Integrating Proof-Based and Model-Checking Techniques for the Formal Verification of Cryptographic Protocols

Dominique Bolignano

Dyade, B.P.105 78153 Le Chesnay Cedex France, Dominique.Bolignano@dyade.fr

Abstract. We discuss the advantages and limitations of the main proof-based approaches to the formal verification of cryptographic protocols. We show possible routes for addressing their limitations by combining them with model-checking techniques. More precisely we argue that proof-based techniques can be used for providing a general framework, model-checking techniques for mechanization and invariant techniques for bringing precise understanding of protocol strengths and weaknesses.

1 Introduction

Three different basic research directions have been adopted for the formal verification of cryptographic protocols: one is based on the use a specific modal logic (e.g. a logic of authentication); another is based on the use of general purpose formal methods; the third is uses of model-checking techniques [13, 11]. The two first approaches use proof-based techniques for the verification phase.

We believe that neither of these approaches provides a complete solution to the verification problem, and we try here to discuss the benefits of integrating the various techniques, taking general formal methods based techniques as a framework. As an illustration we use the integration achieved for this purpose in [6]. Of course the adequacy of the integration clearly depends on the objectives that are assigned to the use of formal methods. In the following discussion we basically assume, based on our experience in applying formal methods in the design of large secure systems, that no potentially unsafe approximation or simplification should be allowed, neither in the modelization of the protocol, nor in the expression of properties, or in the verification itself. We further believe that verification should be a vehicle to bringing precise understanding of the protocol strengths and weaknesses. We argue that these objectives are not necessarily conflicting with automatization.

Typical modal logic approaches such as the BAN logic [7] provide a very elegant way of proving authentication properties, but the modeling is in some way wired and corresponds to a high level of abstraction. They are thus the most efficient when the objective of the verification is to identify major flaws (as opposed to proving the absence of flaws). Some other modal logics have been proposed to achieve more precision, but this is often done at the expense of a loss in conciseness or simplicity (see [3] for a more detailed discussion).

Model checking techniques on the other hand perform a verification on a finite model. The verification is thus automatic. Typically this model is not the specification itself but only corresponds to an abstraction of the specification. As a result, nothing can be formally inferred from the verification when it is successful. To this respect they are very similar to modal logic approaches. A discussion on this issue can be found in [6]. As compared to BAN-like modal logics they provide more automatization capabilities. But since the verification is a black-box process it also provides less insight into the precise understanding of the strength and weaknesses of a protocol.

The use of general purpose formal methods has the advantage of relying on largely used techniques. The main approaches are the approach of Kemmerer [12] based on the formal specification language Ina Jo, the approach of Chen and Glicor [9] also based on the Ina Jo specification language, but using the BAN logic [7] to model belief, Bieber's approach [2] based on the formal specification language B and finally the approach of Meadows [14], based on communicating processes and which builds upon the approach of Millens [15] and the approach of Dolev and al. [10]. More recently two other approaches have been presented. The first approach defined in [3] uses general purpose formal methods as a framework but it also relies on abstract interpretation and model-checking techniques for achieving automatization, while the second one [16] is a pure proof-based technique which, as we will see, imposes a few constraints.

The main challenge with general purpose formal methods is to achieve precision and conciseness at the same time. It is also of prime importance to achieve significant automatization to keep the approach efficient and workable. An added but very important potential benefit of formal methods is to provide a very precise understanding of the protocol design issues (weaknesses, strengths, hypotheses, etc.). We will use three of the most recent accounts on these issues [14], [16] and [6], and use the formalism proposed in the latter one as a means of illustration of our discussion.

