THE PUBLISHING HOUSE
OF THE ROMANIAN ACADEMY

PROCEEDINGS OF THE ROMANIAN ACADEMY, Series A,
Volume 13, Number 2/2012, pp. 141–148

LOGIC ENGINEERING WITH APPLICATIONS TO SECURITY*

Cristian-Dumitru MASALAGIU, Vasile ALAIB

"Al. I. Cuza" University of Iaşi, Romania
E-mail: mcristy@info.uaic.ro

Computer (or Information) Security is a branch of Information Technology used for distributed and concurrent systems (multiagent systems, networks, etc.). The main purpose is to prevent unauthorized access to the information passed through the system. In the same time, the information has to remain accessible and reliable for the trusted customers. Abstract cryptographic (security, encrypted, etc.) protocols are generally used. These have to be proved *a priori* to be correct and this is done using formal logic techniques. The so called logics of belief are usually preferred. Unfortunately, the problem we are facing is far more complicated and we have to learn how to engineer such logics.

Key words: security, logics of belief, cryptographic protocols, logic engineering.

1. INTRODUCTION TO LOGIC ENGINEERING CONCEPTS

Logic engineering is an emerging discipline concerning the application of logic to specific fields of research. The problems logic engineering addresses are to **build formal modeling languages** for specific problems or classes of systems, **develop and simulate models** using logic-based formal languages and, most importantly, **property specification and verification** by expressing properties of systems as logic formulas and check their validity under a specific interpretation.

The subject of using logic based formalisms and tools to solve domain specific problems benefited from attention from various authors over time. Although there is no unitary curriculum yet or a solid common vision around the concept of logic engineering, the few definitions of the term agree to the basic notions of using logics, formalism, in new contexts. The term engineering suggests that this can be a repeatable and predictable process, up to a point.

The views on the term *logic engineering* found in literature can be summarized as the choice, adaptation and usage of formalisms and related techniques, similar to software engineering, in order to solve specific problems [10, 11, 12, 1, 13].

Our goal in this paper is to outline the first steps towards the application of logic engineering methods to the security protocol verification problem. We start by introducing general notions of logic and security and continue with the description of **BAN** logic. We then create for it a formal syntax description using the *Backus-Naur* notation and make the first steps towards semantics. We finalize the paper with conclusions regarding the application of the syntax to formal verification and steps for future research.

2. GENERAL USEFUL NOTIONS

Some modal logics have particular applications, such as the representation and treatment of knowledge in multiagent systems. The main modalities (unary modal operators) are □Q (**box**) and ◊Q (**diamond**) but the formal semantics is different (from e.g. temporal logics)

Briefly:

□: *agent_Q_knows_certainly_that*…

◊: *from_what_agent_Q_knows*…

---

* This paper was presented in part at the 1st Conference *Romanian Cryptology Days* – RCD2011, October 11–12 2011, Bucharest, Romania.

Epistemic modal logics are modal logics concerned mainly with *reasoning about knowledge*. Without entering into details, we say only that we may go back to Greece, Aristotle, and – generally – to *epistemology*, as a branch of philosophy. All of these have to be treated in a formal, and - in the same time – practical effective way.

Related concepts to the main topic are modal and epistemic logics (**MDE**) [8], model checking (**MC**) [1], natural deduction (**ND**) [1], logic programming (**PROLOG**) [14], lambda notation and calculus (**LAMBDA**) [7] and decidability and complexity (**DECO**) [3].

Definition. Let **SD** = $<\mathcal{A}, \mathcal{R}>$ be a deductive system in a class of formulae denoted **FORM**. A *proof* (for $\mathbf{F_m}$ starting with $\mathcal{A}$) in **SD** is a list of formulae ($\mathcal{D}$): $F_1, F_2, \ldots, F_m$ such as for each $i \in [m]$, either $F_i \in \mathcal{A}$, or F_i is obtained from $F_{j1}, F_{j2}, \ldots, F_{jn}$ using a rule: $r = << \{F_{j1}, F_{j2}, \ldots, F_{jn}\}, F_i>, c> \in \mathcal{R}$, where $i_1, i_2, \ldots, i_n < i$.

