Breaking Security Protocols as an AI Planning Problem

Fabio Massacci

Dipartimento di Informatica e Sistemistica
Università di Roma I "La Sapienza"
via Salaria 113, Roma
e-mail:massacci@dis.uniroma1.it

Abstract. Properties like confidentiality, authentication and integrity are of increasing importance to communication protocols. Hence the development of formal methods for the verification of security protocols.
This paper proposes to represent the verification of security properties as a (deductive or model-based) logical AI planning problem. The key intuition is that security attacks can be seen as plans. Rather then achieving "positive" goals a planner must exploit the structure of a security protocol and coordinate the communications steps of the agents and the network (or a potential enemy) to reach a security violation.
The planning problem is formalized with a variant of dynamic logic where actions are explicit computation (such as cryptanalyzing a message) and communications steps between agents. A theory of computational properties is then coupled with a description of the particular communication protocols and an example for a key-distribution protocol is shown.

1 Introduction

The development and formal verification of security protocols is one of the key requirement for modern distributed systems which face the need of secure communication over an insecure network.

The key idea is to rely on cryptographic primitives (shared and public key encryption, hashing etc.) to guarantee security properties viz. confidentiality, integrity or authentication (see [8,11,16,27] for an introduction).

Yet the presence of well designed cryptographic primitives is far from guaranteeing such security properties. On the contrary, most "secure" systems are not broken by cryptographic attacks, but rather by exploiting operational blunders [4] and logical errors in the protocol design [1,6,17]. The subtlety of logical faults may even be such that a protocol may become a standard (e.g. a version of the CCITT X.509) before being proved badly broken [6].

Logical errors can be prevented by formal verification and a number of techniques have been developed for the logical analysis of security protocols [19]. Formal systems abstract away the cryptographic details and develop a theory of actions (communication and computing) where some messages can be "opened" by an agent only if she has the corresponding (secret) key.

For a formal analysis one can use *logics of knowledge and belief* in the same spirit of the works of Halpern et al [9] for high-level reasoning about communication protocols. After the seminal paper on the BAN logic [6] those logics have become a large family, e.g. [2,12,29,30]. At the other end of the spectrum, researches have described *protocols as traces of atomic actions*, modeling explicitly the properties low-level of the network. The finite states approach (based on the process algebra CSP and model checking [17,26] or on states enumeration methods [19,20]) can be used to find attacks whereas the approach based on induction can be used to prove properties such as secrecy and authenticity [23].

1.1 Security Verification as a Planning Problem

At this stage one may start wondering: this is surely interesting but... what formal verification has to do with planning?

Undoubtedly the use of logic and theorem proving techniques for reasoning about action is a backbone of deductive planning [13,25,18,28,5] and the use of proof tactics borrowed from interactive theorem provers [22] is the basis for both tactical planning and the verification of security properties [23].

The similarity is tighter for verification systems based on traces: they have a sophisticated modeling of the actions available to the legitimate participants, of the computing power of a potential enemy and of his abilities to tamper (read, destroy or spoof) legitimate messages [17,19,20,23,26]. This is just (low-level) reasoning about action in disguise.

Yet one may argue that deductive planning and formal verification share the same tools (logic and theorem proving) but have different objectives. It is so because we are used to think that in a planning problem we *accomplish something "good"*: from stacking blocks on a table to moving a mechanical arm, from avoiding collisions between a moving robot and other objects, to complex scheduling of satellites. The second point is that we often assume that *multi-agent planning is cooperative.*

To represent the verification of security protocol as a classical AI planning problem we need to change slightly our perspective: we need a "disruptive" planner, who wants to exploit the available (legitimate or illegitimate) actions to achieve unwanted effects and somehow deceive some of the agents. The task of the planner is then to break the systems.