2 Formalizing the Protocol

Following the approach of [3] we first have to identify the different principals involved in a protocol. Principals receive messages at one end and emit other messages at another end. Some principals will be considered to be "trustable" (i.e. to work according to their role in the protocol) and some not. Communication media are typically considered to be non-trustable, because messages can usually be intercepted, replayed, removed, or created by intruders. We will consider that this is the case in the following discussion. The set of untrustable principals is modeled as a single (black box) agent which is called the "external world" or, more concisely, the intruder. The intruder is modeled as a principal that may know some data initially and that will store and try to decrypt all data passed to him and thus in particular all information circulating on the communications media. The intruder will also be able to encrypt data to create new messages that will be sent to mislead other principals. But the intruder will be able to decrypt and encrypt data only with keys he knows. This modeling will in particular allow us to determine at any time which data are potentially known to the intruder under the chosen "trustability" hypothesis. The same protocol can be studied in terms of many different hypotheses. According to [3], the knowledge of the intruder is formalized as a set of data components. Data components range over domain C and sets of data components over domain S. Data components can be basic data, which may be cryptographic keys which take their values in domain KA (for asymmetric keys) or KS (for symmetric ones), or other basic data which take their values in domain D. Data components can also be obtained by composition using the pair operator which takes some data c_1 and some c_2 and returns the pair (c_1, c_2), or by encryption of some data c using key k which is noted c_k. The domains S and C are formalized as:

$S = C \cup S \mid \emptyset$	$B = K \mid D$
$C = C_K \mid (C, C) \mid B$	$K = KA \mid KS \mid K^{-1}$

modulo a few axioms[1] ($\cup$ is for example an ACUI operator with neutral $\emptyset$).

For illustration purposes, we use the Needham-Schroeder protocol which allows two principals, A and B, to perform mutual authentication. The protocol is initiated by one of the principals. This principal is A in our case. The protocol requires the use of a certification authority, S, in order to distribute public keys to principals requesting them. The protocol is composed of seven message exchanges between A, B and S: (1) $A \rightarrow S : (A, B)$; (2) $S \rightarrow A : (K_B, B)_{K_S^{-1}}$; (3) $A \rightarrow B : (N_A, A)_{K_B}$; (4) $B \rightarrow S : (B, A)$; (5) $S \rightarrow B : (K_A, A)_{K_S^{-1}}$; (6) $B \rightarrow A : (N_A, N_B)_{K_A}$; (7) $A \rightarrow B : (N_B)_{K_B}$.

It is assumed here that the principals A and B both know the public key K_S, and that S knows the public keys K_A and K_B. The keys K_A^{-1}, K_B^{-1}, K_S^{-1} are the corresponding private keys. They are respectively known by A, B and S, and only by them. N_A and N_B are fresh nonces.

The two first messages can for example be read as follows : (1) A sends a message to S to tell him or her that he or she is the principal A and wants to start an authentication procedure with B; no encryption is used; (2) S replies to the request by sending A the public key K_B of B; this message is encrypted with the private key K_S^{-1} which S is the only one to know and which thus authenticates the producer.

The formal specification of the protocol consists in the description of the role of each trustable agent and is given as a set of atomic actions. Sending and reception of a message are not synchronous. Consequently the transmission of a message is considered as two atomic actions, one for sending and one for receiving. Our modeling of the NS protocol will thus distinguish 14 different kinds of atomic actions. These actions will be identified using the labels drawn from $\mathcal{L} = \{1a, 1s, 2s, 2a, 3a, 3b, ..., 7a, 7b\}$. Each of the 14 labels n_X of $\mathcal{A}$ stands for one action: principal X sends or receives message n.

[1] See [3] for a more precise definition.

The system is defined as a pair (s_0, r) where s_0 is the initial global state, and r is a relation binding the global state before applying an action to the global state after applying the action. The relation r is defined using a predicate or logic formula p, defined on $(\mathcal{S} \times (\mathcal{L} \times C) \times \mathcal{S})$ where the domain for global states $\mathcal{S}$ is the Cartesian product, $\mathcal{S}_A \times \mathcal{S}_B \times \mathcal{S}_S \times \mathcal{S}_I$, of local state domains, i.e. $\mathcal{S}_A$, $\mathcal{S}_B$ and $\mathcal{S}_S$ for the three trustable principals, A, B and S, and $\mathcal{S}_I$ for the intruder. By definition $p(s, (l, m), s')$ is true if and only if the global state s is modified into s' upon firing the action labelled l for sending or receiving of message m. The set $\mathcal{S}_I$ is the domain S of data components defined in the previous section. Intuitively the state of the intruder is the set of data components that have been listened to on the communication line and that the intruder may use to build new messages. The state of a trustable principal is defined as an aggregate or a tuple describing the value of each local state variable. Here the states for the three trustable principals are defined as the following aggregates:

$S_A =$			$S_B =$		
	$slave$	$: Principal$		$master$	$: Principal$
	n_a	$: Nonce$		n_a	$: Nonce$
	n_b	$: Nonce$		n_b	$: Nonce$
	k_b	$: K$		k_A	$: K$
	$nonces$	$: set\ of\ Nonce$		$nonces$	$: set\ of\ Nonce$
	at	$: ProgramAddress$		at	$: ProgramAddress$
$S_S =$					
	key	$: Principal \to K$			
	at	$: ProgramAddress$			

Fields *at* are used in each state to hold the value of the abstract program counter for the algorithm executed by the corresponding principal. The field *slave* is used by principal A to store the identifier of the supposed requested principal. The field *master* is used in the same way to store the identifier of the supposed requester. The public and private keys of A, B, and S which are constant values will be noted $K_A, K_B, K_S, K_A^{-1}, K_B^{-1}$ and K_S^{-1}.

The predicate $p((s_A, s_B, s_S, s_I), (l, m), (s'_A, s'_B, s'_S, s'_I))$ that formalizes the protocol is defined as the disjunction of 14 predicates, i.e. one for each atomic action (1) $action(s_a.at, 1a, s'_a.at) \wedge s'_i = s_i \cup (A, s'_a.slave)$; (2) $action(s_s.at, 1s, s'_s.at) \wedge (d_1, d_2)\ known\_in\ s_i$; (3) $action(s_s.at, 2s, s'_s.at) \wedge s'_i = s_i \cup (s_s.key(d_3), d_3)$; (4) $action(s_a.at, 2a, s'_a.at) \wedge (s'_a.k_b, s_a.slave)_{K_S^{-1}}\ known\_in\ s_i$; (5) $action(s_a.at,\ 3a,\ s'_a.at) \wedge s'_i = s_i \cup (s'_a.n_a, A)_{s_A.K_b} \wedge s'_a.n_a \notin s_a.nonces \wedge s'_a.nonces = s_a.nonces \cup \{s'_a.n_a\}$; (6) $action(s_b.at, 3b, s'_b.at) \wedge (s'_b.n_a, s'_b.master)_{K_B}\ known\_in(s_i)$; (7) $action(s_b.at, 4b, s'_b.at) \wedge s'_i = s_i \cup (B, s_b.master)$; (8) $action(s_s.at, 4s, s'_s.at) \wedge (d_4, d_5)\ known\_in\ s_i$ (9) $action(s_s.at, 5s, s'_s.at) \wedge s'_i = s_i \cup (s_s.key(d_6), d_6)_{K_S^{-1}}$; (10) $action(s_b.at, 5b, s'_b.at) \wedge (s'_b.k_a, B)_{K_S^{-1}}\ known\_in\ s_i$; (11) $action(s_b.at, 6b, s'_b.at) \wedge s'_i = s_i \cup (s_b.n_a, s'_b.n_b)_{s_B.K_A} \wedge s'_b.n_b \notin s_b.nonces \wedge s'_b.nonces = s_b.nonces \cup \{s'_b.n_b\}$; (12) $action(s_a.at, 6a, s'_a.at) \wedge (s_a.n_a, s'_a.n_b)_{K_A}\ known\_in(s_i)$; (13) $action(s_a.at, 7a, s'_a.at) \wedge s'_i = s_i \cup (s_a.n_a)_{s_A.K_B}$; (14) $action(s_b.at, 7b, s'_b.at) \wedge$

$(s_b.n_b)_{K_B}$ *known_in* s_i, where by convention state variables that are not explicitly changed are supposed to be unchanged. Predicate *action* is used to specify sequencing constraints, i.e. control structure for each role. Here we assume that A (resp. B) is repeating an infinite loop $1a, 2a, 3a, 6a, 7a$ (i.e. $1b, 3b, 4b, 5b, 6b, 7b$). This complete formalization can be found in [3].