Consequently, each element of the list ($\mathcal{D}$) is either an **axiom** (element of $\mathcal{A}$), or it is the consequent of an **inference rule** (element of $\mathcal{R}$) with the previous elements in the list as hypotheses. **Soundness** and **completeness** theorems may then be introduced.

3. SECURITY

Every security protocol incorporates at least one of the following aspects: the establishment and relationing of keys, entity identification, symmetric encryption and message authentication, insurance of secrecy at the lowest possible level and using combined methods. For example, **TLS** (**T**ransport **L**ayer **S**ecurity) is a cryptographic protocol used to secure **HTTP** connections (web). We will provide only some elementary notions needed for a better understanding of the logical part and we will point out the true link between *computer security* and *logic*.

The **security protocols** will be treated only from an abstract point of view (i.e. there is no deduction on lists of bites, only on text messages). It is supposed that all the participants are able to recognize different types of messages although in the 0-1 presentation part of the initial format may be missing. A **cryptographic/security/encryption protocol** is a protocol additionally having embedded a "function" assuring security by applying some cryptographic methods. Thus we have the environment, cryptography, communication protocols. It is not enough, as we need tools for *a priori* verification of protocol *correctness*. These are based upon (various types of) **logic**. Logics can be integrated in formal meta-models.

Prove the following statement in a fixed cryptographic protocol (and using **BAN**, e.g.) for a fixed security system (part of proving that the protocol *works properly*, i.e. it is *correct/sound,* i.e. it *reach its specifications*, etc.). For example, **because** this message was signed by the *machine/agent/principal* B, **then** the *machine/agent/principal* A can be certain that the considered *message* comes indeed from B.

Consider the **Dolev/Yao model** [16] for *designing* specific protocols. Short *description*: the design is based on the *enumeration* of all actions that can take place in a given computer security system. Additional *supposition*: the encryption is *perfect*, but any *intruder* may have a complete control of the net in the sense that he/she/it can view, delete or falsify any message transmitted/received in the net. In fact all this is related to the complexity of the semantic domains.

Advantages. There exist many analysis techniques which may be developed inside this model (some of them – [semi]automatized), allowing the creation of [semi]formal proofs for some general properties of the given protocol (such as the *certitude of secretization*).

Disadvantages. Unnatural from a human point of view. To design a protocol we rather think to use some high-level concepts (e.g. "the *secret key* is known only by A and B and it is the only one used for the communication between the two agents") such that we can infer statements like "if a message arrives encrypted with a key known only to me and to the machine M, and it is not me which send it in the first place, then the considered message has to have been sent by M".

The *Needham-Schroeder* (with *shared keys*) **protocol** ((**NS**) for short) [17], is defined as sequences of *messages* (*plain texts, cryptotexts*), as follows (see the semantic part for details):

O1. $P \rightarrow S : P, Q, N_p$

O2. $S \rightarrow P : \{N_p, Q, \mathbf{K}_{pq}, \{\mathbf{K}_{pq}, P\}_{\mathbf{K}_{qs}}\}_{\mathbf{K}_{ps}}$

O3. $P \rightarrow Q : \{\mathbf{K}_{pq}, P\}_{\mathbf{K}_{qs}}$

O4. $Q \rightarrow P : \{N_q\}_{\mathbf{K}_{pq}}$

O5. $P \rightarrow Q : \{N_q - 1\}_{\mathbf{K}_{pq}}$

S, P, Q are *machines/agents/principals* (usually, S is considered to be a *server*). N_p and N_q are *invented words* (*nonces*); they represent in fact *random numbers*, chosen by P respectively Q, and used to prevent *replay attacks*; these types of attacks mean that an *intruder* will replicate (parts of) a message sent/kept in one of the previous sessions.

