# Some decidability result for logic constructed for checking user authentication protocols

László Aszalós[*] and Philippe Balbiani[**]
[*] Department of Computer Science,
University of Debrecen, Faculty of Informatics,
PoBox 12, 4010 Debrecen, Hungary, aszalos@inf.unideb.hu

[**] Institut de recherche en informatique de Toulouse
Irit-CNRS, Université Paul Sabatier
118 route de Narbonne, 31062 Toulouse Cedex 4, France balbiani@irit.fr

***Abstract*** ***–The core of our paper is a general purpose logical system for reasoning about user authentication protocols. Proposed as an extension of the propositional epistemic logic by dynamic operators, the potential usefulness of our calculus for protocol verification is illustrated with examples.***

***Keywords:*** ***authentication protocol, modal logic, dynamic logic***

## I. INTRODUCTION

To protect data from exposure, it is desirable to encipher plaintext information under keys which allow users to send and receive messages over an insecure network. As a result, the users must find secure methods for exchanging the keys either by themselves or by means of a system key manager. Authentication protocols emerged from numerous works of computer scientists and their use has become common in the science and study of methods of exchanging keys. They are basically sequences of message exchanges, whose purpose is to assure users that communications do not leak confidential data. Indeed, there is a wide variety of protocols that have been specified and implemented, from protocols with trusted third party, to protocols with public key and, even more generally, hybrid protocols. The one drawback is that many of them have been shown to be flawed, from which one may explain the great deal of attention devoted to the formal verification of security properties of protocols. Examples of protocols can be found in [3].

In the literature, the most popular logic-based formal approach to the analysis of authentication protocols is perhaps the modal BAN calculus introduced by Burrows, Abadi and Needham [2]. From the point of view of computer science, a virtue of BAN is that it allows static characterization of epistemic concepts. In spite of its success in finding flaws or redundancies in some well-known protocols, the effectiveness of BAN as a formal method for the analysis of authentication protocols has been a source of debate, see [12] for details. The problem with the BAN logic is that it explicitly excludes time. On the other hand there is no way to represent actions performed by users. Communication, by its nature, refers to time, and its properties are naturally expressed in terms of actions like sending and receiving messages. When devising a protocol, we usually think of some property that we want the protocol to satisfy. We are mainly interested in the correctness of a protocol with respect to epistemic properties between two users like the arranging of a secret key known only to them. Therefore, our emphasis is on the interplay between knowledge and action. This leads us to consider a language that allows to express notions of knowledge and actions in a straightforward way: the language of modal logic.

Many-dimensional modal logics are logics arising from the study of formal languages that are capable of characterizing different aspects of a domain, from time, to space, and, even more generally, intensional concepts like knowledge, action, obligation, etc. They form a part of the field of modal logic and have applications in artificial intelligence and computer science. Their study can be found in [1,6,7,11]. To illustrate the truth of this, many-dimensional modal logics allow an intuitive and attractive approach to the analysis of the behavior of multi-agent distributed systems by means of a formalism containing both epistemic operators and temporal operators, see [5] for details. This paper uses epistemic operators and dynamic operators to develop a formalized language, focusing on the fundamental notions of knowledge and action. This paper presents what in our opinion constitutes the basis of authentication protocol verification. In section 2, we provide some necessary background on cryptography and data security.
In addition to the basic definitions, we present the Needham-Schroeder public key protocol. A formalized language for the analysis of authentication protocols, proposed as an extension of the propositional epistemic logic by dynamic operators, is introduced in section 3. We also see examples illustrating its potential usefulness for the analysis of how knowledge evolves when protocols are executed. The semantical presentation, based on the notion of a global state is given in section 4. Here we show several

examples of protocol runs. In section 5 we give negative answer for the question of equivalent ordering of different modalities. Next in section 6 we list some problem and solve them. Finally, section 7 concludes the paper through a number of open questions.