The first action (i.e. 1_A) describes A sending a pair composed of the identification of A and of the identification of the principal *id* for which the public key is requested. Each sending of a message m increases the knowledge of the intruder, i.e. $s_i' = s_i \cup m$. The value of *id* is not constrained in any way. This allows A to request any public key he wishes. The second action (i.e. 1_S) describes S receiving a pair of data. This pair can be the pair just sent by A or any pair of data known by the intruder. The second situation is meaningful if this can go undetected by A: here, there is no particular checking other than on the form of the message. Receiving a message m does not change the state of the intruder (i.e. $s_i' = s_i$), but the message should be deducible from the knowledge of the intruder (i.e. m *known_in* s_i). The third action (i.e. $2s$) describes S sending a pair composed of the public key of d and of the identifier d stored previously. The fourth action (i.e. 2_A) implicitly specifies that the received message is to be signed using K_s^{-1}.

Describing the protocol with as much accuracy as required is a quite difficult problem for modal logic approaches where the problem is known as the idealization problem: the modelization of the protocols requires some abstraction or adaptation of the protocol at hand in order to fit within the chosen logical framework. This problem is difficult to avoid as the formalism has to satisfy at the same time conflicting logical and practical properties.

The situation is less problematic for proof-based approaches relying on general purpose formal methods, because the logical constraints imposed by these formalisms are much weaker: it is typically possible and easy in these approaches to use as many variables as required, to describe control or sequencing constraints. The main problem is that the use of these features has a very negative impact on the size and automatization of the proof process.

Most recent approaches based on general purpose formal methods [14], [3] and [16] allow for the description of the protocol at a level of detail that is comparable to the approach illustrated here. The two main differences concern the way the intruder is modeled and the way the control structure of the protocol is described.

The intruder knowledge is here implictly modeled and axiomatized once for all, whereas in other approaches such as [14] the internal behavior of the intruder needs to be explicitly formalized and analyzed. This is typically done by describing the actions that the intruder can perform to exploit data. Paulson's approach lies somewhat in between (intruder actions are still explicitly modeled but some already proven theorems can be used during the verification).

Control structure is specified in the previous section by the means of the *action* predicate and the presence in the principal states of a control field, namely the *at* field. It is thus for example possible, by just changing the definition of the *action* predicate to specify a version of the same protocol that would allow for multi-session either for the master (i.e. A is allowed to start a new session before having completed a previous one), the slave or the both of them. But adding such a control structure adds more complexity and this has a cost when proof-based techniques are used. Thus in most approaches, such as Paulson's one or Meadows's one, this control structure is not specified which basically amounts to specifying a multi-session version of the protocol (i.e. taking the *action* predicate to be defined as *true* in our approach). Of course if the multi-session version works, it is likely that a single version will also work, but the converse is not true.

The situation is similar for modal logics: the control structure is wired in the formalism. Parallel multi-role for trusted principals is not supported by most modal logics. It is supported in [17], but it is then the only possible choice.

3 Expressing security properties

The way security properties are expressed is an even greater area of differentiation among various approaches. In this section we briefly present the approach described in [6]. In this approach a security property is described using a *filtering function* characterizing the visible actions and an automaton specifying the required sequencing of these visible actions. The filtering allows to focus on any particular session and on the relevant actions expected for this session: unexpected actions or actions carrying non coherent values will be abstracted away, and the security property can be more concisely and more simply expressed on remaining actions, which are called the visible actions. The automaton is to express constraints on visible actions. Here we consider one of the security properties expressed in [3] and referred to as the master authentication property. This property is expressed using the following filtering function:

$$\begin{array}{l} ff_{N_a}\ (((s_a, s_b, s_s, s_i), (l, msg), (s'_a, s'_b, s'_s, s'_i))) = \\ 3a\ if\ l = 3a \wedge s'_a.slave = B \wedge s'_a.n_a = N_a \\ 3b\ if\ l = 3b \wedge s'_b.master = A \wedge s'_b.n_a = N_a \\ 6b\ if\ l = 6b \wedge s_b.master = A \wedge s_b.n_a = N_a \\ 6a\ if\ l = 6a \wedge s_a.slave = B \wedge s_a.n_a = N_a \\ \varepsilon \text{ otherwise} \end{array}$$