The general idea behind the nonces is that the principals verify the fact that the values used in certain encrypted messages correspond to the correct values of the nonces for the current session; the discrepancies, which may occur, e.g., because of the presence/accessibility of the messages memorized during some previous sessions, can be detected by the principals (they can, in this way, avoid such messages if the protocol is correctly managed). **K**xy ($\mathbf{K}_{xy}$) is a generic notation for a key **K** *shared* between x (e.g. p, P) and y (e.g. q, Q).

The **main goal** of the (**NS**) protocol is to allow P and Q to agree to share (in communication) the key $\mathbf{K}_{pq}$. Consequently, in a first step, P and Q will use a *trusted* server S, which will generate the key during the execution of the protocol. In a second step, S will communicate with P and Q using the shared keys $\mathbf{K}_{ps}$ respectively $\mathbf{K}_{qs}$, presumed to be known from the very beginning by the implied parts.

The *intuitive* semantics of **O1** and **O2**: P sends a message to the server identifying itself and Q (using the nonce N_p), telling the server that it wants to communicate with Q; then, the server generates $\mathbf{K}_{pq}$ and sends back to P a copy, encrypted under $\mathbf{K}_{qs}$, for P to forward to Q and also a copy for P. Since P may be requesting keys for several different agents, the nonce assures P that the message is *fresh* and that the server is replaying to that particular message and the inclusion of Q name tells P who it is to share the key with.

4. LOGICS OF BELIEF

Logics of belief were introduced in order to have a corresponding **formal** framework (clearly, a *great advantage* if we think at *implementations*). **BAN** logic was the first to be developed and it was followed by more expressive and evolved *extensions*. An unpleasant *limitation* of these logics is the necessity to (semantically) *annotate* the protocols with *extra (meta)* logical formulae (again, semantic domains have to be found). These are supposed to represent with high fidelity the intentions of the agent which sends a message and are used to express the *secretization* or the *freshness* of (some parts) of a message. Another important *disadvantage* is that the secretization cannot yet be formally proved (much more, the fact that the secret keys are protected is *implicitly* assumed). Hence, the term "**computer system security**" implies the idea that there exist formal methodologies and procedures which help that valuable services and information can be *a priori* protected against intruders or unpredictable events. These methods generally use the following paradigm: **we do not allow the behaviors which are not desirable, instead of allowing the acceptable behaviors**.

The **Dolev-Yao** model is used as a medium to formally *a priori* prove some specific properties of the interactive protocols. Using this abstract model, the net may be viewed as a set of abstract machines which have the role to "discuss with partners" (by changing messages). In this environment, the **intruder** may hear, intercept and memorize any message, but it is limited only by the used cryptographic methods.

Note. Cryptographic primitives are modeled by (abstract) operators. Thus, if we refer to the (asymmetric) encryption for a user **X**, it will be represented by the couple $<\mathbf{E}_\mathbf{X}, \mathbf{D}_\mathbf{X}>$ (represented by the encryption function, respectively decryption); this way we satisfy the property that their composition is identity ($\mathbf{E}_\mathbf{X}\mathbf{D}_\mathbf{X} = \mathbf{D}_\mathbf{X}\mathbf{E}_\mathbf{X} = \mathbf{1}$). Given a message **M**, its encrypted variant ($\mathbf{E}_\mathbf{X}(\mathbf{M})$) can not reveal anything about **M**. Although not in accordance with the real world, it is supposed that the adversary can not manipulate the encrypted message (not even at the 0-1 representation) and can not "guess" keys: $\mathbf{E}_\mathbf{x}$ may be written as $\mathbf{D}_\mathbf{x}^{-1}$.