## II. CRYPTOGRAPHY AND DATA SECURITY

A cryptosystem or cipher system is a method of disguising messages so that only certain people can see through the disguise. Cryptography is the art of creating and using cryptosystems. The original message is called a plaintext. The disguised message is called a ciphertext. Encryption means any procedure to convert plaintext into ciphertext. Decryption means any procedure to convert ciphertext into plaintext. At symmetric encryption, we use the same key for encryption and decryption. At public key encryption there are pairs of keys. If we use one of the keys to encrypt then we can use the other key to decrypt and vice versa. Usually one of the keys is known only to the owner (private key) and the other is known to everybody (public key). In this article we denote by $k_{AB}$ the symmetric key shared by A and B whereas we denote by $k_A$ and $k_A^{-1}$ the public and private key of user A, respectively. At the public key schemes, the encryption and the description is a very lengthy procedure, whereas symmetric key encryption can be done more efficiently, Hence the public key cryptosystems usually generate symmetric "session" keys and are using this key. We assume that users communicate over a network and hence they need to exchange the session key over the network. At this exchange we require that

- After the exchange the sender and receiver can perform encryption and decryption using the session key.
- Intruders cannot decrypt messages, only the receiver (confidentiality)
- Receiver knows that the message was encrypted by a given entity and not someone else (authentication)
- Intruders cannot modify messages (integrity)

The cryptosystem consists of protocols. One protocol is an ordered list of messages.
Let us see a famous example of a protocol:

- $A \rightarrow B$: $k_B(N_A,A)$
- $B \rightarrow A$: $k_A(N_A,N_B)$
- $A \rightarrow B$: $k_B(N_B)$

Needham-Schroeder public key protocol

Here we have three messages, the sender of the first and last messages is A, and in these cases the receiver is B. At the second message the sender is B, and the receiver is A.
The first message contains the name of the sender and a nonce $N_A$. A nonce is a randomly generated number, and at the successive runs of a protocol nonces never get the same value. (Nonces can help to exclude several flaws, see [4,10].) The first message is encrypted with the public key of user B, so this ciphertext can be opened (decrypt) with the private key of B, and this is the secret of B. Hence only B can acquire the nonce $N_A$. He send back this nonce to prove that he got the message and send a new nonce. This second message is encrypted with public key of A, so only A can decrypt and get the nonce $N_B$. By sending back this nonce A can prove that he got B's message.

## III. A FORMALIZED LANGUAGE

When analyzing protocols run by users over a network that is vulnerable to many attacks, we want to focus on the communication aspects. As a result, the notion of message is basic. Suppose we fix a finite set *USE* of users' names, with typical member denoted *i*. We assume countably infinite sets $VAR_t$, $VAR_k$, $VAR_n$ and $VAR_c$ of text variables, key variables, nonce variables and ciphertext variables, respectively. We usually write text variables as $x^t$, $y^t$, $z^t$, etc. We use suitable superscript for the other types, too. The set of all messages is inductively defined as follows:

$m := i \mid x \mid \langle m_1,m_2\rangle \mid \text{left}(m) \mid \text{right}(m) \mid k_i(m) \mid k_i^{-1}(m) \mid E(x^k,m)$.

If we consider a countably infinite set *Pt* of plaintexts, with typical member denoted *P*, a countably infinite set *Sk* of symmetric keys, with typical member denoted *k*, a countably infinite set *N* of nonces, with typical member denoted *n*, and countably infinite set *Ct* of ciphertext, with typical member denoted *c*, then let $M_0$, $M_1$, … be sets defined as follows: $M_0=Pt\cup Sk\cup N\cup Ct$, $M_{i+1}=M_i\cup M_i\times M_i$ for $i\geq 0$. Let $M=M_0\cup M_1\cup ...$ Now, a model based on *Pt*, *Sk*, *N* and *Ct* is a structure of the following form $\mathbf{M}=(Pt, Sk, N, Ct, \sigma, I_\sigma)$, where $\sigma$ is a substitution of variables and $I_\sigma$ is the function that satisfies the following conditions:

- $I_\sigma(i)=i\in Pt$, $I_\sigma(x_t)=\sigma(x_t)\in Pt$, $I_\sigma(x_k)=\sigma(x_k)\in Sk$, $I_\sigma(x_n)=\sigma(x_n)\in N$, and $I_\sigma(x_c)=\sigma(x_c)\in Ct$.
- $I_\sigma(\langle .,.\rangle):M\times M\rightarrow M$, $I_\sigma(\text{left}):M\rightarrow M$ and $I_\sigma(\text{right}):M\rightarrow M$ are partial functions, such that
  - If $m\in M_0$ then $I_\sigma(\text{left})(m)$ and $I_\sigma(\text{right})(m)$ are undefined.
  - $I_\sigma(\langle .,.\rangle)(m,m') =\langle m,m'\rangle\in M$
  - $I_\sigma(\text{left})(\langle m,m'\rangle)=m$ and $I_\sigma(\text{right})(\langle m,m'\rangle)=m'$.
- $I_\sigma(k_i):M\rightarrow M$ and $I_\sigma(k_i^{-1}):M\rightarrow M$ such that
  - $I_\sigma(k_i)(I_\sigma(k_i^{-1})(m))=m$ and $I_\sigma(k_i^{-1})(I_\sigma(k_i)(m))=m$.
- If $m\in M_0\backslash Ct$ then $I_\sigma(k_i)(m)\in Ct$, $I_\sigma(k_i^{-1})(m)\in Ct$ and $I_\sigma(E)(k,m)\in Ct$,
- $I_\sigma(E): Sk\times M\rightarrow M$ such that $I_\sigma(E)(k,I_\sigma(E)(k,m))=m$,

where $m, m'\in M$ and $k\in Sk$.

During the execution of an authentication protocol like the Needham-Schroeder public-key protocol, interact by making actions. Within our framework, we will have to consider atomic actions like sending and receiving messages. Further actions with several atomic actions in sequence are typically described by means of dynamic constructs like sequential composition, nondeterministic choice, nondeterministic iteration and test. As a result, the set of all actions is inductively defined as follows:

$\alpha$:=$\lambda$|*exp*?|text($x^t$)|key($x^k$)|nonce($x^n$)|send(m)|rec(m)|
$\alpha_1;\alpha_2|\ \alpha_1\cup\alpha_2|\alpha^*$

where *exp* ranges over the set of all expressions to be defined later. $\lambda$ is the nullary action "do nothing". Roughly speaking, expressions allow us to reason about messages, users and the messages that users have. We formally defined them in this section. It must be remarked that programs are formed by starting with tests like *exp*?, atomic actions like text($x^t$), key($x^k$), nonce($x^n$), send($m$) and rec($m$), and closing off under dynamic constructs. With these definitions in hand, we can now define what it means for a user to execute an action:

- When $i$ performs the atomic action text($x^t$), it chooses an element in the plaintext space *Pt* the value of which is given to $x^t$.
- When $i$ executes the atomic action key($x^k$), it picks an element in the key space *Sk* the value of which is given to $x^k$.
- When $i$ does the atomic action nonce($x^n$), it finds an element in the nonce space *N* the value of which is given to $x^n$.
- When $i$ makes the action $\alpha_1;\alpha_2$, it performs $\alpha_1$ and then $\alpha_2$.
- When $i$ carries out the action $\alpha_1\cup\alpha_2$, it makes either $\alpha_1$ or $\alpha_2$ nondeterministically.
- When $i$ performs the action $\alpha^*$, it repeats $\alpha$ some finite number of times.

Remark that the dynamic constructs of our language come from the standard language of propositional dynamic logic [8,9]. As for test, when user $i$ does the action *exp*?, it evaluates according to its own knowledge, whether the current global state satisfies *exp*. It continues if *exp* is known to be true, otherwise it fails. We define expressions in the following inductive way:

$exp := \text{has}_i(m) \mid m=m' \mid \neg exp | exp_1 \vee exp_2 | \text{K}_i\ exp$

Atomic expression $\text{has}_i(m)$ is interpreted to mean that user $i$ can compute $m$ from:

- The pairing operators $\langle .,. \rangle$, left and right.
- The public keys $k_1, \ldots, k_n$.
- Its private key $k_i^{-1}$.
- The plaintexts, the symmetric keys and the nonces he has already computed.
- The messages it has already received.

We read $\text{K}_i$ *exp* as "user $i$ knows that *exp* is true". We should consider, for instance, the expression $\text{K}_i\text{has}_j(m)$ which represents the fact that user $i$ knows that user $j$ can compute $m$. Our language should allow us to express the notion of a user gaining knowledge over time as it receives messages from the network. To make this idea precise, we start with the primitive formulas $\text{K}_i exp$ and we form more complicated formulas by closing off under negation, disjunction, and the dynamic operators $[\alpha_1 \| \ldots \| \alpha_n]$. This is expressed in one line as:

$\varphi := \text{K}_i exp | \neg\varphi | \varphi_1\varphi_2 | [\alpha_1 \| \ldots \| \alpha_n]\varphi \mid \text{K}_i\varphi$

with the formula $[\alpha_1 \| \ldots \| \alpha_n]\varphi$ being read "after the parallel execution of actions $\alpha_1, \ldots, \alpha_n$ $\varphi$ is true" or "after every terminating execution of actions $\alpha_1, \ldots, \alpha_n$ in parallel, $\varphi$ is true".

