_A}$ (nonce N_A generated by user A),
3. states representing the obtaining of knowledge by receivers of steps of protocol K_A^X - user A acquired message X,
4. states representing the requirement of possessing a given knowledge element necessary for executing a given step described as e.g. P_A^X (- user A has to possess knowledge of element X in order to carry out a given step).

As you can easily notice, for execution of NSPK gave in *Example 1* we have the three following communication states: S_1^1, S_2^1, S_3^1. We present the chains of the states representing the executions of individual steps of the discussed protocol below:

$$\alpha_1^1 = (G_A^{N_A}, S_1^1, K_B^{N_A}), \quad \alpha_2^1 = (P_B^{N_A}, G_B^{N_B}, S_2^1, K_A^{N_B}), \quad \alpha_3^1 = (P_A^{N_B}, S_3^1). \tag{4}$$

The set of the states preceding the state corresponding with the execution of the steps S will be marked hereinafter by $PreCond(S)$. Accordingly, by using $PostCond(S)$ we will mark the set of the states found in the sequence after the state S. Taking into consideration the above, we can model the execution of the protocol NSPK as the following sequence of the states.

$$NSPK = \alpha_1^1, \alpha_2^1, \alpha_3^1 = (G_A^{N_A}, S_1^1, K_B^{N_A}, P_B^{N_A}, G_B^{N_B}, S_2^1, K_A^{N_B}, P_A^{N_B}, S_3^1). \tag{5}$$

Observe that additionally we have:
$PreCond(S_2^1) = \{P_B^{N_A}, G_B^{N_B}\}$ and $PostCond(S_2^1) = \{K_A^{N_B}\}$.

4.1 Infrastructure with Intruder

The definitions introduced in the third section do not include the presence of the Intruder in the network. As we wrote in the introduction we can consider its various models. In this paper we explore some version of the Dolev-Yao model, so we assume that there is only one Intruder, who actively tries to deceive the others by executing the protocols against their assumptions. The Intruder may

use any information obtained from a network, and impersonate other users. Note that the Intruder can compose sent messages contrary to the assumptions of the protocol. This can be done in many ways. Consider the following example.

Example 6. If the sent message is ciphertext $\langle N_A\rangle_{K_A} \cdot \langle N_B\rangle_{K_B}$, the message can be composed in five ways. In each case the Intruder can compose and use during the execution of the protocol the message: $X_1 = \{\langle N_A\rangle_{K_A} \cdot \langle N_B\rangle_{K_B}\}$, $X_2 = \{\langle N_A\rangle_{K_A}, \langle N_B\rangle_{K_B}\}$, $X_3 = \{N_A, K_A, \langle N_B\rangle_{K_B}\}$, $X_4 = \{\langle N_A\rangle_{K_A}, N_B, K_B\}$, $X_5 = \{N_A, K_A, N_B, K_B\}$. In each case the Intruder can compose and use during the execution of the protocol the message $\langle N_A\rangle_{K_A}, \langle N_B\rangle_{K_B}$. The fact, that from a given set X a message M can be composed, is denoted by $X \vdash M$.[3]

To express this we need to slightly modify the definition of the execution of protocol's step where the initiator of this step is an Intruder. In the definition of the execution's step we introduce the following change:

Definition 2. *If* $Send(\alpha) = \iota$, *then* $\alpha = (\iota, Q, M, \emptyset, \{X \mid X \vdash M\})$.

In the case of a given execution of the protocol's step, the Intruder is the sending message party ($Send(\alpha^{i_j}_{k_j}) \in \mathbf{P}_\iota$) we have to adapt the knowledge condition of the run definition in the following manner. If $Know(\alpha^{i_j}_{k_j}) = \{X \mid X \vdash Mess(\alpha^{i_j}_{k_j})\}$, then, the Condition 2 of the definition 1 takes the form::

$$\exists_{X \in Know(\alpha^{i_j}_{k_j})}[X \subseteq Know^{j-1}_{Send(\alpha^{i_j}_{k_j})} \cup Gen(\alpha^{i_j}_{k_j})].$$

This condition says that in order to execute each step and send a message to the Intruder it is enough to compose a sent message in one of the possible ways.

The corresponding chains for the *Example 2* take the following form:

$$\begin{array}{ll} \alpha^1_1 = (G^{N_A}_A, S^1_1, K^{N_A}_\iota), & \alpha^2_1 = (P^{N_A}_\iota, S^2_1, K^{N_A}_B), \\ \alpha^1_2 = (P^{\langle N_A \cdot N_B\rangle_{K_A}}_\iota, S^1_2, K^{N_B}_A), & \alpha^2_2 = (P^{N_A}_B, G^{N_B}_B, S^2_2, K^{\langle N_A \cdot N_B\rangle_{K_A}}_\iota), \\ \alpha^1_3 = (P^{N_B}_A, S^1_3, K^{N_B}_\iota), & \alpha^2_3 = (P^{N_B}_\iota, S^2_3). \end{array}$$

4.2 Correct chains of states and new approach for verification

Now we proceed to define the sequences consisting of the states defined above that represent true executions of the protocols in the computer networks. This definition will refer to the definition of the runs of the third section.