Logics of belief are designed to show the conclusions a participant can draw upon in a communication dialog based upon received messages and the initial **beliefs**. We expect that an analysis based on logics of belief to guarantee that only the desired properties (regarding, for example, the **security of data**, **non-replicating transitions**, **admitting "trusted" persons**, etc.) are accepted during a **communication session**. Obviously, proofs related to abstract protocols are not proofs that the concrete protocols ("accurate" implementations of the abstract ones) are correct. There are numerous assumptions taken "for granted" for a certain implementation which, if not valid in reality, will lead to the conclusion that a protocol implementation is incorrect although the protocol is (classic example: *the crypto-algorithm is safe*).

When we design or explain a cryptographic (abstract) protocol, the formulation "**because** this message was *signed by the machine B,* **then** *machine A can be certain* that the message originates from *B*" is often used within the informal proof justifying that the protocol works *correctly*. From such a "proof" the idea of a *trust relationship* is missing. Much more, the fact that the certainty of such a message is not a copy of another from a previous session, is not formaly derived. Although special *formal logics of belief*, which we call upon, do not substitute automated proof or verification techniques (for example, model checking), they can be of good use, at least in uncovering of sensible points of the protocol that can weaken it. We must outline from the start that the **lack** of a *formal semantics* represents a handicap in using such logics in practice. **BAN** logic, as well as its successors are in fact *types of* (belief) **multisorted** (agents; encryption keys; nonces; formulae, etc.) **modal epistemic logics** [7]. Suitable formal semantics, as far as we know, they are not yet quite profitable from an implementation point of view (if they exist anyway).

4.1. Classical Approach for BAN Logic

The **BAN** logic refers to some ***deductive systems*** for defining/analyzing/authentifying cryptographic protocols. This logic (and its further extensions) helps the users to determine if the messaged information is to be *trust* and/or *certain*, versus *eavesdropping*. Anyway, the idea is that any information passed from one "person" to another, uses a vulnerable medium, where falsification and public monitorizing are normal. Suitable particular notations will be used, starting with the protocols themselves.

A simple derivation sequence in **BAN** logic usually includes three *supplementary* steps: the verification of the message origin, the verification of the freshness of the message and the verification of trust of the transmitter.

BAN logic is **decidable** and the classic algorithm uses a variant of the *magic sets* [3]. More specifically, there exists an algorithm which, having as input some *assumption* formulae and a *goal* formula (written in a **BAN** specific language), always stops with the answer **YES** (the goal may be inferred from the assumptions) or **NO. BAN** logic has practically *a pure formal syntactic definition*. The work on developing a formal *convenient* semantics is not yet finalized. **BAN** logic, as well as the succeeding ones (mentioned before), are ***modal logics*** (of trust, epistemic, etc. – from a semantic point of view). They deal primarily with agents **trusting** their ability to control their environment; for example, they could control the distribution of common or shared keys. Other necessary rules will be explicitly presented as part of the respective logic definition. As we have already pointed out, a complete formal semantics was not yet discovered and maybe it won't even be (or it is not worth the computational necessary effort). One cannot prove secrecy within this framework, but *it is presumed implicitly that secrets are protected*.

Let us go into details. The **axioms** and the **inference rules** of **BAN** logic follow (P, Q are *agents*, X is a *message* - or part of, and **K** is an *encryption key:* public, secret, and shared).

In what immediately follows we present the classical approach. Clearly, it is semi-formal.

AXIOMS

1A. P *believes* X (P|≡ X): P acts as it knows that X has to be trusted (X is *true*) and can guarantee that (including the situation when X is part of another message).

2A. P *has jurisdiction over* X (P |⇒ X): the opinion of P about (the truth of X) should be trusted. For example, the keys distribution made by the servers has to be trusted if we speak about keys.

3A. P *said* X (P |~X): At one time, P transmitted (and *believed*) X, although (in the present) it is possible that P does not yet *believe* X (that is, we don't know if the message was sent a long time ago or during the current communication session).

4A. P *sees* X (P < X): P receives the message X, it can read it and it can retransmit it

5A. {X}**K** (or $\{X\}_{\mathbf{K}}$): The message X is encrypted with the key **K**.