Now consider the following protocol: $1 \rightarrow 2$: $k_1^{-1}(k_2(k_{12}))$, where user 1 sends $k_1^{-1}(k_2(k_{12}))$ to user 2. We assume that $k_{12}$ is a symmetric key picked by 1 in the key space *SYM*. As a result, 1 executes first the action key($x^k$), where $x^k$ is some key variable, and performs then the action $\text{send}(k_1^{-1}(k_2(x^k)))$. As for user 2, it does not know in advance the symmetric key it will receive. As a result, user 2 does the action $k_1^{-1}(k_2(y^k))$. Thus, in this protocol, users 1 and 2 are making respectively the actions $\beta_1$ and $\beta_2$:

$\beta_1 = \text{key}(x^k); \text{send}(k_1^{-1}(k_2(x^k)))$
$\beta_2 = \text{rec}(k_1^{-1}(k_2(y^k)))$

## IV. SEMANTICAL PRESENTATION

Now that we have described the syntax of our logic, we need a semantics to determine whether a given formula is true or false. Following the line of reasoning suggested by Fagin et al. [5], the first step consists of defining the notion of local state and the notion of global state. In our framework, a user's knowledge is determined by its local state whereas the global state describes the system at a given point of time. The user $i$'s local state consists of two substitutions: $\theta_i$ and $\tau_i$. Roughly speaking, $\theta_i$ is about the values given to variables by $i$ when it executes atomic actions like text($x$), key($x$) or nonce($x$), whereas $\tau_i$ is about the values given to variables by $i$ when it executes the atomic action rec($m$). Since a user's local state reflect the knowledge it has acquired, we assume that $i$'s local state is getting longer over time. Once we associate a local state to each user, we have to associate to the whole system a global state. A global state, at a given point of time, must contain the n-tuple of users local states. It must also contain the list of all messages that has been sent up to this time point as well as markers indicating that part of the list such and such user has already received.

Take the case of the global state:

$$\left(\begin{pmatrix} (x/m) & (w/m') \\ (y/m') & (z/m) \end{pmatrix} k_2(m,1) \quad k_1(m,m') \quad k_2(m') \quad \begin{matrix} \\ 1 \end{matrix} \quad \begin{matrix} \\ 2 \end{matrix}\right)$$

At this point of time, the above global state indicates that:

- User 1 has given the value $m$ to variable $x$ while executing some atomic action like text($x$), key($x$) or nonce($x$).
- User 1 has given the value $m'$ to variable $y$ while receiving some message.
- User 2 has given the value $m'$ to variable $w$ while executing some atomic action like text($w$), key($w$) or nonce($w$).
- User 2 has given the value $m$ to variable $z$ while receiving some message.
- Three messages have already been sent: $k_2(m,1)$, $k_1(m,m')$ and $k_2(m')$.
- User 1 has not read the third message yet, while user 2 has read all messages.

Let *Pt* is a countably infinite set of plaintexts, *Sk* be a countably infinite set of symmetric keys, *N* be a countably infinite set of nonces, and *Ct* be a countably infinite set of

ciphertexts. Let $\mathbf{M}=(Pt, Sk, N, Ct, \sigma, I_\sigma)$ denote a model based on *Pt*, *Sk*, *N* and *Ct*. Consider an expression *exp*, a formula φ and a global state *s*. The relation "*exp* is true at global state *s* with respect to a substitution σ", denoted $exp\in T_M(s,\sigma)$, and the relation "φ is true at global state *s* in model **M**", denoted $\mathbf{M},s\models\varphi$, are defined inductively on the formation of *exp* and on the formation of φ as follows:

- $has_i(m)\in T_M(s,\sigma)$ iff, according to the information it has at its local state $s_i$, user *i* can compute $I_\sigma(m)$.
- $m=m'\in T_M(s,\sigma)$ iff $I_\sigma(m)=I_\sigma(m')$.
- $\neg exp\in T_M(s,\sigma)$ iff $exp\notin T_M(s,\sigma)$.
- $exp\vee exp'\in T_M(s,\sigma)$ iff $exp\in T_M(s,\sigma)$ or $exp'\in T_M(s,\sigma)$.
- $K_i exp\in T_M(s,\sigma)$ iff, for every global states *t*, if $s\equiv_i t$ then $exp\in T_M(s,\sigma)$.
- $s\models K_i exp$ iff, for every global states *t*, if $s\equiv_i t$ then $exp\in T_M(t, \theta_i\cup\tau_i)$.
- $s\models\neg\varphi$ iff $s\not\models\varphi$.
- $s\models\varphi\vee\psi$ iff $s\models\varphi$ or $s\models\psi$.
- $s\models[\alpha_1\| \dots \|\alpha_n]\varphi$ iff, for all global states *t*, if $sR_{\alpha 1\| \dots \|\alpha n}t$ then $t\models\varphi$.
- $s\models K_i\varphi$ iff, for all available global states *t*, if $s\equiv_i t$ then $t\models\varphi$.

The definition of availability is given later in this section. The binary relations $\equiv_i$ are the relations between global states defined by $s\equiv_i s'$ iff:

- $\theta_i=\theta'_i$
- $\tau_i=\tau'_i$
- User i has sent in *l* and *l'* the same messages in the same order.
- User i has received in *l* and *l'* the same messages in the same order.

where

$$s=\left(\begin{pmatrix}\theta_1\dots\theta_n\\ \tau_1\dots\tau_n\end{pmatrix}l\right) \text{ and } s'=\left(\begin{pmatrix}\theta'_1\dots\theta'_n\\ \tau'_1\dots\tau'_n\end{pmatrix}l'\right)$$

Remark that $\equiv_i$ is an equivalence relation between global states for every user i. The binary relations $R_{\alpha 1\dots\alpha n}$ are the relations between global states defined by:

$R_{\alpha 1,\dots,\alpha i1\cup\alpha i2,\dots,\alpha n}=R_{\alpha 1,\dots,\alpha i1,\dots,\alpha n}\cup R_{\alpha 1,\dots,\alpha i2,\dots,\alpha n}$ and

$R_{\pi 1;\alpha 1,\dots,\pi n;\alpha n}=R_{\pi 1,\lambda,\dots,\lambda}\circ R_{\alpha 1,\dots,\pi n;\alpha n}\cup\dots\cup R_{\lambda,\dots,\lambda,\pi n}\circ R_{\pi 1;\alpha 1,\dots,\alpha n}$,

where $\pi_i$ are atomic actions, namely actions of the form text(.), key(.), nonce(.), send(.) or rec(.). $sR_{\lambda,\dots,exp?,\dots,\lambda}t$ iff $s=t$ and $t\models K_i exp$.

$$\left(\begin{pmatrix}\theta_1\dots\theta_n\\ \tau_1\dots\tau_n\end{pmatrix}l\right)R_{\lambda,\dots,\text{key}(x),\dots,\lambda}\left(\begin{pmatrix}\theta_1\dots\theta_i(x/c)\dots\theta_n\\ \tau_1\quad\dots\quad\dots\quad\tau_n\end{pmatrix}l\right)$$

where *c* is some symmetric key in *Sk*. For the actions nonce and text we have a similar condition.

$$\left(\begin{pmatrix}\theta_1\dots\theta_n\\ \tau_1\dots\tau_n\end{pmatrix}l\right)R_{\lambda,\dots,send(m),\dots,\lambda}\left(\begin{pmatrix}\theta_1\dots\theta_n\\ \tau_1\dots\tau_n\end{pmatrix}l,m(\theta_i\cup\tau_i)\right)$$

Here $m(\theta_i\cup\tau_i)$ is the result of applying substitutions for $\theta_I$ and $\tau_i$ to the term *m*. ($m(\theta_i\cup\tau_i)$ must be ground.)

$$\left(\begin{pmatrix}\theta_1\dots\theta_n\\ \tau_1\dots\tau_n\end{pmatrix}\dots\quad\begin{matrix}m'\dots\\ i\end{matrix}\right)R_{\lambda,\dots,rec(m),\dots,\lambda}$$

$$\left(\begin{pmatrix}\theta_1\quad\dots\quad\theta_i\quad\dots\quad\theta_n\\ \tau_1\dots\tau_i\mu(m,m')\dots\tau_n\end{pmatrix}\dots m'\quad\begin{matrix}\dots\\ i\end{matrix}\right)$$

where the substitution $\mu(m,m')$ matches the message *m* to the message *m'*.