This filtering function uses here only one parameter, N_a, which intuitively stands for a nonce characterizing the particular session of interest. This function identifies four visible actions, 3a, 3b, 6b, 6a, which correspond to the situation where the action is fired with the correct session parameter (e.g. the master is going through action 3a, believes he is talking to B and uses the nonce that characterizes the particular session of interest). The automaton just states that the visible action should be seen (by the global observer) always in the same order: 3a, 3b, 6b, 6a. The automaton thus corresponds to the regular expression $\underline{3a\ 3b\ 6b\ 6a}$

By definition the property is satisfied iff $\forall x.\forall t.t \in T \Rightarrow \widetilde{ff_x}(t) \in P$ where $\widetilde{ff}$ is the extension of ff to traces, T is the set of possible traces and P is the prefix closure of the language recognized by the automaton. Thus, if for some trace t of T, it is possible for the slave to answer to A using the correct nonce N_a, but in believing that he is talking to another (potentially corrupted) principal C with A not being aware of this (even after receipt of message 6) then we have $\widetilde{ff}_x(t) = 3a\ 6a$ which is not a prefix of $3a\ 3b\ 6b\ 6a$. This is exactly the situation for the flaw described in [3] for the same protocol (but for another authentication property). This property is then transformed automatically into a very simple invariant property. This invariant can be proven using proof-based techniques or can be automated as will be seen in the next section.

The problem of precisely expressing security properties using a modal logic is very similar to the problem of describing a protocol with adequate precision: basic modal operators are typically not general enough to cover all potential needs (e.g. the expression of freshness). But the problem is even more complex for more elaborate properties that are found in electronic commerce protocols and no modal logic has yet be proposed for this class of cryptographic protocols.

For general purpose formal methods the situation is quite different. Very few approaches laid down the basis for expressing sophisticated security properties: most approaches only provide for the expression of confidentiality; only a few provide for the expression of authentication properties; the first attempt to express more elaborate properties such as those found in electronic commerce was done in [4] using the approach just illustrated; another tentative has just been presented very recently in [8]. We nevertheless believe that the use of a filtering function and of an automaton as illustrated previously could be adapted to fit with most approaches. The only negative side-effect being that it would then possibly conflict with the corresponding proof automatization.

4 Bringing automatization

Now that we have discussed the ability to describe the protocols and security properties with adequate accuracy, we consider automatization of the verification. Most general purpose formal methods based approaches use proof-techniques for bringing some automatization: term-rewriting based techniques are for example used in Meadow's approach, and induction in Paulson's approach. As discussed in the previous section, using such an automatization has the drawback of puting some constraints on the formalization itself. Here we show that the various features used in the previous section (control structures, variables, etc.) do not necessarily prevent automatization.

Following [5] we first define an abstraction function $h : B \rightarrow B_0$ where B is the set of basic data (keys, nonces, numbers and identifiers), and B_0 is a finite subset of B which is to be proposed by the user when willing to verify a particular property. In practise this comes down to selecting a few values of interest (a few keys, a few nonces, etc.) and defining h to leave these values unchanged (i.e. $h(x) = x$) and to collapse other elements into one representative for each basic type, i.e. one for keys, one for nonces, etc. An abstract interpretation based technique is then applied where the abstraction function is derived from h and the verification is performed in an automatic and safe way using model-checking techniques on the abstract model. An algorithm is also provided for trying to compute the abstract model and the abstract properties in an automatic way. In fact the algorithm tries to prove that the same specification can be used for both the general protocol and its finite abstraction. The algorithm may fail to show that a particular sub-expression meets the associated sub-goal. In this situation, which is very rare in practice the user should then either prove manually using a proof tool that the problematic sub-expression indeed meets an automatically generated proof-obligation, or should provide a new sub-expression by himself.