Let Π be the base space consisting of the users and their attributes (identifiers, nonces, cryptographic keys, etc..). We have to consider all the executions of the protocol in this space and all state's chains for all executions. Under the set of all these chains we define a correct chain of states that represents the real executions of the protocol in the network.

Definition 3. *We call the sequence of the protocol's states:* $\mathfrak{s} = s_1, s_2, \ldots$ **a correct chain of states** *iff the following conditions holds:*

[3] The formal definition can be found in the work [19].

1. *if* $s_i = S_j^k$ *for some* j,k *then* $j = 1 \vee \exists_{t<i}(s_t = S_{j-1}^k)$ *and* $PreCond(S_j^k) \subseteq \{s_1, \ldots, s_{i-1}\} \ \wedge \ PostCond(S_j^k) \subseteq \{s_{i+1}, \ldots\}$,
2. *if* $s_i = G_U^X$, *then* $\forall_{t \neq i}(s_t \neq G_U^X)$,
3. *if* $s_i = P_U^X$, *then* $\exists_{t<i}(s_t = G_U^X \ \vee \ s_t = K_U^X)$.

As you can see the definition of the considered chains is based on the definition of runs.

The first point guarantees a suitable dependence of the order of carrying out the individual steps of a given execution. Points 2 and 3 guarantee a proper dependence of the users' knowledge necessary to execute the individual steps.

Our new proposed concept of verification is as follows. The input data consists of the generated set of chains representing the steps of the executions in the given, investigated space of executions. On an empty stack we firstly put a chain that represents one of the initial steps of executions (chains inserted into stack are labeled as explored). Next we build the limited tree of chains by adding only those unused chains conditions of which match the conditions introduced in Definition 3. Simultaneous creating and searching through the tree ends up in two cases. Firstly if we find the path of the attack, which is derived to the output. Secondly if we utilize chains, then we return the output information about the lack of the attack. Although the time complexity of the algorithm is exponential due to the number of executions and steps, we have obtained very good experimental results as the constructed chains are very short. The space complexity is linear due to the parameters mentioned above. As the results presented below show, it seems that currently the presented approach is one of the most efficient.

5 Experimental results

To the verification we have modeled and verified three protocols: NSPK, NSPKL, and the untimed version of the WMF Protocol. The experimental results obtained by us are very promising. In all cases, the times of the encoding as well as of the direct verification of the obtained model are smaller than 1 ms. The attention needs to be drawn to the fact that the hitherto methods dealt differently with the verification of the secure and unsecured protocols. In most cases the verification of the secure protocols are quicker. In the case of the VerICS tool [16, 20] the quicker results of the unsecured protocols were obtained. In both of these cases so far, our solution has been giving the results just as quickly. It is also worth underlining that in our approach the verification using a backward induction (backtracking) may be performed just as easily. In this case, the experimental results, however, do not differ from the forward method of verification.

In the published results, a division of the times of the verification into the encoding time and the time of the direct verification of the encoded model is performed. The best results of verification with the AVISPA tool (graph planning) are close to zero (less than 10 ms). However, in the process of the automatic verification the coding time needs to be noted as well. In this case, the times are approximately several-several hundred milliseconds.

We have compared our results to the best ones known to us obtained from the AVISPA ([2, 3]) and VerICS Tool [16, 20]. The table below shows that in all cases, when it comes to coding as well as verifying, our method in the case of verified protocols is better.

Obviously, many more experiments need to be carried out to fully compare our method with the AVISPA, VerICS or the other tools. The research in this area is still in the making, and in the near future we expect the results for other protocols, and the optimization of our method as well as the implementation.

The computer used to perform the experiments was equipped with the processor Intel Pentium D (3000 MHz), 2 GB main memory, and the operating system Linux. In the table, we marked the time of the model's encoding with the **EnT**. The **SolT** stands for the time of its direct verification. The **NA** means we do not have the access to the verification times of the protocol. All the times shown below are expressed in milliseconds.

Protocol	AVISPA		VerICS		Chains	
	EnT (ms)	SolT (ms)	EnT (ms)	SolT (ms)	EnT (ms)	SolT (ms)
NSPK	90	<10	<1	36	<1	<1
NSPK_{Lowe}	90	<10	<1	960	<1	<1
UnT_WMF	NA	NA	<1	32	<1	<1

Table 1. Comparison of experimental results.

