SAT-Based Verification of Security Protocols Via Translation to Networks of Automata [⋆]

Mirosław Kurkowski[1], Wojciech Penczek[2,3], and Andrzej Zbrzezny[1]

[1] Institute of Mathematics and Computer Science, Jan Długosz University, Armii Krajowej 13/15, 42-200 Częstochowa
m.kurkowski@ajd.czest.pl, a.zbrzezny@ajd.czest.pl

[2] Institute of Computer Science, PAS, Ordona 21, 01-237 Warsaw, Poland
penczek@ipipan.waw.pl

[3] Institute of Informatics, Podlasie Academy, Sienkiewicza 51, 08-110 Siedlce, Poland

Abstract. In this paper we show a novel method for modelling behaviours of security protocols using networks of communicating automata in order to verify them with SAT-based bounded model checking. These automata correspond to executions of the participants as well as to their knowledge about letters. Given a bounded number of sessions, we can verify both correctness or incorrectness of a security protocol proving either reachability or unreachability of an undesired state. We exemplify all our notions on the Needham Schroeder Public Key Authentication Protocol (NSPK) and show experimental results for checking authentication using the verification tool VerICS.

Keywords: security protocols, model checking, authentication.

1 Introduction

Security protocols define the rules of exchanging messages between the parties in order to establish a secure communication channel between them. Similarly to communicating protocols there are several approaches to verification of security protocols. These protocols are usually verified using deductive methods (e.g., theorem proving) or algorithmic ones. Deductive methods have been exploited in many verification systems like: Isabelle [2], Murϕ [26], TAPS [9], PVS [13], and NRL [25]. Algorithmic approaches include mainly methods based on model checking, which have been an object of an intensive research for several years in both academic and commercial institutions.

Intuitively, model checking of a security protocol consists in checking whether a model of the protocol accepts an execution (or contains a reachable state) that is representing an attack on the protocol. Comparing to standard model checking methods for communicating protocols or for distributed systems, the main difficulty is caused by the need to model both the intruder who is responsible

[⋆] The authors acknowledge partial support from the Ministry of Science and Information Society Technologies under the grant number 3 T11C 01128.

for generating attacks as well as changes of knowledge (about keys, nonces, etc.) of the participants. Typically, a model is constructed as a product of processes representing the participants and the intruder.

Properties expressing correctness of security protocols are usually formulated as reachability properties or in linear (branching) time temporal logic. Following early achievements in model checking of cryptographic protocols by the teams of E. Clarke [8], C. Meadows [25], G. Lowe [23], or D. Bolignano [4], over the last five years the state-of-the-art verification system AVISPA [1] has been designed and implemented as the result of the EU research project. 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 type flaws, *SATMC* being a bounded model checker exploiting a SAT-solver, and *TA4SP* applying a method based on regular tree languages and therm rewriting.

On the other hand, verification systems for distributed and real time systems like SMV [24], Spin [14], KRONOS [29], UppAal [3], or Verics [11] enjoy a much longer history and experience in use. It is clearly very interesting to investigate methods of applying the above tools to verification of security protocols [17,28,10,16,15]. In this paper we are interested in using tools that accept inputs represented by networks of (timed) automata as these can be then verified with most of the existing symbolic and non-symbolic model checkers. Verification can be performed either indirectly by specifying a protocol in a higher order language and then translating it to automata or directly by modelling a protocol by a network of automata. In this paper[1] we offer a new syntax and semantics of security protocols, and an entirely novel approach to their verification (to the best of our knowledge). Our main and original idea about consists in using networks of automata for modelling separately the participants and their knowledge about secrets. Thanks to that we get a very distributed representation of the protocol executions, which is important for an efficient symbolic encoding and model checking. To this aim we develop a novel semantics of security protocols, where the notion of a computational structure and an interpretation is based on the ideas that appeared in [18,5]. Next, we give a method for representing the executions of a security protocol (within a computational structure for a bounded number of sessions) by the runs of the product automaton of a network of the above mentioned automata and show how to look for attacks on authentication. To this aim we use Bounded Model Checking (BMC), which consists in translating the problem of reachability in the product automaton to satisfiability of some propositional formula. Moreover, in addition to prove reachability of an undesired state (in case there is an attack on the protocol), we can also prove unreachability of such a state if there is no attack in the computational structure. This seems to be as well a novel application of BMC to verification of security protocols.

[1] Some preliminary results [19] were presented at CS&P'06.

Our model allows for specification and verification of untimed cryptographic protocols which realise the well-known *challenge-response* idea. The Needham-Schroeder public key protocol [6] is the best known example here, but there are other more complicated protocols like NSPK-Lowe, Andrew, TMN, Otway-Rees, and Yahalom [6,7] that fall into that class as well. In this paper, for simplicity of a translation to automata, we assume that letters sent in the executions of the protocols cannot include nested ciphers. We focus on the public key cryptography, but it is easy to observe that this model is adequate in the case of symmetric cryptography too. Our model of the Intruder's behaviour follows the well known Dolev-Yao model [12] in which the Intruder can intercept and modify all the letters. However, in our experiments we are dealing with a limited model of the Intruder in which he can only receive letters sent to him when playing the role of himself or impersonating another participant. This limitation allows to look for attacks in a more efficient way as the size of a state space is then much limited.

The rest of the paper is organised as follows. In Section 2 we introduce syntax for dealing with untimed security protocols. A computational structure generating all the runs of the protocols considered is defined in Section 3. A method for finding attacks by analysing computations of the protocol is shown in Section 4. Section 5 defines network of automata for representing the participants of a protocol and their knowledge about secrets. Then, experimental results are given in Section 6 and some concluding remarks in Section 7.

2 Syntax of Security Protocols

In this section we introduce syntax for dealing with untimed security protocols. To this aim, we give some notations used in the rest of the paper. Next, we define the following basic syntactic notions of our model.

- $\mathcal{T}_P = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{n_P}\}$ is a set of symbols representing the users of the computer network,
  $\mathcal{T}_I = \{\mathcal{I}_{\mathcal{P}_1}, \mathcal{I}_{\mathcal{P}_2}, \ldots, \mathcal{I}_{\mathcal{P}_{n_P}}\}$ is a set of symbols representing the identifiers of the users,
- $\mathcal{T}_K = \bigcup_{i=1}^{n_P} \{\mathcal{K}_{\mathcal{P}_i}, \mathcal{K}_{\mathcal{P}_i}^{-1}\}$ is a set of symbols representing the cryptographic keys of users (public and private respectively),
- $\mathcal{T}_N = \bigcup_{i=1}^{n_P} \{\mathcal{N}_{\mathcal{P}_i}^{1}, \ldots, \mathcal{N}_{\mathcal{P}_i}^{n_N}\}$[2] is a set of symbols representing the users' *nonces*,
- {"(",")","{","}",","} is a set of the auxiliary symbols.

Definition 1 (Letter Terms). *By a set of* letter terms $\mathcal{T}$ *we mean the smallest set satisfying the following conditions:*

1. $\mathcal{T}_P \cup \mathcal{T}_I \cup \mathcal{T}_K \cup \mathcal{T}_N \subseteq \mathcal{T}$.
2. *If* $X \in \mathcal{T}$ *and* $Y \in \mathcal{T}$, *then the concatenation* $X \cdot Y \in \mathcal{T}$,

[2] We assume that n_P and n_N are some fixed natural numbers.

3. *If $X \in \mathcal{T}$ and $\mathcal{K} \in \mathcal{T}_{\mathcal{K}}$, then $\langle X \rangle_{\mathcal{K}} \in \mathcal{T}$*[3].

Next, we define some useful relations over the set $\mathcal{T}$.