In a nutshell we simply need to *model security attacks as plans* in a suitable formalism for reasoning about actions, communications and computations. A planning task which is interesting for its characteristics:

- the planner must combine legitimate and "illegitimate" steps of the protocol to construct an attack, somehow coordinating the bad and good agents involved in the communication;
- "illegitimate" steps are strongly constrained: they must "look like" legitimate steps, for instance the attacker can redirect a message, or modify a message by replacing the name of the sender with her own name, but must always respect the format of the messages foreseen by the protocol;

– not all undesirable states correspond to attacks (i.e. valid plans) we are only interested in attacks where, according to some of the legitimate parties of the protocol, "everything seems to go smoothly";
– among possible action we have to make explicit the computational activity of the agents, and thus we have to cope with knowledge changing actions;
– we want to describe the initial condition sparingly (e.g. only the secrecy of some long term cryptographic keys) and let the planner figure out the rest;
– not all security violations are possible for a given protocol and thus we may fail to find a plan in that case.

If we chose a formalism based on dynamic logic [15] which has been successfully applied to planning (see e.g. [25,28,7]) we can use deduction to show that a plan (an attack) exists and, most important, use the proof to generate the attack. We can also try to prove that no plan exists and hence that the particular security property represented by the negation of the planning goal is attained.

Notice that we are not bound to deductive planning. Model-generation planning [14] may be equally well suited.

As an aside remark, computational complexity is not an issue here. At first because protocols are usually short and second and foremost because the plan construction is once-off. If we can find an attack to a protocol, the protocol will be revised or the threat model[1] will be changed. Indeed we are not using the planner for (repeated) actions but rather for verification and trading time (ex-ante) for avoiding problems (after implementations) is worth the trouble.

Reusing previous plan (attacks) to check a modified protocol can also be configured as a problem of re-usable planning: trying to adapt past plans to a modified description of the environment and available actions.

In the rest of the paper we give a high-level introduction to security protocols (§2) . On the technical side we introduce a dynamic logic for modeling communication and computation (§3). Then we explain how to transform it into a planning problem (§4), model the general communication and computing properties (§5) and discuss possible extensions (§6).

2 A High Level View of Security Protocols

We sketch some characteristics and goals of security protocols to make the paper self-contained[2]. Security protocols can be succinctly described as follows [8]:

> What distinguishes a cryptographic protocol from any other algorithm is the underlying model of computation. In a cryptographic protocol, two or more participants communicate with each other over a clearly defined communication network. Each participant may function asynchronously... and has access to a basic set of cryptographic utilities.

[1] The attacks we are worried about and that the protocol guarantee to fight off.

[2] An introduction to security protocols can be found in [8,21,16] and an high level analysis in [6,11]. A practical overview can also be found in [27].

Furthermore each participant has a basic computational power... may apply cryptographic transformation, make decision and generate messages. [...] a participant may combine a priori knowledge with the properties of the messages he generates and receives to determine a property of the communication system... In a worst case analysis of a protocol one must assume that a participant may try to subvert the protocol.

As stated above, one of the key aspects of cryptographic protocols is the use of cryptographic primitives (see [8,6]). Here, we can consider such cryptographic primitives at an high level of abstraction. The key intuition is to regard them as *computationally hard functions with trapdoors.*

For example, consider the case of encryption. At an high level it is simply a function from strings (messages) to strings which needs a subsidiary string (the key) to be computed. Inverting the function (getting the original messages from the encrypted one) without knowing the key is extremely hard (in some cases even impossible). The knowledge of the key provide us with a trapdoor through the computational barrier. Thus, if we give an encrypted message to a number of agents, we may assume than only those who have the key will be able to read it. We may exploit this assumption to achieve useful properties [21,27] in an anonymous and untrusted medium such an electronic network, where everybody can pretend to be anybody else.

For instance, to know whether *Alice* is alive, *Bob* can send a message encrypted with a key that only she knows. If he gets the original message in reply, then he can be "sure" that *Alice* has read it (indeed she has decrypted it). It seems simple, yet it is easy to get it wrong [6,1]: the key must be secret, it must not be distributed to the "wrong guys", *Bob* text must be difficult to guess etc.

In this framework there is a number of security properties and goals that we may want to establish. We (informally) describe some of them and refer to [1,6,11,31] for some of the subtleties involved:

Secrecy: a message is only disclosed to its intended recipients;
Freshness: a certain message is recent, and is not an old replay of a previously used (and possibly compromised) one;
Authenticity: a message, which should come from a certain participant of the protocol, did indeed come from its legitimate sender;
Proof of Identity: a certain type message may be uniquely attributed to a particular participants in the protocols;
Non-repudiation: a participant may not deny of having sent a message or having received a message.