Note. A message $\mathbf{K}_{pq}$ which in fact transports a key created to be used in the communication between P and Q, will be represented in **BAN** by $P \leftrightarrow_{\mathbf{K}_{pq}} Q$; because any key is, at its turn, encrypted (in a way which cannot be discovered by an intruder, the transmitted message will be in fact encrypted; for example, $\{\mathbf{K}_{pq}\}_{\mathbf{K}_{qs}}$; the corresponding formula will be then $\{P \leftrightarrow_{\mathbf{K}_{pq}} Q\}_{\mathbf{K}_{qs}}$.

6A. *fresh*(X) (#X): X has not ever been transmitted in the current session; this fact is valid especially for the nonces; #X will be used as a complement for P |~X to establish that a message coming from P truly refers to the current session (i.e. it is not an older, memorized, message used already by an intruder).

7A. *key*(**K**, P↔Q) (or $P \leftrightarrow_{\mathbf{K}} Q$; this operator is *commutative*): P and Q may communicate using the *shared key* **K**; the key **K** is supposed to be *good*, in the sense that it will never be discovered by any other agent (except for P, Q or any other *trusted* by P or Q); this is the first time we have applied the supposition that *secrecy is implicitly protected*.

8A. P *has/possesses* a *public key* **K** ($\vdash\rightarrow_{+\mathbf{K}}$ P): the corresponding (*matching*) *secret key*, i.e. the *converse* of **K** will be denoted by „-**K**"; we suppose, again implicitly, that this key will never be discovered by any agent, except for P or any other in which P trusts.

Notation. <X, Y> will denote the *concatenation* of the messages X and Y.

Inference rules

1I (also denoted (**MM1**)). **If** P *believes key*(**K**, P↔Q) **and** P *sees* $\{X\}_{\mathbf{K}}$, **then** P *believes*(Q *said* X).

The idea behind: if a key **K** is shared by P and Q and **K** is maintained secret, if P *sees* a message encrypted with **K**, then P may be sure that it comes from Q; an additional implicit supposition is that the message is not necessarily sent by P; Burrows, Abadi and Needham explain this by saying: "$\{X\}_{\mathbf{K}}$" is in fact an abbreviation for "$\{X\}_{\mathbf{K}}$ *from* P", that is the encryption was made by P; much more, it is supposed that any agent may recognize the encrypted messages it has sent from the very beginning.

1I′. **If** P *believes key* (K, P↔Q) **and** P *sees* $\{X\}_{\mathbf{K}}$ **where** the message was not necessarily sent by P, **then** P *believes* (Q *said* X).

2I. **If** P *believes* (Q *said* X) **and** P *believes fresh*(X), **then** P *believes* (Q *believes* X)

This rule denoted also by (**NV**) (the *nonce verification* or the *freshness rule*), expresses the fact that we have verified that a message is recent (i.e. it was transmitted during the current session and the sender still believes in it); this prevents the *replay attacks*.

Note. For the above last two inference rules, the following local condition for applicability has to be considered.

1C. P *believes fresh*(X); so, if X is not recognized as being fresh, it is possible that it is an older message (maybe replaced by a potential hacker).

3I. **If** P *believes* (Q *has jurisdiction over* X) **and** P *believes* (Q *believes* X), **then** P *believes* X; this rule, denoted also by (**J**), "says" that: if P *believes* that Q *has jurisdiction over* X, then P *believes* Q, on the assumption that X is "trusted".

4I. **If** P *believes* (Q *said* <X, Y>), *said* P *believes* (Q *said* X) (**and** P *believes* (Q *said* Y)); this rule will also be denoted by (**SG**); the idea behind is that if an agent said something about a group of messages, he/she/it already told something about any individual.