Definition 2. *Let $\prec\!\cdot_{\mathcal{T}} \subseteq \mathcal{T} \times \mathcal{T}$ be the smallest relation (called* (immediate) subterm relation*), which satisfies the following conditions:*

1. *If $X, Y \in \mathcal{T}$, then $X \prec\!\cdot_{\mathcal{T}} X \cdot Y$ and $Y \prec\!\cdot_{\mathcal{T}} X \cdot Y$,*
2. *If $X \in \mathcal{T}$ and $\mathcal{K} \in \mathcal{T}_{\mathcal{K}}$, then $X \prec\!\cdot_{\mathcal{T}} \langle X \rangle_{\mathcal{K}}$ and $\mathcal{K} \prec\!\cdot_{\mathcal{T}} \langle X \rangle_{\mathcal{K}}$.*

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

– $\mathcal{X}^0 \overset{def}{=} \mathcal{X}$,
– $\mathcal{X}^{n+1} \overset{def}{=} \mathcal{X}^n \cup \{Z \in \mathcal{T} \mid (\exists X, Y \in \mathcal{X}^n,\ \mathcal{K} \in \mathcal{X} \cap \mathcal{T}_{\mathcal{K}})\ Z = X \cdot Y \vee Z = \langle X \rangle_{\mathcal{K}}\}$.

Intuitively, the set $\mathcal{X}^{n+1}$ contains the, gradually built, letter terms from $\mathcal{X}^n$ using the operations of composition and encryption. In what follows, for any set Z by 2^Z_{fin} we denote a set of all the finite subsets of Z.

For $\mathcal{X} \in 2^{\mathcal{T}}_{fin}$ the set $Comp(\mathcal{X}) \overset{def}{=} \bigcup_{n \in \mathbf{N}} \mathcal{X}^n$ is composed of all the letters that can be constructed out of elements of $\mathcal{X}$ only[4].

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

Definition 3. *By a* (protocol) step α *we mean a five-tuple $(\mathcal{P}, \mathcal{X}, \mathcal{G}, \mathcal{Q}, \mathcal{L}) \in \mathcal{T}_P \times 2^{\mathcal{T}}_{fin} \times 2^{\mathcal{T}_{\mathcal{K}} \cup \mathcal{T}_N}_{fin} \times \mathcal{T}_P \times \mathcal{T}$, with the following intuitive meaning:*
$\mathcal{P}$ - the sender of the step,
$\mathcal{X}$ - the set of letters necessary to compose $\mathcal{L}$,
$\mathcal{G}$ - the set of generated secrets necessary to compose $\mathcal{L}$,
$\mathcal{Q}$ - the receiver of $\mathcal{L}$, and
$\mathcal{L}$ - the letter sent from $\mathcal{P}$ to $\mathcal{Q}$,
that satisfies the following conditions:

1. *$\mathcal{P} \neq \mathcal{Q}$ (nobody can send letters to himself),*
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.

[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] Description is not allowed here.

Example 1. We consider Needham Schroeder Public Key Authentication Protocol (NSPK) as a working example. Below, syntax of NSPK is defined. $\mathcal{T}_P = \{\mathcal{A}, \mathcal{B}\}$, $\mathcal{T}_I = \{\mathcal{I}_\mathcal{A}, \mathcal{I}_\mathcal{B}\}$, $\mathcal{T}_K = \{\mathcal{K}_\mathcal{A}, \mathcal{K}_\mathcal{B}\}$, $\mathcal{T}_N = \{\mathcal{N}_\mathcal{A}, \mathcal{N}_\mathcal{B}\}$. The protocol NSPK is given by the following sequence of steps: $(\alpha_1, \alpha_2, \alpha_3)$, where:
$\alpha_1 = (\mathcal{A}, \{\mathcal{N}_\mathcal{A}, \mathcal{I}_\mathcal{A}, \mathcal{K}_\mathcal{B}\}, \{\mathcal{N}_\mathcal{A}\}, \mathcal{B}, \langle \mathcal{N}_\mathcal{A}, \mathcal{I}_\mathcal{A} \rangle_{\mathcal{K}_\mathcal{B}})$,
$\alpha_2 = (\mathcal{B}, \{\mathcal{N}_\mathcal{A}, \mathcal{N}_\mathcal{B}, \mathcal{K}_\mathcal{A}\}, \{\mathcal{N}_\mathcal{B}\}, \mathcal{A}, \langle \mathcal{N}_\mathcal{A}, \mathcal{N}_\mathcal{B} \rangle_{\mathcal{K}_\mathcal{A}})$,
$\alpha_3 = (\mathcal{A}, \{\mathcal{N}_\mathcal{B}, \mathcal{K}_\mathcal{B}\}, \emptyset, \mathcal{B}, \langle \mathcal{N}_\mathcal{B} \rangle_{\mathcal{K}_\mathcal{B}})$. $\square$

3 Computational Structure

In this section we define a computational structure generating all the computations (under the interpretations considered) of an authentication protocol investigated. Later, we aim at representing these computations by runs of some network of automata. In general, we could deal with an infinite number of sessions in a computational structure, but because we aim at verifying our protocols in an automatic way, we restrict ourselves to a bounded number of sessions by limiting the number of nonces. We start 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 $1 \le i \le n_p$,
- $\mathbf{I} = \{i_{p_1}, \ldots, i_{p_{n_p}}, i_\iota\}$ - 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 cryptographic keys 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*[5].

Definition 4. *By* a set of letters $\mathbf{L}$ *we mean the smallest set satisfying the following conditions:*

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

Next, we define some auxiliary relations over the set $\mathbf{L}$.

Definition 5. *Let* $\prec \subseteq \mathbf{L} \times \mathbf{L}$ *be the smallest relation (called* (immediate) subletter relation*) satisfying the following conditions:*

1. *If* $x, y \in \mathbf{L}$, *then* $x \prec x \cdot y$ *and* $y \prec x \cdot y$,
2. *If* $x \in \mathbf{L}$ *and* $k \in \mathbf{K}$, *then* $x \prec \langle x \rangle_k$ *and* $k \prec \langle x \rangle_k$.

[5] 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.

By $\preceq$ we denote the transitive and reflexive closure of $\prec$. Next, for any $X \subseteq \mathbf{L}$ we define a sequence of the sets $(X^n)_{n\in\mathbf{N}}$ that are also subsets o $\mathbf{L}$, where

- $X^0 \overset{def}{=} X$,
- $X^{n+1} \overset{def}{=} X^n \cup \{z \in \mathbf{L} \mid (\exists x, y \in X^n,\ k \in X \cap \mathbf{K})\ z = x \cdot y\ \vee\ z = \langle x \rangle_k\}$.

The intuition behind this definition is the same as for the corresponding one in Section 2, i.e., the set X^{n+1} contains the, gradually built, letters from X^n using the operations of composition and encryption. Next, define the set $Comp(X) \overset{def}{=} \bigcup_{n\in\mathbf{N}} X^n$ which consists of all the letters that can be composed out of elements of X only[6] and the set $Sublet(X) \overset{def}{=} \{l \in \mathbf{L} \mid (\exists x \in X)\ l \preceq x\}$ which contains all the subletters of X.

Definition 6. *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 interpretations of the terms of $\mathcal{T}$. Each interpretation determines one execution of the protocol (defined as a syntactical object).