With the current protocol specification the algorithm will in fact fail for action 3a and 6b on sub-expressions $s'_a.n_a \notin s_a.nonces$ and $s'_b.n_b \notin s_b.nonces$. We first notice that the algorithm would succeed for the first sub-expression if the value of $s'_a.n_a$, let us say a, would be such that $\hat{h}^{-1}(a) = \{a\}$. This is clearly not always true, but we can easily notice that $s'_a.n_a \notin s_a.nonces$ is the equivalent to $(s'_a.n_a = N_a \wedge s'_a.n_a \notin s_a.nonces \vee s'_a.n_a \neq N_a \wedge s'_a.n_a \notin s_a.nonces)$ and can thus be replaced by the weaker proposition $s'_a.n_a = N_a \wedge s'_a.n_a \notin s_a.nonces \vee s'_a.n_a \neq N_a$. The problem is similar with the second sub-expression and is handled in the same way. The algorithm then completes on this revised version. By construction, the specification of the abstract model is thus identical to the specification of the protocol itself but for the two relations describing the actions 3a and 6b. The new relation for 3a is for example $action(s_a.at, 3a, s'_a.at)$ $\wedge\ s'_I = s_I \cup (s'_a.n_a, A)_{s_A.K_b} \wedge s'_a.n_a = N_a \wedge s'_a.n_a \notin s_a.nonces \vee s'_a.n_a \neq N_a \wedge$ $s'_a.nonces = s_a.nonces \cup \{s'_a.n_a\}$.

The same algorithm is then applied to the authentication property and succeeds without any user intervention. Now we are in a position to verify the abstract property on the abstract model using model-checking techniques. The verification succeeds[2]. Since the abstraction is safe, we can conclude that the protocol also meets the expressed authentication property. Typically since only a few distinct keys, nonces and identifiers need to be distinguished to prove a particular property on a given protocol, the number of states used during the verification is very small according to model-checking standards (a few hundred for current real life protocols). Thus by integrating model-checking techniques it is possible to automatize a significant part of the verification process without any compromise on the precision of the protocol modeling and of security property expression. By keeping to pure proof-based techniques this would have been very difficult to achieve if not impossible.

[2] In fact the verification shows that B can apparently repeat the sequence of visible action $3b$ $6b$ more that once. This is due to the fact that the abstraction function transforms nonces in a way that does not preserve freshness property of nonces generated by B. It would be easy to avoid this situation by revisiting h to distinguish one more nonce, let say N_b. But this situation is clearly not problematic from the master perspective: what is important is that B answers at least once in a coherent manner. Thus we could as an alternative revisit the automaton by allowing a few more transitions that seem acceptable. This is what we assume in the following to simplify presentation.

5 Bringing understanding into the protocol design

The verification process can be used to generate a set of reachable global states, G. This set is very small according to model-checking standards but it is quite big for human standards. The verification thus does not provide a lot of insight into understanding in a very precise manner protocol correctness. We believe that this understanding is very important in many respects. First the verification is based on some hypotheses, and it is of prime importance to understand the precise impact of each one. Of course it is still possible to do the verification for all set of possible hypotheses but the number of such combinations is typically too big. Second, bringing such insight is very useful for designing or improving the protocol itself. It is furthermore a way of assessing the adequacy of the modeling of the protocol.

In order to achieve this, it is suggested in [6] to transform the abstract model by applying a second abstraction. The main advantage of the second model is that it is much easier to understand. This bears some similarities with the situation using invariant proof-based techniques: an invariant does not need to be precise enough to characterize the set of reachable states; it should be only precise enough for the proof to work.