Thinking at "easy to use" (but not at the efficiency) we may also introduce the following rules (without damaging the present proof system):

5I. Not surprisingly, an agent will *believe* a set of *messages* (*statements*) if and only if he (she) will *believe* any element of the given set (we remind that the catenation is associative and commutative); so, we may stipulate that:

5Ia. (**BE1**) **If** P *believes* X **and** P *believes* Y, **then** P *believes* <X, Y>.

5Ib. (**BE2**) **If** P *believes* <X, Y>, **then** P *believes* X (**and** P *believes* Y).

5Ic. (**BE3**) **If** P *believes* that (Q *believes* <X, Y>), **then** P *believes* that (Q *believes* X) (**and** P *believes* that (Q *believes* Y)).

Similarly:

5Id. (**BE4**) **If** P *believes* that (Q *believes* X) **and** P *believes* that (Q *believes* Y), **then** P *believes* that (Q *believes* <X, Y>).

6I. **If** an agent *sees* a *formula*, **then** he may *see* also its components (supposing he/she/it knows the necessary keys):

6Ia. (**SP1**) **If** P *sees* <X, Y>, **then** P s*ees* X (**and** P s*ees* Y).

6Ib. (**SP2**) **If** P *sees* $\{X\}_{\mathbf{K}}$ **and** P *believes* that *key*(**K**, P↔Q), **then** P s*ees* X.

Note. Let's point out that, indeed, the hypothesis in the last rule is "P *believes* that *key*(**K**, P↔Q)" and not "P *sees* K" (even this is not quite dangerous or strange).

In fact the last rule may be replaced (without damages) by the following pair of rules:

6Ib′. (**SP2'**) **If** P *sees* $\{X\}_{\mathbf{K}}$ **and** P *sees* **K**, **then** P *sees* X.

6Ib″. (**SP2''**) **If** P *believes* that *key*(K, P↔Q), **then** P *sees* **K**.

7I. **If** part of a formula (or subformula) is known to be *fresh*, **then** the entire formula has to be *fresh*:

7Ia. (**FR1**) **If** P *believes* that X is *fresh*, **then** P will *believe* also that <X, Y> is *fresh*, no matter who is Y.

7Ib. (**FR2**) **If** P *believes* that X is *fresh*, **then** P will *believe* that $\{X\}_{\mathbf{K}}$ is *fresh*.

Note. There exists a postulate unanimously accepted by all the people working in *public-key cryptosystems*, namely the fact the encrypted messages by using public keys may be decrypted only by using private keys.

8I. (**SP3**) **If** P *sees* {X}+**K** **and** P *believes* ($\mapsto_{+\mathbf{K}}$ P), **then** P *sees* X.

Note. In the preceding rule we have supposed that if **K** represents the public key for an agent, then he/she/it will indeed possesses the corresponding private key as well.

The next (and last for the moment!) rule is optional and expresses the fact that for some public-key cryptosystems (e.g. **RSA** [18]), it is possible that anyone who has access to the public key may decrypt every message encrypted with a private key:

9I. (**SP4**) **If** P *sees* $\{X\}_{\mathbf{K}}$ **and** P *believes* ($\mapsto_{-\mathbf{K}}$ Q), **then** P *sees* X.

Our main contribution is contained in the next sections.

5. FOMAL SYNTAX FOR THE *BAN* LOGIC

We have to point out (again) that the **BAN** logic has been created with the purpose to fit some particular demands. Anyone who tries to apply ***logic engineering techniques*** for proving the ***formal correctness of security abstract protocols*** has to use additional constructs and application restrictions to the inference rules and has to gain first some *specific skills*.

By using *automated deduction decision procedures* [8] we may discover e.g. missing or overloaded axioms and inference rules. We may also **prove**, by using *consistent deduction systems* (although the corresponding *logical theory* is usually missing if we think at a *complete formal way* and at a corresponding **completeness theorem**), that the agents may be confident that they are able to safely communicate by using certain protocols. If the **proof fails**, we have to conclude that there was an attack from an intruder which has compromised the communication protocol.