Definition 7. *By an interpretation of the set of the letter terms $\mathcal{T}$ we mean any injection $f : \mathcal{T} \to \mathbf{L}$ satisfying the following conditions:*

1. *$f(\mathcal{T}_P) \subseteq \mathbf{P} \cup \mathbf{P}_\iota$, $f(\mathcal{T}_I) \subseteq \mathbf{I}$, $f(\mathcal{T}_K) \subseteq \mathbf{K}$, $f(\mathcal{T}_N) \subseteq \mathbf{N}$,*
2. *$(\forall X, Y \in \mathcal{T})\ f(X \cdot Y) = f(X) \cdot f(Y)$ (homomorphism),*
3. *$(\forall X \in \mathcal{T})(\forall \mathcal{K} \in \mathcal{T}_K)\ f(\langle X \rangle_{\mathcal{K}}) = \langle f(X) \rangle_{f(\mathcal{K})}$ (homomorphism),*
4. *If $f(\mathcal{P}) = p$ for $p \in \mathbf{P}$, then $f(\mathcal{I}_\mathcal{P}) = i_p$, $f(\mathcal{N}_\mathcal{P}) \in \{n_p^1, \ldots, n_p^{k_S}\}$, $f(\mathcal{K}_\mathcal{P}) = k_p$ and $f(\mathcal{K}_\mathcal{P}^{-1}) = k_p^{-1}$.*
5. *If $f(\mathcal{P}) = \iota$, then $f(\mathcal{I}_\mathcal{P}) = i_\iota$, $f(\mathcal{K}_\mathcal{P}) = k_\iota$ and $f(\mathcal{K}_\mathcal{P}^{-1}) = k_\iota^{-1}$.*
6. *If $f(\mathcal{P}) = \iota(p)$, then $f(\mathcal{I}_\mathcal{P}) = i_p$, $f(\mathcal{K}_\mathcal{P}) = k_p$ and $f(\mathcal{K}_\mathcal{P}^{-1}) = k_p^{-1}$,*
7. *$f(\mathcal{T}_P) \setminus \mathbf{P}_\iota \neq \emptyset$*

The condition 1 states that the atomic terms are mapped into corresponding objects of the computational structure, i.e., symbols representing participants are mapped into participants, etc. The condition 2 and 3 guarantee the homomorphical separation between the mapped symbols. The condition 4 says that the symbols related to a given participant are mapped into corresponding objects (identifiers, keys, nonces) in the structure. The conditions 5-7 are related to our model of the Intruder. The condition 5 determines that if the Intruder

[6] Description is not allowed here.

wants to play in an execution of the protocol the role of himself, then he uses his own identifier and keys. There is no condition on the nonces used by the Intruder, as we assume that he can use any nonce. The condition 6 states that if the 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.

In order to define later an interpretation of a protocol step in which the 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, we have $X \vdash l$ if all the elements of X are subletters of l, l can be composed out of the elements of X, and X is a minimal such a set.

Example 2. Consider the letter $l = \langle i_a, n_a \rangle_{k_b}$. Observe that the sets $X_1 = \{i_a, n_a, k_b\}$ and $X_2 = \{\langle i_a, n_a \rangle_{k_b}\}$ are sets of independent generators of l, i.e., we have $X_1 \vdash l$ and $X_2 \vdash l$.

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

Definition 9. *Consider a step $\alpha = (\mathcal{P}, \mathcal{X}, \mathcal{G}, \mathcal{Q}, \mathcal{L})$ of a given protocol Σ and an interpretation f of $\mathcal{T}$. By the* f-interpretation *of the step α (denoted by $f(\alpha)$) we mean the following five-tuple:*

– $(f(\mathcal{P}), f(\mathcal{X}), f(\mathcal{G}), f(\mathcal{Q}), f(\mathcal{L})^{7})$*, if* $f(\mathcal{P}) \in \mathbf{P}$,
– $(f(\mathcal{P}), \{X \mid X \vdash f(\mathcal{L})\}, \emptyset, f(\mathcal{Q}), f(\mathcal{L}))$*, if* $f(\mathcal{P}) \in \mathbf{P}_\iota$.

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

By *the execution of a protocol* $\Sigma = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ under an interpretation f we mean the sequence $f(\Sigma) = (f(\alpha_1), f(\alpha_2), \ldots, f(\alpha_n))$.

Example 3. Again, we exemplify the above notions on NSPK. $\mathbf{P} = \{a, b\}$, $\mathbf{P}_\iota = \{\iota, \iota(a), \iota(b)\}$, $\mathbf{I} = \{i_a, i_b, i_\iota\}$, $\mathbf{K} = \{k_a, k_b, k_\iota\}$, $\mathbf{N} = \{n_a, n_b, n_\iota\}$. Consider the interpretation f_1 defined as follows: $f_1(\mathcal{A}) = a$, $f_1(\mathcal{B}) = b$, $f_1(\mathcal{I_A}) = i_a$, $f_1(\mathcal{I_B}) = i_b$, $f_1(\mathcal{N_A}) = n_a$, $f_1(\mathcal{N_B}) = n_b$, $f_1(\mathcal{K_A}) = k_a$, $f_1(\mathcal{K_B}) = k_b$. We have the following execution of NSPK: $(f_1(\alpha_1), f_1(\alpha_2), f_1(\alpha_3))$, where:

[7] We assume that any private key cannot be an element of the contents of $f(\mathcal{L})$.

– $f_1(\alpha_1) = (a, \{n_a, i_a, k_b\}, \{n_a\}, b, \langle n_a, i_a\rangle_{k_b})$,
– $f_1(\alpha_2) = (b, \{n_a, n_b, k_a\}, \{n_b\}, a, \langle n_a, n_b\rangle_{k_a})$,
– $f_1(\alpha_3) = (a, \{n_b, k_b\}, \emptyset, b, \langle n_b\rangle_{k_b})$. □

In order to define knowledge of the participants and the Intruder we need to introduce the following auxiliary notions. If $f(\alpha_i) = (p, X, G, q, l)$, 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:
$Send^{f(\alpha_i)} = p$ (the sender of $f(\alpha_i)$),
$Lett^{f(\alpha_i)} = l$ (the letter of $f(\alpha_i)$),
$Gen^{f(\alpha_i)} = G$ (the set of generated new secrets in $f(\alpha_i)$),
$Resp^{f(\alpha_i)} = q$ (the responder of $f(\alpha_i)$), and
$Part^{f(\alpha_i)} = \{Send^{f(\alpha_i)}, Resp^{f(\alpha_i)}\}$.
Additionally 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 which generate $Lett^{f(\alpha_i)}$).

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 $p \in \bigcup_{f\in\mathcal{F}} f(\mathcal{T}_P) \setminus \mathbf{P}_\iota$ (the Intruder ι, resp.) needs to compose all the letters sent in an execution under any interpretation $f \in \mathcal{F}$.

Definition 10. *The set* $Comp^p_{\mathcal{F}} = \bigcup_{1\le i\le n} \bigcup_{\{f\in\mathcal{F} \mid Send^{f(\alpha_i)}=p\}} Comp^{f(\alpha_i)}$ *for an honest user* p *is the union of all the sets* $Comp^{f(\alpha_i)}$ *for all* $i \le n$ *and* $f \in \mathcal{F}$, *where* $Send^{f(\alpha_i)} = p$.

The set $Comp^\iota_{\mathcal{F}} = \bigcup_{1\le i\le n} \bigcup_{\{f\in\mathcal{F} \mid Send^{f(\alpha_i)}\in\mathbf{P}_\iota\}} Comp^{f(\alpha_i)}$ *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 11. *For an honest participant* $p \in \bigcup_{f\in\mathcal{F}} f(\mathcal{T}_P) \setminus \mathbf{P}_\iota$ *his knowledge at the step* j *is given inductively as follows:*
$\kappa^0_p = \mathbf{I} \cup \{k^{-1}_p\} \cup \{k_q \mid q \in \mathbf{P}\} \cup \{k_\iota\}$,

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

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 $\mathcal{F}$.

We define two models of the Intruder's knowledge. The first one is the full Dolev-Yao model the Intruder's knowledge, whereas the second model restricts the Intruder such that if he is not the responder of a letter, then his knowledge does not change.

Definition 12. *The Intruder's 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:*

$\kappa^0_\iota = \mathbf{I} \ \cup \ \{k^{-1}_\iota, k_\iota\} \ \cup \ \{k_q \ \mid \ q \in \mathbf{P}\} \cup \{n^1_\iota, \ldots, n^{k_N}_\iota\}$,
For the D-Y model of the Intruder:
$\kappa^{j+1}_\iota = Comp^\iota_{\mathcal{F}} \cap \xi_{\{k^{-1}_\iota\}}(\kappa^j_\iota \cup \{Lett^{f^{j+1}(\alpha_{i_{j+1}})}\})$.
For the restricted model of the Intruder:

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

Notice the Intruder is retrieving all the possible letters from his knowledge and the letter he has interecepted (received), which is 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 which guarantee that a sequence of protocol step interpretations is a computation of the protocol.