Initial global state: $s_0=\left(\begin{pmatrix}\lambda\dots\lambda\\ \lambda\dots\lambda\end{pmatrix}_{1\dots n}\quad\lambda\right)$

A global state *s* is said to be *available* when there exists programs $\alpha_1, \dots, \alpha_n$ such that $s_0 R_{\alpha 1\| \dots \|\alpha n} s$.

We have seen the program of the protocol: $1\to 2$: $k_1^{-1}(k_2(k_{12}))$, where user 1 sends $k_1^{-1}(k_2(k_{12}))$ to user 2. Let us see its running! Here at the beginning there are no substitutions and there are no messages. The initial global state with two agents is equal to:

$$\left(\begin{pmatrix}\lambda&\lambda\\ \lambda&\lambda\end{pmatrix}_{12}\quad\lambda\right)$$

As user 1 executes the key(*x*) action and produces the key *k*, the global state became the following:

$$\left(\begin{pmatrix}(x/k)&\lambda\\ \lambda&\lambda\end{pmatrix}_{12}\quad\lambda\right)$$

In this step there are no messages yet. But when user1 send the message $k_1^{-1}(k_2(k))$, we move to the following global state:

$$\left(\begin{pmatrix}(x/k)&\lambda\\ \lambda&\lambda\end{pmatrix}_{12}\quad k_1^{-1}(k_2(k))\right)$$

When user 2 receives this message, its marker moves forward, and user 2 realizes that the variable *y* gets the value *k*:

$$\left(\begin{pmatrix}(x/k)&\lambda\\ \lambda&(y/k)\end{pmatrix}_1\quad k_1^{-1}(k_2(k))\quad_2\right)$$

We are interested in users' knowledge, for example at this global state user 2 can deduce that user 1 knows (or using our terminology has) the key *k*. Because private key $k_1^{-1}$ is known only by user 1, user 2 is sure that user 1 has the message $k_2(k)$, because user 1 is the only user able to produce $k_1^{-1}(k_2(k))$ from $k_2(k)$. In this case we have only two users, so user 2 knows that it cannot receive the

message $k_2(k)$ from a third user. Hence user 1 has generated $k_2(k)$ from $k$ itself. And in this case user 1 really has the key $k$.

What is the situation if we have more users, e.g. three, and user 2 executes its own program? Is it true that in each case user 1 has the key $k$? We can restate this question as: for all programs $\zeta_1$ and $\zeta_3$: $s_0 \models [\zeta_1 \| \beta_2 \| \zeta_3] K_2 has_1(y)$? It can be checked easily that with $\zeta_1$=rec($x$);send($k_1^{-1}(x)$) and $\zeta_3$=key($z$);rec($k_2(z)$) we have a case where user 1 does not own the key.

## V. COUNTEREXAMPLES

If we work with several different logical modalities, the following question arises: can we change the ordering of modalities? In other words, the resulting two formulae are equivalent or one of them is the consequence of the other formula? The following examples prove that the answer is *no*.

If $\alpha$=nonce($x$);send($k_1^{-1}(x)$), $\beta$=rec($k_1^{-1}(y)$), and $\varphi$= $K_2 has_1(x)$ then $s_0 \models \neq K_1[\alpha \| \beta]\varphi \supset [\alpha \| \beta] K_1 \varphi$. If $\alpha$= nonce($x$);send($k_1^{-1}(x)$);rec($y$), $\beta$=rec($k_1^{-1}(z)$);send($k_2^{-1}(z)$), and $\varphi = K_1(x=k_2(y))$ then $s_0 \models \neq [\alpha \| \beta] K_1 \varphi \supset K_1[\alpha \| \beta]\varphi$. In this case there is only one way to execute parallel programs $\alpha$ and $\beta$, hence $s_0 \models \neq [\alpha \| \beta] K_1 \varphi \supset K_1 \langle \alpha \| \beta \rangle \varphi$.