Anyway, the construction of the alphabet, formulae, correct and consistent deduction systems and logical theories, is a matter of knowledge, skills, proper methodologies, automatic tools, etc. In order:

- Alphabet
- Formulae
- Axioms and Inference Rules (Proof system)
- Semantics

These steps are necessary in order to define a formal syntax, according to the generic construction mode of a deduction system and to be able to start defining a formal semantics.

We have to define first the *alphabet* and the *formulae* for the just described **BAN** logic. Let's start by pointing out some already known notations:

- P, Q, …(or P_1, P_2, …) – ***agents/principals***
- S (or S_1, S_2, …) – ***servers/machines***
- X, Y (or X_1, X_2, …, Y_1, Y_2, …) – ***messages***
- N_p, N_q, …(or N_P, N_Q, …) – ***nonces*** (N_p or N_P is attached to the agent P, or p)
- **K**xy (or $\mathbf{K}_{xy}$) – a ***shared key*** between the agents x and y
- **K** – a ***public or*** a ***secret key***

binary operators:

1. believes: $|\equiv$
2. has jurisdiction over: $|\Rightarrow$
3. said: $|\sim$
4. sees: $<$
5. communicate (by a public key $+\mathbf{K}$ or a corresponding secret key $\mathbf{-K}$): $\leftrightarrow_{\mathbf{K}}$ ($\leftrightarrow_{+\mathbf{K}}$; $\leftrightarrow_{\mathbf{-K}}$)
6. from (taking into account **K**): $\leftarrow_{\mathbf{K}}$
7. concatenation: $<\bullet,\bullet>$

unary operators:

1. *encryption* (taking into account **K**): $\{\bullet\}_{\mathbf{K}}$
2. *fresh*: #(•)
3. *has/possesses* (taking into account **K**): $\mapsto_{\mathbf{K}}$ (again, we may speak about $+\mathbf{K}$ and $-\mathbf{K}$)

We are able now to give a precise *Backus-Naur description* of the class of ***formulae***:

<*server*> ::= S_1 | S_2 | …

<*agent*> ::= <*server*> | P_1 | P_2 | …

<*machine*> ::= <*agent*>

<*principal*> ::= <*agent*>

<*message*> ::= X_1 | X_2 | …

<*binary-operator*> ::= $|\equiv$ | $|\Rightarrow$ | $|\sim$ | $<$ | $\leftrightarrow_{\mathbf{K}}$ | | $\leftrightarrow_{+\mathbf{K}}$ | $\leftrightarrow_{\mathbf{-K}}$ | $\leftarrow_{\mathbf{K}}$ | $\leftarrow_{+\mathbf{K}}$ | $\leftarrow_{\mathbf{-K}}$ | $<\bullet,\bullet>$

<*unary-operator*> ::= $\{\bullet\}_{\mathbf{K}}$ | #(•) | $\mapsto_{\mathbf{K}}$ | $\mapsto_{+\mathbf{K}}$ | $\mapsto_{\mathbf{-K}}$

<*nonce*> ::= $N_{<agent>}$

<*public-key*> ::= $\mathbf{K_1}$ | $\mathbf{K_2}$ | …

<*secret-key*> ::= $\mathbf{K'_1}$ | $\mathbf{K'_2}$ | … (or -**K**)

<*shared-key*> ::= $\mathbf{K}_{<agent><agent>}$

<*key*> ::= <*public-key*> | <*shared-key*> | <*secret-key*>

<*formula*> ::= <*agent*> <*binary-operator*> <*message*> |

| < <*message*>, <*message*>> |

| <*agent*> <*binary-operator*> <*agent*> |

| <*unary-operator*> <*message*> |

| <*unary-operator*> <*agent*>

Note. Not every formula must have a significant formal meaning. That is, all the *names* above are *variables* (which will be *interpreted* in the corresponding *semantics*).