Definition 13. *By a computation of the protocol Σ we mean any injective 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. $(\forall k \in \mathbf{N}_+)[i_k > 1 \ \Rightarrow \ (\exists j < k)(\ f^j = f^k \wedge \ i_j = i_k - 1)]$,
2. $(\forall k, j \in \mathbf{N}_+)[k \neq j \ \Rightarrow \ Gen^{f^k(\alpha_{i_k})} \cap Gen^{f^j(\alpha_{i_j})} = \emptyset]$,
3. $(\forall j \in \mathbf{N}_+)[Lett^{f^j(\alpha_{i_j})} \in Comp(\kappa^{j-1}_{Send^{f^j(\alpha_{i_j})}} \cup Gen^{f^j(\alpha_{i_j})})]$.

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 $j-1$.

4 Attacks Upon Protocols

Security protocols are used in order to establish a secure communication channel between two parties involved in the communication. This is obtained by ensuring that each party is confident about several security properties: e.g., that the other party is who they say they are (*authentication*), a confidential information is not visible to non-authorised parties (*secrecy*), the information exchanged by two parties cannot be altered by an intruder (*integrity*), and finally the parties taking part in the transaction cannot deny it later (*non-repudiation*).

Below, we focus on checking authentication only. We say that a given protocol is *correct* if the protocol cannot be executed in such a way that identifiers or keys of one participant are used by someone else. Having this in mind, we give the following definition.

Definition 14 (Attacking execution). *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 $p \in \mathbf{P}$.*

Example 4. Consider the interpretation f_2 defined as follows: $f_2(\mathcal{A}) = \iota(a)$, $f_2(\mathcal{B}) = b$, $f_2(\mathcal{I_A}) = i_a$, $f_2(\mathcal{I_B}) = i_b$, $f_2(\mathcal{N_A}) = n_a$, $f_2(\mathcal{N_B}) = n_b$, $f_2(\mathcal{K_A}) = k_a$, $f_2(\mathcal{K_B}) = k_b$. We have the following execution of the NSPK protocol: $(f_1(\alpha_1), f_1(\alpha_2), f_1(\alpha_3))$, where:
$f_2(\alpha_1) = (\iota(a), \{X_1, X_2\}, \emptyset, b, \langle n_a, i_a\rangle_{k_b})$,
$f_2(\alpha_2) = (b, \{n_a, n_b, k_a\}, \{n_b\}, \iota(a), \langle n_a, n_b\rangle_{k_a})$,
$f_2(\alpha_3) = (\iota(a), \{X_3, X_4\}, \emptyset, b, \langle n_b\rangle_{k_b})$,
$X_1 = \{n_a, i_a, k_b\}, X_2 = \{\langle n_a, i_a\rangle_{k_b}\}, X_3 = \{n_b, k_b\}, X_4 = \{\langle n_b\rangle_{k_b}\}$. □

Definition 15 (Attack). *By* an attack *upon a protocol we mean any of its computations such that an attacking execution is its subsequence.*

The following example shows an attack on NSPK.

Example 5. Consider the interpretation f_2 of Example 4 and the interpretation f_3 defined below: $f_3(\mathcal{A}) = a$, $f_3(\mathcal{B}) = \iota$, $f_3(\mathcal{I_A}) = i_a$, $f_3(\mathcal{I_B}) = i_\iota$, $f_3(\mathcal{N_A}) = n_a$, $f_3(\mathcal{N_B}) = n_b$, $f_3(\mathcal{K_A}) = k_a$, $f_3(\mathcal{K_B}) = k_\iota$.
For f_3 we have the following execution of NSPK: $(f_3(\alpha_1), f_3(\alpha_2), f_3(\alpha_3))$, where:
$f_3(\alpha_1) = (a, \{n_a, i_a, k_\iota\}, \{n_a\}, \iota, \langle n_a, i_a\rangle_{k_\iota})$.
$f_3(\alpha_2) = (\iota, \{X_5, X_6\}, \emptyset, a, \langle n_a, n_b\rangle_{k_a})$,
$f_3(\alpha_3) = (a, \{n_b, k_\iota\}, \emptyset, \iota, \langle n_b\rangle_{k_\iota})$
with $X_5 = \{n_a, n_b, k_a\}$ and $X_6 = \{\langle n_a, n_b\rangle_{k_a}\}$.
Observe that the sequence $\mathfrak{r} = (f_3(\alpha_1), f_2(\alpha_1), f_2(\alpha_2), f_3(\alpha_2), f_3(\alpha_3), f_2(\alpha_3))$ is a computation [8] which contains an attacking execution. Thus, $\mathfrak{r}$ is an attack.

[8] A simplified notation of this computation is the following:

$$\begin{array}{rcl} a \to \iota & : & \langle n_a, i_a\rangle_{k_\iota}, \\ \iota(a) \to b & : & \langle n_a, i_a\rangle_{k_b}, \\ b \to \iota(a) & : & \langle n_a, n_b\rangle_{k_a}, \\ \iota \to a & : & \langle n_a, n_b\rangle_{k_a}, \\ a \to \iota & : & \langle n_b\rangle_{k_\iota}, \\ \iota(a) \to b & : & \langle n_b\rangle_{k_b}. \end{array}$$

5 Networks of Communicating Automata

In this section we represent the computations of a protocol by runs of a *network of communicating automata*, where each automaton represents one component of the protocol.

Definition 16 (Automaton). *An automaton A_i is a 4-tuple $(\Sigma_i, L_i, s_i^0, T_i)$, where*

- *Σ_i is a finite set of actions,*
- *L_i is a finite set of locations,*
- *$s_i^0 \in L_i$ is the initial location,*
- *$T_i \subseteq L_i \times \Sigma_i \times L_i$ is a transition relation.*

A set of communicating automata can be composed into the global (*product*) automaton by the standard multi-synchronisation approach: the transitions that do not correspond to a shared action are interleaved, whereas the transitions labelled with a shared action are synchronised. Assume a set of n communicating automata $\{A_1, \ldots, A_n\}$ and let $\Sigma(a) = \{1 \leq i \leq n \mid a \in \Sigma_i\}$.

Definition 17 (Product Automaton). *The product automaton of the automata A_i is defined by $\mathcal{A} = (\Sigma, G, s^0, T)$, where:*

- *$\Sigma = \bigcup_{i=1}^n \Sigma_i$ is a finite set of actions,*
- *$G = L_1 \times \ldots \times L_n$ is a finite set of global states,*
- *$s^0 = (s_1^0, \ldots, s_n^0)$ is the initial state,*
- *T is a transition relation, where $((l_1, \ldots, l_n), a, (l_1', \ldots, l_n')) \in T$ iff $\forall_{i \in \Sigma(a)}\ (l_i, a, l_i') \in T_i$ and $\forall_{i \in \{1,\ldots,n\} \setminus \Sigma(a)}\ l_i = l_i'$.*

By a *run* of $\mathcal{A}$ on a word $a_1 \cdots a_n$ we mean a sequence of states $(s_0, \ldots, s_n)$ such that $s_0, \ldots, s_n \in G$, $s_0 = s^0$, and $(s_i, a_i, s_{i+1}) \in T$ for all $1 \leq i \leq n-1$. A state $s \in G$ is *reachable* if there is a run of $\mathcal{A}$ s.t. its final state is equal to s.

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

5.1 Automata for Modelling Executions of the Participants

Assume we are dealing with a protocol $\Sigma = (\alpha_1, \ldots, \alpha_n)$.