6 Conclusions

In the article, we have presented a new efficient approach for the security protocols properties' verification, based on the structures of chains describing actions carried out by participants of protocols, as well as changes of their states of knowledge. The method has been implemented. The obtained results are very promising: the method for protocols NSPK, NSPKL as well as the untimed version of WMF Protocol gives better results than the known verification tools: the AVISPA and VerICS. A research on further optimization of the method and its implementation, as well as its application for other protocols is in progress. Also, the adaptation of the method for time dependent protocols has been planned.

References

1. Amnell, T. et all: Uppaal - Now, Next, and Future, Proc. of the 4th Summer School 'Modelling and Verification of Parallel Processes' (MOVEP'00), LNCS, 2067, 99–124, Springer-Verlag, 2001.
2. Armando, A. et all: The AVISPA tool for the automated validation of internet security protocols and applications. In Proc. of 17th Int. Conf. on Computer Aided Verification (CAV'05), vol. 3576 of LNCS, pp. 281–285. Springer, 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. Basin, D. A., Wolff, B. (editors): Theorem Proving in Higher Order Logics, 16th International Conference, TPHOLs 2003, Roma, Italy, September 8-12, 2003, Proceedings, volume 2758 of Lecture Notes in Computer Science. Springer, 2003.

5. Bella, G., Massacci, F., and Paulson, L. C.: Verifying the set registration protocols. IEEE Journal on Selected Areas in Communications, 20(1):77–87, 2003.
6. Bella G, Paulson L.C.: Using Isabelle to prove properties of the kerberos authentication system. In H. Orman and C. Meadows, editors, *Proc. of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.
7. Burrows, M., Abadi, M., and Needham, R. M.: A logic of authentication. ACM Trans. Comput. Syst., 8(1):18–36, 1990.
8. E. Cohen. Taps: A first-order verifier for cryptographic protocols. In *CSFW '00: Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW'00)*, page 144, Washington, DC, USA, 2000. IEEE Computer Society.
9. Corin, R., Etalle, S., Hartel, P. H., and Mader, A.: Timed model checking of security protocols. In Proc. of the 2004 ACM Workshop on Formal Methods in Security Engineering (FMSE'04), pages 23–32. ACM, 2004.
10. Delzanno, G., and Ganty, P.: Automatic verification of time sensitive cryptographic protocols.In Proc. of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04), volume 2988 of LNCS, pages 342–356. Springer, 2004.
11. Dolev, D. and Yao, A.: On the security of public key protocols.IEEE Transactions on Information Theory, 29(2):198–207, 1983.
12. Evans, N., and Schneider, S.: Analysing time dependent security properties in CSP using PVS. In Proc. of the 6th European Symposium on Research in Computer Security (ESORICS'00), vol. 1895 of LNCS, pp. 222–237. Springer, 2000.
13. Gorrieri, R., Locatelli, E., and Martinelli, F.: A simple language for real-time cryptographic protocol analysis. In Proc. of the 12th European Symposium on Programming (ESOP'03), volume 2618 of LNCS, pages 114–128. Springer, 2003.
14. Jakubowska, G., and Penczek, W.: Is your security protocol on time? In Proc. of FSEN'07, volume 4767 of LNCS, pages 65–80. Springer-Verlag, 2007.
15. Jakubowska, G., and Penczek, W., and 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'05), pages 100–115. Warsaw University, 2005.
16. Kacprzak, M., et. all : Verics 2007 - a model checker for knowledge and real-time. Fundam. Inform., 85(1-4):313–328, 2008
17. Kurkowski, M., Srebrny, M.: A Quantifier-free First-order Knowledge Logic of Authentication, Fund. Inform., vol. 72, pp. 263-282, IOS Press 2006.
18. Kurkowski, M., Penczek, W.: Verifying Security Protocols Modeled by Networks of Automata, Fund. Inform., Vol. 79 (3-4), pp. 453-471, IOS Press 2007.
19. Kurkowski, M., Penczek, W.: Verifying Timed Security Protocols via Translation to Timed Automata, Fund. Inform., vol. 93 (1-3), pp. 245-259, IOS Press 2009.
20. Kurkowski M., Penczek W.: Applying Timed Automata to Model Checking of Security Protocols, in ed. J. Wang, Handbook of Finite State Based Models and Applications, pp. 223–254, CRC Press, Boca Raton, USA, 2012.
21. Lowe, G.: Breaking and Fixing the Needham-Schroeder Public-key Protocol Using fdr., In TACAS, LNCS, Springer, 147-166, 1996
22. Meadows C.: The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):13–131, 1996.
23. Needham, R.M., Schroeder, M.D.: Using encryption for authentication in large networks of computers. Commun. ACM, 21(12), 993-999, 1978.
24. Paulson, L. C.: Inductive analysis of the internet protocol tls. ACM Trans. Inf. Syst. Secur., 2(3):332–351, 1999.