For this we proceed as follows : (1) for each trustable principal x we propose a set $E^x = \{E_1^x, E_2^x, ..., E_{n_x}^x\}$ such that $\bigcup_{E_i^x \in E^x} \subseteq L_x$ where L_x is the set of reachable states for principal x; (2) we propose a finite set $S_i = \{s_i^1, s_i^2, ..., s_i^{n_i}\}$ where each element of S_i is some potential intruder knowledge (i.e. a value drawn from domain S); (3) we finally identify a subset G' of $E^{x_1} \times E^{x_2} \times ... \times E^{x_n} \times S_i$ where $\{x_1, x_2, ..., x_n\}$ is the set of trustable principals (i.e. $\{A, B, S\}$ for the protocol at hand) such that for each reachable global state $(s_{x_1}, s_{x_2}, ..., s_{x_n}, s_i)$ of G there exists a corresponding element $(E_{x_1}, E_{x_2}, ..., E_{x_n}, s_i')$ of G' where $s_{x_1} \in E_{x_1}, s_{x_2} \in E_{x_2}, ..., s_{xn} \in E_{x_{n1}}$ and $s_i \subseteq s_i'$. Intuitively, G' which has to be a very small set in order to achieve its objective, represents in a structured way the basic dependence between local states. The similarity with an invariant is more then just apparent and is discussed in [6].

Each subset $E_1^x, E_2^x, ..., E_{n_x}^x$ of E^x can be described by extension or more likely by intension. A particularly concise way is to characterize each of them using two logic formulae that apply respectively on states and transitions. The generated subset is then by definition the one that can be reached from the set of states characterized by the first logic formula without firing transitions drawn from a set of transitions characterized by the second one. Here for example we propose to use three subsets for A and three for B. For A we have (1) the subset of states characterized by $s_a.at = 71 \land s_a.nonces = \{N\}$[3], and $l = 3a \land s_a'.slave = B \land s_a'.n_a = N_a$; (2) the subset characterized by $s_a.at = 36 \land s_a.nonces = \{N, N_a\} \land (s_a.slave = B \land s_a.k_b = K_b) \land s_a.n_a = N_a$ and $l = 6a \land s_a'.slave = B \land s_a'.n_a = N_a$; (3) and the subset characterized by $s_a.at = 67 \land s_a.nonces = \{N, N_a\} \land (s_a.slave = B \land s_a.k_b = K_b) \land s_a.n_a = N_a \land s_a.n_b = N$ and *false*.

[3] Label nm stands by convention for the control point of principal x between the firing of action n_x and m_x (e.g. 71 is the control point of A between $7a$ and $1a$).

For B we have the subset characterized by the state condition $s_b.at = 73 \wedge s_b.nonces = \{N\}$, and $l = 3b \wedge s'_b.n_a = N_a$; the subset characterized by the state condition $s_b.at = 36 \wedge s_b.nonces = \{N\} \wedge s_a.n_a = N_a$, and $l = 6b$; and a third subset characterized by the state condition $s_b.at = 67 \wedge s_b.nonces = \{N\} \wedge (s_b.master = A \wedge s_b.k_a = K_a) \wedge s_a.n_a = N_a$ and $false$. The three first sets of states will be called respectively A_1, A_2, A_3 and the three other ones B_1, B_2, B_3. For the intruder we use three different knowledge values $s_i^1 = K_s \cup K_c^{-1} \cup (K_a, A)_{K_s^{-1}} \cup (K_b, B)_{K_s^{-1}} \cup (K_c, C)_{K_s^{-1}} \cup N$, $s_i^2 = s_i^1 \cup N \cup (N_a, A)_{K_b}$ and , $s_i^3 = s_i^2 \cup (N_a, N)_{K_a}$.

The set G' is $\{(A_1, B_1, s_i^1), (A_2, B_1, s_i^2), (A_2, B_2, s_i^2), (A_2, B_3, s_i^3), (A_3, B_3, s_i^3)\}$. The initial state of the abstract model is corresponding to (A_1, B_1, s_i^1). All moves from states corresponding to this element lead to states that are still associated to the same element, but for the visible action $3a$, which leads to an element corresponding to (A_2, B_1, s_i^2): the intruder knowledge has increased by $(N_a, A)_{K_b}$; principal A is just after the sending of visible action $3a$; it cannot proceed because he would need some data of the form $(N_a, n)_{K_a}$, but the intruder is unable to produce such a value (either by replay or construction). The only possible move is the visible action $3b$ which leads to a state corresponding to (A_2, B_2, s_i^2). The only possible move now is visible action $6b$ which leads to the sending of $(N_a, N)_{K_a}$ and to a state corresponding to (A_2, B_3, s_i^3). Now moves either lead to a state that corresponds to the same element (A_2, B_3, s_i^3), either a visible action $6a$ is performed and the new states that is reached corresponds to (A_3, B_3, s_i^3) and all subsequent moves lead to states that correspond to the same object.