Definition 18 (Automaton for execution). *Consider the execution of the protocol Σ under an interpretation f, i.e., $(f(\alpha_1), f(\alpha_2), \ldots, f(\alpha_n))$. This execution is modelled by the automaton $A_f = (\Sigma_f, Q_f, f(\alpha_0), \delta_f)$, where:*
- *$Q_f = \{s_0^f, s_1^f, s_2^f, \ldots, s_n^f\}$ is the set of states, where s_0^f is the initial state,*
- *$\Sigma_f = \{k_{f(\alpha_i)} \mid 1 \leq i \leq n \ \wedge\ Send^{f(\alpha_i)} \in \mathbf{P}\} \cup \bigcup_{i=1}^n \bigcup_{X \subseteq L} \{k_{f(\alpha_i)}^X \mid Send^{f(\alpha_i)} \in \mathbf{P}_\iota \wedge X \vdash Lett^{f(\alpha_i)} \wedge X \neq \{Lett^{f(\alpha_i)}\}\}$,*
- *$\delta_f = \{(s_{i-1}^f, k_{f(\alpha_i)}, s_i^f) \mid 1 \leq i \leq n \wedge k_{f(\alpha_i)} \in \Sigma_f\} \cup \{(s_{i-1}^f, k_{f(\alpha_i)}^X, s_i^f) \mid 1 \leq i \leq n \wedge k_{f(\alpha_i)}^X \in \Sigma_f\}$.*

The intuition behind the above definition is as follows. Each state s_i^f of the automaton is reached after executing the step α_i of the execution (under f) of the protocol. 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 the sender of this step is the Intruder, then there are many possibilities to execute this step determined by the sets of generators of the letter to be sent. Each of these possibilities is labelled with a different label $k^X_{f(\alpha_i)}$.

5.2 Automata for Modelling the Knowledge of the Participants

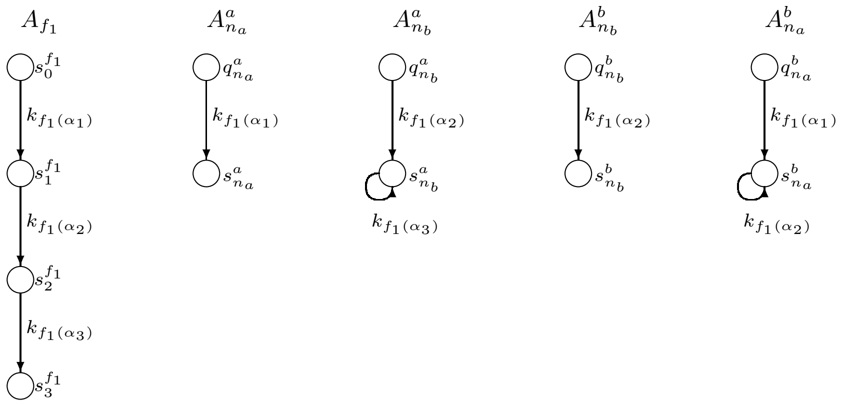
Consider a finite set of protocol interpretations $\mathcal{F}$.

Definition 19 (Automaton for the knowledge of a honest participant) *For each honest participant $p \in (\bigcup_{f\in\mathcal{F}} f(\mathcal{T}_P) \setminus \mathbf{P}_\iota)$ and each element $l \in Comp^p_{\mathcal{F}} \setminus \kappa^0_p$, we define the following (knowledge) automaton $A^p_l = (\Sigma^p_l, Q^p_l, q^p_l, \delta^p_l)$, where*

– $\Sigma^p_l \overset{def}{=} \{k \in \bigcup_{f\in\mathcal{F}} \Sigma_f \mid Cond_1(k) \vee Cond_2(k)\}$ *with*
$Cond_1(k) := (\exists f \in \mathcal{F})(\exists i \leq n)(s^f_{i-1}, k, s^f_i) \in \delta_f \ \wedge$
(i) $\big(p = Send^{f(\alpha_i)} \ \wedge l \in Gen^{f(\alpha_i)}\big) \vee$
(ii) $(p = Resp^{f(\alpha_i)} \ \wedge \ l \in \xi_{\{k_p^{-1}\}}(\{Lett^{f(\alpha_i)}\}) \wedge$
$\wedge \ (\forall j \in \{1,\ldots,i-1\})((p = Resp^{f(\alpha_j)} \Rightarrow l \notin \xi_{\{k_p^{-1}\}}(Lett^{f(\alpha_j)})) \ \wedge$
$\wedge \ (p = Send^{f(\alpha_j)} \Rightarrow l \notin Gen^{f(\alpha_j)}))$
$Cond_2(k) := (\exists f \in \mathcal{F})(\exists i \leq n)(s^f_{i-1}, k, s^f_i) \in \delta_f \ \wedge$
(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*
$(q^p_l, k, s^p_l) \in \delta^p_l$ *iff* $Cond_1(k)$, $(s^p_l, k, s^p_l) \in \delta^p_l$ *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 $f(\alpha_i)$. The condition (ii) says that p learns about l at the step $f(\alpha_i)$. This is modelled only once in order to reduce the number of the transitions. The condition (iii), which defines the loop, enables p to use l while composing new letters. The condition (iv) enables to receive l in a different execution that the one, which was used to define the condition (ii).

Example 6. The network of automata that model the execution and the knowledge of the participants of Example 3 is shown in the figure below.



We give two versions of the automata for the Intruder's knowledge.

Definition 20 (Automaton for the knowledge of the Intruder). *First, we give the automaton corresponding to the full D-Y Intruder's knowledge and then we discuss the modifications for the restricted model.*

The D-Y model of the Intruder: *for each letter*
- $l \in Comp^\iota_{\mathcal{F}} \cap \xi_{\{k_\iota^{-1}\}}(\bigcup_{f\in\mathcal{F}} \bigcup_{i \le n} \{Lett^{f(\alpha_i)}\}) \setminus \kappa^0_\iota$,

we define the knowledge automaton $A^\iota_l = (\Sigma^\iota_l, Q^\iota_l, q^\iota_l, \delta^\iota_l)$, *where*

– $\Sigma^\iota_l \overset{def}{=} \{k \in \bigcup_{f\in\mathcal{F}} \Sigma_f \mid Cond^\iota_1(k) \vee Cond^\iota_2(k)\}$ *with*

$Cond^\iota_1(k) := (\exists f \in \mathcal{F})(\exists i \le n)(s^f_{i-1}, k, s^f_i) \in \delta_f \wedge$

(i) $[(l \in \xi_{\{k_\iota^{-1}\}}(\{Lett^{f(\alpha_i)}\}) \wedge (\forall j \in \{1,\ldots,i-1\})(l \notin \xi_{\{k_\iota^{-1}\}}(Lett^{f(\alpha_j)}))$

$Cond^\iota_2(k) := (\exists f \in \mathcal{F})(\exists i \le n)(s^f_{i-1}, k, s^f_i) \in \delta_f \wedge$

(ii) $(Send^{f(\alpha_i)} \in \mathbf{P}_\iota \wedge (\exists X \subseteq \mathbf{L})(X \vdash Lett^{f(\alpha_i)} \wedge l \in X \wedge k = k^X_{f(\alpha_i)})) \vee$

(iii) $(l \in \xi_{\{k_\iota^{-1}\}}(\{Lett^{f(\alpha_i)}\}) \wedge (\forall j \in \{1,\ldots,i-1\})(l \notin \xi_{\{k_\iota^{-1}\}}(Lett^{f(\alpha_j)})))$.

– $Q^\iota_l = \{q^\iota_l, s^\iota_l\}$ *is the set of states,*

– q^ι_l *is the initial state,*

– δ^ι_l *is the transition relation given as follows:*
$(q^\iota_l, k, s^\iota_l) \in \delta^\iota_l$ *iff* $Cond^\iota_1(k)$, $(s^\iota_l, k, s^\iota_l) \in \delta^\iota_l$ *iff* $Cond^\iota_2(k)$.