5.1. A Few Words on Formal Semantics for BAN and Logics of Belief

Semantics is indeed a complex problem. As we have already pointed out (several times up to now), a *complete and reliable* **formal** *operational* (or other types...) *semantics* has **not yet** been **invented** (maybe it will never be, or it will be not valuable from a computational point of view).

We will anyway suggest some ideas for the definition of the concrete domains for: *servers*, *agents*, *principals*, *machines*, *messages*, *operators*, *nonces*, *keys* (public, secret, shared), *and formulae* in order to understand the complexity of the work which has to be (and we intend to) accomplished in the future. The *association* between a variable and the corresponding domain element is quite straightforward. The domains, however, are much harder to determine.

6. CONCLUSIONS

BAN logic refers to some deductive systems for defining, analyzing and authentifying the abstract cryptographic computer security protocols in interactive media **before** their implementation. This type of modal epistemic logic helps the users to determine if the passed information has to be trusted and safe versus *eavesdropping*.

This particular logic and its successors start with the idea that all the implied information circulates in vulnerable media which faces public monitoring and falsification. So we have not to trust **a priori** the **safety** of INTERNET.

REFERENCES

1. M. Huth, M. Ryan, *Logic in Computer Science: Modelling and Reasoning about Systems*, Cambridge University Press, England, 2000.
2. R. Stalnaker, *On Logic of Knowledge and Belief*, Springer Verlag, 2006.
3. P.C. van Oorschot, *Handbook of Applied Cryptography*, Carleton University, 2002.
4. T. Kwon, S. Lim, *Automation-Considered Logic of Authentication and Key Distibution*, Spinger Verlag, 2003.
5. D. Yiqiang, *An Improvement of GNY Logic for the Reflection Attacks*, Springer Verlag, 1999.
6. D. Monniaux, *Analysis of Cryprographic Protocols Using Logics of Belief: An Overview*, J. T. I.T., 2002.
7. R. Fagin et al., *Reasoning about Knoledge*, M. I. T. Press, 2003.
8. J.J. Ch. Meyer, W. van der Hoek, *Epistemic Logic for AI and Computer Science*, Cambridge Univ. Press, 2004.
9. M. Benerecetti et al., *A Logic of Belief and a Model Checking Algoritm for Security Protocols*, 2000.
10. P. Lucas, *Logic Engineering in Medicine*, The Knowledge Engineering Review, **10**, pp. 153–179, 1995.
11. S. Berghofer, M. Wenzel, *Inductive datatypes in HOL – Lessons Learned in Formal Logic Engineering*, Proc. of TPHOLs'99, 19–36, 1999.
12. C. Areces, *Logic Engineering. The Case of Description and Hybrid Logics*, PhD Thesis, Institute for Logic, Language and Computation, University of Amsterdam, 2000.
13. S. Veloso, P. Veloso, R. de Freitas, *An Application of Logic Engineering*, Logic Journal of the IGPL, **13**, *1*, pp. 29–46, 2005.
14. J.J. Alferes, L.M. Pereira, *Reasoning with Logic Programming*, Springer-Verlag, 1996.
15. A. Pnueli, Y. Sa'ar, L.D. Zuck, *JTLV: A Framework for Developing Verification Algorithms*, Computer Aided Verification, LNCS, **6174**, pp. 171–174, 2010.
16. D. Dolev, A.C. Yaho, *On the security of public key protocols*, IEEE Transactions on Information Theory, **IT-29**, *12*, pp. 198–208, 1983.
17. R.N. Needham, M.D. Schroeder, *Using encryption for authentication in large networks of computers*, Communications of the ACM, **21**, *12*, 1978.
18. R. Rivest, A. Shamir, L. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM, **21**, *2*, pp. 120–126, 1978.
19. M. Wenzel, *The Isabelle/Isar Reference Manual*; url: http://www.cl.cam.ac.uk/research/hvg/isabelle/dist/Isabelle2011/doc/isar-ref.pdf, 2011.

Received February 29, 2012