The element of G' thus represent the five different phases that correspond to a particular session. This number would have been unchanged also in the event where a more discriminating abstraction function had been changed in the previous step (i.e. an abstraction function distinguishing one more nonce). For each of the five phases the element of G' precisely characterize the corresponding global states. A simple invariant expressed as the disjunction of five different sub-formulae could in fact be derived from this set. Another benefit here is that the correctness of the invariant can be checked automatically by using model-checking techniques.

6 Conclusion

Thus we have tried to show that by integrating various verification techniques it is possible to achieve at the same time automatization and precision in modeling. In particular we have used modal logic style to achieve conciseness in the description of the intruder knowledge, general purpose formal methods for providing a general framework and for applying abstract interpretation techniques, and model-checking based techniques for automatizing the verification process. Finally we have used a second abstraction to achieve better understanding of the protocol design. But the formalism that is used here is in some way over-killing: it is typically possible to verify the same properties using model-checking techniques or alternatively using pure proof-based techniques, filtering functions are more general then required, etc. The main objectives here was to show that the various objectives were achievable within a same framework. This framework is currently used for the verification of large protocols. We nevertheless believe that there is some room and need for designing a more dedicated formalism (i.e. typically a modal logic) that would achieve the same needs. This formalism would be more elegant and its theoretical properties could be studied more easily.

References

1. J.R. Abrial. The B-method for large software specification, design and coding. In *VDM'91*. Springer Verlag, 1991.
2. P. Bieber and N. Boulahia-Cuppens. Formal development of authentication protocols. In *BCS-FACS sixth Refinement Workshop*, 1994.
3. D. Bolignano. Formal verification of cryptographic protocols. In *Proceedings of the third ACM Conference on Computer and Communication Security*, 1996.
4. D. Bolignano. Towards the Formal Verification of Electronic Commerce Protocols. In *Proceedings of the 10 th IEEE Computer Security Foundations Workshop*. IEEE, June 1997.
5. D. Bolignano. Towards the Mechanization of Cryptographic Protocol Verification. In *Proceedings of the 9th International Conference on Computer-Aided Verification (CAV'97)*, June 1997.
6. D. Bolignano. Using abstractions for automatizing and simplifying the verification of cryptographic protocols. Technical report, Dyade, 1998.
7. M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8, 1990.
8. P. Syverson C. Meadows. A formal specification of requirements for payment transactions in the set protocol. In *Finacial Cryptography*, 1998.
9. P.C. Chen and V.D. Gligor. On the formal specification and verification of a multiparty session protocol. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 1990.
10. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(2):198–208, 1983.
11. G.Leduc, O. Bonaventure, E. Koerner, L. Léonard, C. Pecheur, and D. Zanetti. Specification and verification of a ttp protocol for the conditional access to services. In *Proceedings of the 12th Workshop on the Application of Formal Methods to System Development (Univ Montreal)*, 1996.
12. R.A. Kemmerer. Analyzing encryption protocols using formal verification techniques. In *IEEE Journal on Selected Areas in Communications*, volume 7(4), 1989.
13. G. Lowe. An attack on the needham-schroeder public-key protocol. In *Information Processing Letters*, 1995.
14. C. Meadows. Applying formal methods to the analysis of a key management protocol. In *Journal of Computer Security*, 1992.
15. J. K. Millen, S.C. Clark, and S.B. Freedman. The interrogator: Protocol security analysis. *IEEE Transactions on Software Engineering*, 13(2), 1987.
16. L. Paulson. The inductive approach to verifying cryptographic protocols. *J. Computer Security*, 1998.
17. E. Snekkenes. Roles in cryptographic protocols. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 105–119. IEEE, 1992.