The following changes to the above definition are made for the restricted model of the Intruder's knowledge:
- $l \in Comp^\iota_{\mathcal{F}} \setminus \kappa^0_\iota$,

(i) $[(Resp^{f(\alpha_i)} \in \mathbf{P}_\iota \wedge l \in \xi_{\{k_\iota^{-1}\}}(\{Lett^{f(\alpha_i)}\}) \wedge$
$\wedge (\forall j \in \{1,\ldots,i-1\})((Resp^{f(\alpha_j)} \in \mathbf{P}_\iota \Rightarrow l \notin \xi_{\{k_\iota^{-1}\}}(Lett^{f(\alpha_j)}))$,

(iii) $(Resp^{f(\alpha_i)} \in \mathbf{P}_\iota \wedge l \in \xi_{\{k_\iota^{-1}\}}(\{Lett^{f(\alpha_i)}\}) \wedge$
$\wedge (\forall j \in \{1,\ldots,i-1\})(Resp^{f(\alpha_j)} \in \mathbf{P}_\iota \Rightarrow l \notin \xi_{\{k_\iota^{-1}\}}(Lett^{f(\alpha_j)})))$.

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

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

The following theorem says that for each computation in the computation structure there is the corresponding run in the product automaton $\mathcal{A}$ built for this structure and moreover each run of $\mathcal{A}$ 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 is a run in the product automaton $\mathcal{A}$ on a word $(\overline{k_{f^1(\alpha_{i_1})}}, \overline{k_{f^2(\alpha_{i_2})}}, \ldots, \overline{k_{f^k(\alpha_{i_k})}})$, where:*

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

Proof. By induction on the length of a computation (run). Omitted because of the lack of space (see [20] for a proof).

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

6 Experimental Results

We start by describing a SAT-based method of testing reachability for a network of automata, i.e., checking whether a state satisfying certain (usually undesired) property is reachable in the product automaton. For this, assume that φ is a property to be verified. Let $\alpha_k(\varphi)$, for $k \in \mathbb{N}$, be a propositional formula that is satisfiable if and only if there exists a run π of length k such that the property φ holds at some state of π. Moreover, let β_k, for $k \in \mathbb{N}$, be a propositional formula that is satisfiable if and only if there exists a run of length k.

Algorithm 1 searches for the greatest natural number k_0 such that every run is of length less or equal to k_0. Such a number k_0 exists if the set of the reachable states is finite and there are no loops in the set of the reachable states, and this is the case for all the networks of automata considered in this paper. Notice that that if there exists a run π on which φ is reachable, then the length of π has to be less or equal to k_0. Therefore, we can conclude that if the property φ is not reachable on any run of length less or equal to k_0, then it is unreachable.

In Algorithm 1 we use the procedure $checkSat(\gamma)$ that for any given propositional formula γ returns one of the following three possible values: SAT, $UNSAT$, or $UNKNOWN$. The meanings of the values SAT and $UNSAT$ are self-evident. The value $UNKNOWN$ is returned in two cases: either the procedure *checkSat* is not able to decide satisfiability of its argument within some timeout period[9] or it has to terminate due to exhaustion of the available memory.

The above method can be applied to all the networks of automata considered in this paper in view of the fact that there are no loops, at least in the set of the reachable states. We would like to stress that for such networks of automata the method is complete. Another SAT-based method of testing reachability can be found in [30].

Algorithm 1. Algorithm for deciding reachability problem

```
 1: $k \leftarrow 0$
 2: loop
 3:    $result \leftarrow checkSat(\alpha_k(\varphi))$
 4:    if $result = SAT$ then
 5:       return $REACHABLE$
 6:    else if $result = UNKNOWN$ then
 7:       return $UNKNOWN$
 8:    end if
       /* $\alpha_k(\varphi)$ is not satisfiable */
 9:    $k \leftarrow k + 1$
10:    $result \leftarrow checkSat(\beta_k)$
11:    if $result = UNSAT$ then
12:       return $UNREACHABLE$
13:    else if $result = UNKNOWN$ then
14:       return $UNKNOWN$
15:    end if
       /* $\beta_k$ is satisfiable */
16: end loop
```

We have tested the correctness (Definition 15) of the NSPK protocol defined in Example 1. The computational structure (defined in Example 3) is given by 18 automata modelling executions of the principals and 20 knowledge automata (for the restricted model of the Intruder's knowledge). Some of them are shown in Examples 7 and 8. According to Definition 14 there are 4 attacking executions. The experiments consisted in checking reachability (in the product automaton) of the final states of the automata representing these four executions. We have verified that one of these states is reachable at a run of the length 6. This run corresponds to the attack discovered by Lowe [21] (see Example 5).

We have also verified two other protocols:

- an improved version of NSPK, known as the protocol NSPK-Lowe [21],
- an untimed version of the Wide-Mouth Frog Protocol ([6]).

The NSPK-Lowe protocol is defined as: $\Sigma_{NSPK-Lowe} = (\alpha_1, \alpha_2, \alpha_3)$, where:

[9] This is preset in advance.

$\alpha_1 = (\mathcal{A}, \{\mathcal{N}_\mathcal{A}, \mathcal{I}_\mathcal{A}, \mathcal{K}_\mathcal{B}\}, \{\mathcal{N}_\mathcal{A}\}, \mathcal{B}, \langle \mathcal{N}_\mathcal{A}, \mathcal{I}_\mathcal{A} \rangle_{\mathcal{K}_\mathcal{B}})$,
$\alpha_2 = (\mathcal{B}, \{\mathcal{N}_\mathcal{A}, \mathcal{N}_\mathcal{B}, \mathcal{I}_\mathcal{B}, \mathcal{K}_\mathcal{A}\}, \{\mathcal{N}_\mathcal{B}\}, \mathcal{A}, \langle \mathcal{N}_\mathcal{A}, \mathcal{N}_\mathcal{B}, \mathcal{I}_\mathcal{B} \rangle_{\mathcal{K}_\mathcal{A}})$,
$\alpha_3 = (\mathcal{A}, \{\mathcal{N}_\mathcal{B}, \mathcal{K}_\mathcal{B}\}, \emptyset, \mathcal{B}, \langle \mathcal{N}_\mathcal{B} \rangle_{\mathcal{K}_\mathcal{B}})$.

For the same computational structure we have obtained 18 automata for the executions of the participants and 24 knowledge automata. It turned out that the length of the longest possible run is equal to 13. Additionally, it has been verified that no final state of the automata corresponding to the attacking executions is reachable at runs of length up to 13. According to Theorem 1 this proves that in the computational structure considered there is no attack upon the protocol NSPK-Lowe.

We have also investigated the untimed Wide-Mouth Frog Protocol[10]. For the same computational structure we have obtained 15 automata for all the executions and 12 knowledge automata. It turned out that the length of the longest possible run in the network for untimed WMF is equal to 6. Additionally, it has been verified that no final state of the automata corresponding to the attacking executions is reachable at runs of length up to 6. Thus, we conclude that there is no attack in the computational structure considered.

The experimental results are shown in the tables below. The computer used to perform experiments was equipped with the processor Intel Pentium D (3000 MHz), 2 GB main memory, the operating system Linux and the SAT-solver MiniSat.

Table 1. Experimental results for NSPK Protocol

Trace length	Variables	Literals	Clauses	Time (s.)	Result
4	3514	24044	10136	0.040	UNSAT
6	5226	35880	15124	0.036	SAT

Table 2. Experimental results for Lowe's NSPK Protocol

Trace length	Variables	Literals	Clauses	Time (s.)	Result
4	3299	22539	9491	0,052	UNSAT
7	5708	39180	16496	0.300	UNSAT
10	8117	55821	23501	2,140	UNSAT
13	10526	72462	30506	54,85	UNSAT

Table 3. Experimental results for Untimed WMF Protocol

Trace length	Variables	Literals	Clauses	Time (s.)	Result
3	3513	5479	15639	0,43	UNSAT
5	6417	7771	22715	0,86	UNSAT

[10] This protocol is defined as follows:
$\alpha_1 = (\mathcal{A}, \{\mathcal{I}_\mathcal{A}, \mathcal{N}_\mathcal{A}, \mathcal{I}_\mathcal{B}, \mathcal{K}, \mathcal{K}_{\mathcal{AS}}\}, \{\mathcal{N}_\mathcal{A}, \mathcal{K}\}, \mathcal{S}, < \mathcal{N}_\mathcal{A}, \mathcal{I}_\mathcal{B}, \mathcal{K} >_{\mathcal{K}_{\mathcal{AS}}})$,
$\alpha_2 = (\mathcal{S}, \{\mathcal{I}_\mathcal{A}, \mathcal{N}_\mathcal{S}, \mathcal{K}, \mathcal{K}_{\mathcal{BS}}\}, \{\mathcal{N}_\mathcal{S}\}, \mathcal{B}, < \mathcal{N}_\mathcal{S}, \mathcal{I}_\mathcal{A}, \mathcal{K} >_{\mathcal{K}_{\mathcal{BS}}})$.