## VI. DECIDABLE PROBLEMS

Let **M**=(*Pt*, *Sk*, *N*, *Ct*, $\sigma$, $I_\sigma$) be a model based on *Pt*, *Sk*, *N* and *Ct*. Let $m$ be in $M$, where $M$ is the union of the sets $M_0$, $M_1$, … defined in section 2. We define the *computability of m for i* as follows:

- If $x/m \in \theta_i$ or ($x/m' \in \tau_i$ and $I_\sigma(m) = I_\sigma(m')$) then $m$ is computable.
- If $m'$ and $m'' \in M$ are computable then $\langle m', m'' \rangle$ is computable too.
- If $m' \in Sk$ and $m'' \in M$ are computable then $I_\sigma(E)(m', m'')$ is computable too.
- If $m' \in M$ is computable then $I_\sigma(k_j)(m')$ and $I_\sigma(k_i^{-1})(m')$ are computable too.
- If $m' \in M \setminus M_0$ is computable then $I_\sigma(\text{left})(m')$ and $I_\sigma(\text{right})(m')$ are computable too.

Let us define the function $d$ on messages as:

- If $m \in M_0$ then $d(m)=0$.
- $d(k_i(m))=d(k_i^{-1}(m))=d(E(k,m))=d(\text{left})(m)=$ $d(\text{right})(m)=d(m)+1$, where $k \in Sk$.
- $d((m,m'))=1+\max(d(m),d(m'))$.

If $\theta_i = (x_1/c_1)\ldots(x_l/c_l)$, $\tau_i=(y_1/m_1)\ldots(y_k/m_k)$ and we are interested that message $m$ is computable for $i$ where $e=\max(d(m_1), \ldots, d(m_k))$ and $f=d(m)$, then we need to check that there exist a message $m'$, for which $I_\sigma(m)= I_\sigma(m')$, $d(m') \le e+f$, and $m'$ does not contains variables but $x_j$ and $y_j$. There is only finite such message $m'$, so it is decidable, that a $m$ is computable, or not.

**Problem 1.** *Given a global state s. Decide whether s is available or not.*

We can assume, that all the create actions (text, nonce, key) are executed before the send and receive actions. And now we go backward. At first we need to find that which user did the last action. Here we have maximum $n$ possibilities. Let us assume that user $i$ did. Next we need to find that was the last action.

- If the last action was send($m$), examine that $m$ is computable for user $i$ or not. If it is, move the pointer $i$ left.
- If the last action was rec($m$), examine that $m$ is computable for user $i$ or not. If it is, delete the last element of $\tau_i$ and move the pointer $i$ left.

If user $i$ cannot compute, then choose a different case (back-track). If there are no messages, i.e. the pointers are the leftmost position, and all the $\tau_i$ are empty list, the original state is available.

**Problem 2.** *Given "star-free" programs $\alpha_1, \ldots, \alpha_n$ and a "dynamic-free" formula $\varphi$ of the form $K_{i1}\ldots K_{il} has_i(m)$, $K_{i1}\ldots K_{il} \neg has_i(m)$ or $K_{i1}\ldots K_{il}(m=m')$. Decide whether $[\alpha_1 \| \ldots \| \alpha_n]\varphi$ or not.*

Star-free programs do not contain the star (*) operator, and dynamic free formulae do not contain the [.‖…‖.] modality.

**Lemma.** *The previous problem is decidable.*

Proof. For any program $\beta$ there exist a program $\beta'$ such that for any global state s and any programs $\gamma_i$: $s \models [\beta \| \gamma_2 \| \ldots \| \gamma_n]\varphi$ if and only if $s \models [\beta' \| \gamma_2 \| \ldots \| \gamma_n]\varphi$, and $\beta'$ is a nondeterministic choice of sequences of atomic programs, send and receive instructions: $\beta' = (\pi_{11};\ldots;\pi_{1j1}) \cup \ldots \cup (\pi_{u1};\ldots;\pi_{ujt})$. According the properties of the accessibility relation between global states, no generality is lost by assuming that the programs $\alpha_i$ are sequences of atomic programs. If we assume that the sequences contains $j_1, \ldots, j_n$ atomic programs, then at most $\frac{(j_1+\ldots+j_n)!}{j_1! \times \ldots \times j_1!}$ different ordering of atomic programs is possible. From the ordering we can determine the final global state $s$. Now we need to answer, that for all such $s$, $s \models K_1\varphi$ holds, or not. This means, that for all global state $t$, such that $s \equiv_1 t$ we need to check, that $t \models \varphi$ or not. Unfortunately there are infinitely many such global states. But we do not need to check all of them.