Table 4. Experimental results from VerICS and SATMC

Tool	Protocol	Time (s)
VerICS	NSPK	0,09
SATMC	NSPK	0,20
VerICS	NSPK-Lowe	0,31
SATMC	NSPK-Lowe	0,27

We have compared[11] our results to these obtained from SATMC[12] of AVISPA ([1]). The results are quite comparable (see the table below), but in our case in addition to finding or not finding attacks, we can also automatically verify with BMC that an attack does not exist at all in the computational structure considered.

In case of verification of NSPK we have got a shorter time whereas for NSPK-Lowe our result is slightly worse (for runs of length 7 as used by SATMC). Clearly, much more experiments need to be made to fully compare our method with AVISPA or other tools. But, these experiments should be conducted only after our implementation has been optimised in order to get an honest comparison.

7 Conclusions and Perspectives

In this paper we have considered attacks on authentication only, but we can easily extend them to attacks on secrecy. In order to prove that an information at is insecure, we have to prove that the state s_l^ι of A_l^ι is reachable.

Our next step is to see what the limits of our method are in terms of the number of sessions as well as in the number of participants for all the protocols which satisfy our restrictions. Then, we are going to relax the assumption on non-nesting ciphers and again conduct experiments with multi-session and multi-user security protocols.

Acknowledgements

The authors wish to thank to Kamil Grondys, Anna Sowik, and Izabela Szczypior for their help and the discussions.

References

1. Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuellar, J., Drielsma, P.H., Heám, P.C., Kouchnarenko, O., Mantovani, J., Mödersheim, S., von Oheimb, D., Rusinowitch, M., Santiago, J., Turuani, M., Viganó, L., Vigneron, L.: 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)

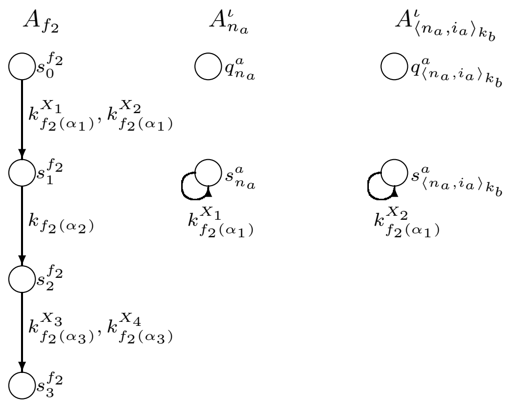
[11] In this experiment we have used the same SAT-solver as SATMC uses, i.e., zChaff.
[12] SATMC is a SAT-based bounded model checker.

2. Bella, G., Paulson, L.C.: Using Isabelle to prove properties of the Kerberos authentication system. In: Orman, H., Meadows, C., (eds). Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols (CD-ROM) (DIMACS'97) (1997)
3. Bengtsson, J., Larsen, K.G., Larsson, F., Pettersson, P., Yi, W., Weise, C.: New generation of Uppaal. In: Proc. of the Int. Workshop on Software Tools for Technology Transfer (STTT'98), BRICS Notes Series, pp. 43–52 (1998)
4. Bolignano, D.: An approach to the formal verification of cryptographic protocols. In: Proceedings of the 3rd ACM Conference on Computer and Communication Security, pp. 106–118 (1996)
5. Bozga, L., Ene, C., Lakhnech, Y.: A Symbolic Decision Procedure for Cryptographic Protocols with Time Stamps. Journal of Logic and Algebraic Programming 65(1), 1–35 (2005)
6. Burrows, M., Abadi, M., Needham, R.: A Logic of Authentication. In: Proceedings of the Royal Society of London A, vol. 426, pp. 233–271 (1989)
7. Clark, J.A., Jacob, J.L.: A survey of authentication protocol literature. Technical Report 1.0, `http://www.cs.york.ac.ukjac/papers/drareview.ps.gz` (1997)
8. Clarke, E.M., Jha, S., Marrero, W.: Verifying security protocols with Brutus. ACM Transactions on Software Engineering and Methodology 9(4), 443–487 (2000)
9. Cohen, E.: Taps: A first-order verifier for cryptographic protocols. In: CSFW '00: Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW'00), pp. 144–158. IEEE Computer Society, Los Alamitos (2000)
10. Corin, R., Etalle, S., Hartel, P.H., Mader, A.: Timed model checking of security protocols. In: Atluri, V., Backes, M., Basin, D., Waidner, M. (eds.) 2nd ACM Workshop on Formal Methods in Security Engineering: From Specifications to Code (FMSE), pp. 23–32. ACM Press, New York (2004)
11. Dembiński, P., Janowska, A., Janowski, P., Penczek, W., Półrola, A., Szreter, M., Woźna, B., Zbrzezny, A.: VerICS: A tool for verifying timed automata and Estelle specifications. In: Garavel, H., Hatcliff, J. (eds.) ETAPS 2003 and TACAS 2003. LNCS, vol. 2619, pp. 278–283. Springer, Heidelberg (2003)
12. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on Information Theory 29(2), 198–208 (1983)
13. 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, pp. 222–237. Springer, Heidelberg (2000)
14. Holzmann, G.J.: The model checker SPIN. IEEE Trans. on Software Eng. 23(5), 279–295 (1997)
15. Jakubowska, G., Penczek, W.: Is Your Security Protocol on Time? In: Proc. of International Symposium on Fundamentals of Software Engineering (FSEN'07) (to appear) (2007)
16. Jakubowska, G., Penczek, W., Srebrny, M.: Verifying security protocols with timestamps via translation to timed automata. In: Czaja, L., (ed). In: Proc. of the Int. Workshop on Concurrency, Specification and Programming (CS&P'05), Warsaw University Press, pp. 100–115 (2005)
17. Jha, S., Clarke, E.M., Marrero, W.: Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In: Gries, D., De Roever, W.P. (eds.) Proceedings of the IFIP Working Conference on Programming Concepts and Methods (PROCOMET'98), pp. 87–106. Chapmann and Hall, Sydney (1998)

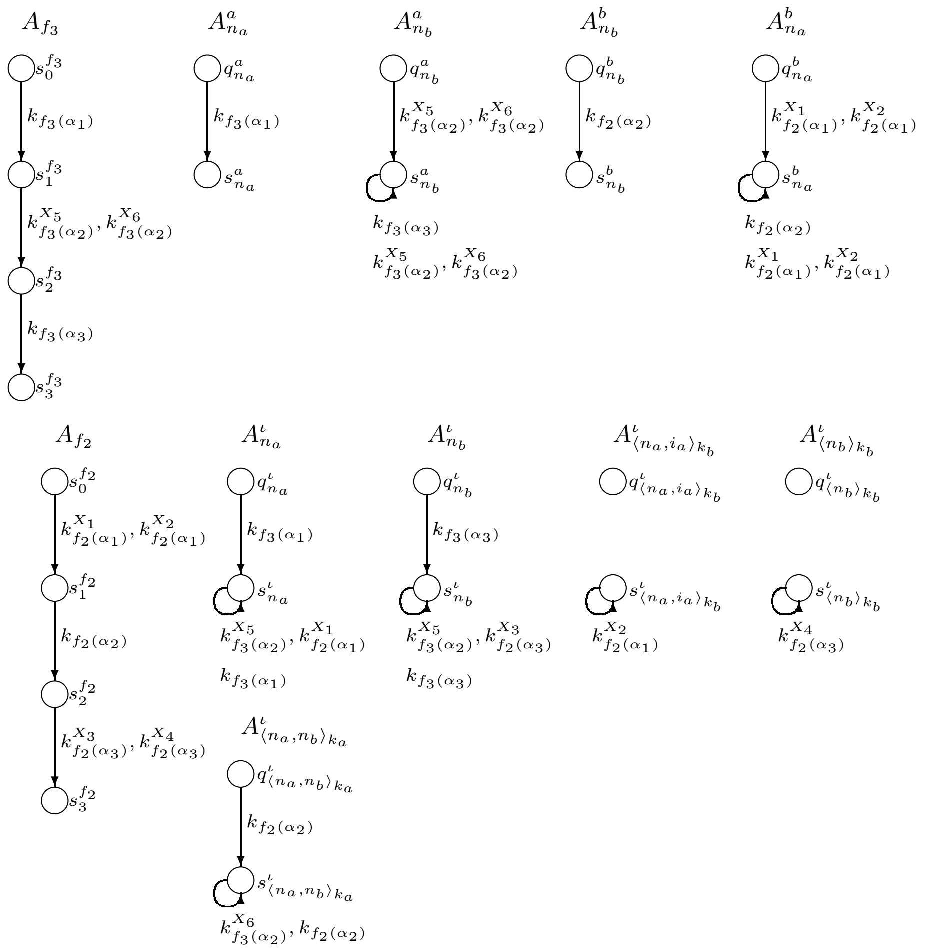
18. Kurkowski, M.: Deductive methods for verification of correctness of the authentication protocols (in polish). PhD thesis, Institute of Computer Science, Polish Academy of Sciences (2003)
19. Kurkowski, M., Penczek, W.: Verifying Cryptographic Protocols modeled by networks of automata. In: Proc. of CS&P'06, Humboldt University Press, pp. 292–303 (2006)
20. Kurkowski, M., Penczek, W.: SAT-based Verification of Security Protocols via Translation to Networks of Automata. Report 998 of ICS PAS, Warsaw, `http://www.imi.ajd.czest.pl/pub/report_kp_07.zip` (2007)
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., Roscoe, B.: Using CSP to Detect Errors in The TMN Protocol. IEEE Transaction on Software Engineering 23(10), 659–669 (1997)
23. Lowe, G.: Casper: A compiler for the analysis of security protocols. Journal of Computer Security 6(1-2), 53–84 (1998)
24. McMillan, K.L.: Symbolic Model Checking. Kluwer Academic Publishers, Dordrecht (1993)
25. Meadows, C.: The NRL protocol analyzer: An overview. Journal of Logic Programming 26(2), 13–131 (1996)
26. Mitchell, M., Mitchell, J.C., Stern, U.: Automated analysis of cryptographic protocols using murϕ. In: Proc. of the 1997 IEEE Symposium on Security and Privacy, pp. 141–151. IEEE Computer Society Press, Los Alamitos (1997)
27. Needham, R., Schroeder, M.: Using Encryption for Authentication in large networks of computers. Communications of the ACM 21(12), 993–999 (1978)
28. Panti, M., Spalazzi, L., Tacconi, S.: Using the NUSMV model checker to verify the Kerberos Protocol. In: Proc. of the Third Collaborative Technologies Symposium (CTS-02), pp. 27–31 (2002)
29. Yovine, S.: KRONOS: A verification tool for real-time systems. International Journal of Software Tools for Technology Transfer 1(1/2), 123–133 (1997)
30. Zbrzezny, A.: SAT-based reachability checking for timed automata with diagonal constraints. Fundamenta Informaticae 67(1-3), 303–322 (2005)

Appendix

Example 7. A part of the network of automata that model the execution and the knowledge of the participants of Example 4 is shown in the figure below.


A_{f_2}
$A^{\iota}_{n_a}$
$A^{\iota}_{\langle n_a, i_a \rangle_{k_b}}$
$s_0^{f_2}$
$q^a_{n_a}$
$q^a_{\langle n_a, i_a \rangle_{k_b}}$
$k^{X_1}_{f_2(\alpha_1)}, k^{X_2}_{f_2(\alpha_1)}$
$s_1^{f_2}$
$s^a_{n_a}$
$s^a_{\langle n_a, i_a \rangle_{k_b}}$
$k_{f_2(\alpha_2)}$
$k^{X_1}_{f_2(\alpha_1)}$
$k^{X_2}_{f_2(\alpha_1)}$
$s_2^{f_2}$
$k^{X_3}_{f_2(\alpha_3)}, k^{X_4}_{f_2(\alpha_3)}$
$s_3^{f_2}$

Example 8. The full network of automata that model both the executions f_2 and f_3 is shown in the figure below.


A_{f_3}
$s_0^{f_3}$
$k_{f_3(\alpha_1)}$
$s_1^{f_3}$
$k^{X_5}_{f_3(\alpha_2)}, k^{X_6}_{f_3(\alpha_2)}$
$s_2^{f_3}$
$k_{f_3(\alpha_3)}$
$s_3^{f_3}$
$A^a_{n_a}$
$q^a_{n_a}$
$k_{f_3(\alpha_1)}$
$s^a_{n_a}$
$A^a_{n_b}$
$q^a_{n_b}$
$k^{X_5}_{f_3(\alpha_2)}, k^{X_6}_{f_3(\alpha_2)}$
$s^a_{n_b}$
$k_{f_3(\alpha_3)}$
$k^{X_5}_{f_3(\alpha_2)}, k^{X_6}_{f_3(\alpha_2)}$
$A^b_{n_b}$
$q^b_{n_b}$
$k_{f_2(\alpha_2)}$
$s^b_{n_b}$
$A^b_{n_a}$
$q^b_{n_a}$
$k^{X_1}_{f_2(\alpha_1)}, k^{X_2}_{f_2(\alpha_1)}$
$s^b_{n_a}$
$k_{f_2(\alpha_2)}$
$k^{X_1}_{f_2(\alpha_1)}, k^{X_2}_{f_2(\alpha_1)}$
A_{f_2}
$s_0^{f_2}$
$k^{X_1}_{f_2(\alpha_1)}, k^{X_2}_{f_2(\alpha_1)}$
$s_1^{f_2}$
$k_{f_2(\alpha_2)}$
$s_2^{f_2}$
$k^{X_3}_{f_2(\alpha_3)}, k^{X_4}_{f_2(\alpha_3)}$
$s_3^{f_2}$
$A^\iota_{n_a}$
$q^\iota_{n_a}$
$k_{f_3(\alpha_1)}$
$s^\iota_{n_a}$
$k^{X_5}_{f_3(\alpha_2)}, k^{X_1}_{f_2(\alpha_1)}$
$k_{f_3(\alpha_1)}$
$A^\iota_{n_b}$
$q^\iota_{n_b}$
$k_{f_3(\alpha_3)}$
$s^\iota_{n_b}$
$k^{X_5}_{f_3(\alpha_2)}, k^{X_3}_{f_2(\alpha_3)}$
$k_{f_3(\alpha_3)}$
$A^\iota_{\langle n_a,i_a\rangle_{k_b}}$
$q^\iota_{\langle n_a,i_a\rangle_{k_b}}$
$s^\iota_{\langle n_a,i_a\rangle_{k_b}}$
$k^{X_2}_{f_2(\alpha_1)}$
$A^\iota_{\langle n_b\rangle_{k_b}}$
$q^\iota_{\langle n_b\rangle_{k_b}}$
$s^\iota_{\langle n_b\rangle_{k_b}}$
$k^{X_4}_{f_2(\alpha_3)}$
$A^\iota_{\langle n_a,n_b\rangle_{k_a}}$
$q^\iota_{\langle n_a,n_b\rangle_{k_a}}$
$k_{f_2(\alpha_2)}$
$s^\iota_{\langle n_a,n_b\rangle_{k_a}}$
$k^{X_6}_{f_3(\alpha_2)}, k_{f_2(\alpha_2)}$

□