- If $\varphi$ is $m=m'$, then we can decide easily whether $I_\sigma(m)= I_\sigma(m')$ or not.
- If $\varphi$ is $has_1(m)$ or if it is $\neg has_1(m)$, then $t \models has_1(m)$ iff $s \models has_1(m)$. ($s$ and $t$ are equivalent states according to user 1), so we need to check only one state.
- If $\varphi$ is $has_2(m)$, user 2 knowledge does not decrese by reading new messages, so if $s \models has_2(m)$ then $t \models has_2(m)$, where the pointer of user 2 in $t$ is to the right to the pointer of user 2 in $s$. The position of the pointers of the other users has no effect on knowledge of user 2. Hence we need to check the remaining cases, namely $s$ and the available states where the pointer of user 2 in $t$ is to the left to the pointer of user 2 in $s$. There are only finite such cases.

- If φ is $\neg has_2(m)$ then we would like to know, that there exists some global state $t$, for which $t \models has_2(m)$. With other words: could the other users combine their knowledge to acquire the message $m$. Their own knowledge could increase, but their distributed knowledge not. So we need to test, that from their distributed knowledge (keys, nonces, messages) plus the messages sent by user 1 the message $m$ is computable or not.
- If $\varphi = K_2\varphi$, then by definition $s \models K_1K_2\varphi$ iff for all $t$, such that $s \equiv_i t$, $t \models K_2\varphi$. From the previous points we know, that if φ is $has_i(m)$, $\neg has_i(m)$ or $m=m'$, then we need to check only finite cases for all $t$, to test that $t \models K_2\varphi$ holds or not. If the pointer of user 2 in $t$ is to the right to the pointer of user 2 in $s$, then $s \models K_2\varphi$ implies that $t \models K_2\varphi$ for this kind of formulae φ. Hence we need to check only finitely many state $t$. By similar reasoning the statement can be proved for longer sequences of epistemic modalities.

First and last we need to test only finitely many cases, so the problem is decidable. □

## VII. CONCLUSIONS

In this paper, we have presented our formal language devoted to the verification of authentication protocols. Through the example of protocol, we have seen how the exchanges of messages between users can be expressed. The axiomatization and the decidability of the set of formulae true at all global states are open problems.

## ACKNOWLEDGMENTS

The research of the first author is partly supported by the French ministry of education whereas the research of the second author is supported by the COST action "Theory and Applications of Relational Structures as Knowledge Instruments".

## REFERENCES

[1] Ágnes Kurucz, Combining modal logics, Handbook of Modal Logic, eds.: J. van Benthem, P.Blackburn, F. Wolter, Studies in Logic and Practical Reasoning, Volume 3. Elsevier, 869-924. 2007.

[2] Michael Burrows, Martín Abadi, and Roger Needham. A logic for authentication. In Proceedings of the Royal Society of London, volume 426, pages 233-271, 1989.

[3] John Clark and Jeremy Jacob. A survey of authentication protocol literature. unpublished

[4] Dorothy Elizabeth and Robling Denning. Cryptography and Data Security. Addison-Wesley Pub Co, 1982.

[5] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. Reasoning about knowledge. MIT Press, Cambridge, MA, 1995.

[6] Dov M. Gabbay, Ágnes Kurucz, Frank Wolter, and Michael Zakharyaschev. Many-Dimensional Modal Logics: Theory and Applications. Studies in Logic and the Foundations of Mathematics, Volume 148. Elsevier, 2003.

[7] Dov M. Gabbay and Valentin B. Shehtman. Products of modal logics. {I}. Logic Journal of the IGPL. Interest Group in Pure and Applied Logics, 6(1):73-146, 1998.

[8] Robert Goldblatt. Logics of Time and Computation, volume 7 of Lecture Notes. Center for the Study of Language and Information, 1987.

[9] David Harel, Dexter Kozen, and Jerzy Tiuryn. Dynamic Logic. MIT Press, 2000.

[10] Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. Information Processing Letters, 56:131-136, 1995.

[11] Maarten Marx and Yde Venema. Multi-dimensional modal logic, volume 4 of Applied Logic Series. Kluwer Academic Publishers, Dordrecht, 1997.

[12] Aviel D. Rubin and Peter Honeyman. Formal methods for the analysis of authentication protocols. unpublished