Such properties are usually determined by the use of some cryptographic primitives. For instance proof of identity can be determined by using electronic signatures (e.g. public and private key cryptography).

2.1 A Practical Example: Challenge Response

To show some practical examples we use the standard notation [6] for representing security protocols. *Principals* (agents) involved in the communications are denoted by $A, B, C, E \ldots$. Typically E represents the environment (or a possible enemy) whereas A and B the legitimate parties of the protocol.

Messages, denoted by M, are constructed from unspecified atomic strings, principals' names, cryptographic keys etc. using functions $f(M_1, \ldots, M_n)$. Some functions, e.g. encryption, are denoted by special symbols and we follow such use in the sequel.

Nonces[3] are denoted by N (possibly with mnemonic indices) whereas keys for symmetric encryption are denoted by K and the corresponding encryption of message M is $\{M\}_K$. Private and public keys are sometimes denoted by SK and PK and the corresponding encryptions are $eSK\{M\}$ and $ePK\{M\}$, following [11] in the use of ISO notation. Messages concatenation is denoted by the standard $\{M, M'\}$.

A *protocol* is a sequence of steps $A \to B : M$, where M is a message:

1. $A \to B : M_1$
2. $B \to A : M_2$
3. $A \to B : M_3$

corresponds to a protocol where A starts by sending M_1 to B. After B has received M_1 he can send M_2 to A which finally replies with M_3.

Below are some challenge-response protocols from the ISO-10181 standard:

protocol 1 - Nonce-Crypto	protocol 2 - Crypto-Nonce
1. $A \to B : N_a$	1. $A \to B : ePK_B\{N_a\}$
2. $B \to A : eSK_B\{N_a\}$	2. $B \to A : N_a$

Loosely speaking, the "task" of both protocols is to "convince" A that B is "alive" or, more properly, that somebody which can use B's signature is alive.

Here the underlying cryptographic primitive is public key cryptography. The idea is that there are two keys: one secret (the private key SK_B), and one widely available (the public key PK_B). The high-level functioning of the cryptographic primitive can be easily described: if you encrypt a message with one key you need the other to decrypt it, and viceversa.

The left protocol is initiated by A which sends a nonce (i.e. a random challenge) to B which reply by encrypting the nonce with his private secret key (i.e. signing the challenge). The intuition is that only B knows his secret key and thus only B can "sign" N_a. Still A can easily verify it since the public key PK_B is known. So she takes the message that she receives back and tries to decrypt it with PK_B. If she gets back N_a than she can be "sure" that only B could have done it. So, if she gets it quickly, she could conclude that B is alive.

In the second protocol, A encrypts the nonce N_a with B's public key. Then B decrypts this message with his secret private key and sends back N_a. The same reasoning seems to apply here, but what happens if somebody else can guess N_a? For instance, if A just use a sequence of integers generated by a counter for subsequent N_as? The second protocol can therefore be broken by anybody who can guess A challenges (see [30]).

[3] A nonce="number used once" is (usually) just a random number, see [16,27].

3 A Dynamic Logic for Security Protocols

To represent formally the actions and the communication properties involved in a security protocols we use a variant of dynamic logic [15], since its use for planning is simple and well-known [25,28] and it suits well to such problem.

The basic components are four: *principals, messages, protocols and formulae.* In comparison with standard dynamic logic [15] protocols play the role of actions and messages could be compared to first order terms.

Principals and messages have been already introduced in §2.

Formulae, denoted by φ, ψ, are constructed from the atomic components $A\,\texttt{made}\,M$ (A synthesized message M), $A\,\texttt{recv}\,M$ (A received message M or M is a sub message in a message received by A), $A\,\texttt{sent}\,M$, $A\,\texttt{has}\,M$ or $A\,\texttt{has}\,M$ with the obvious meaning.

Composed formulae are obtained with the boolean connectives $\neg\varphi$, $\varphi \Rightarrow \psi$ and the modal one $[\pi]\varphi$, whose meaning is that after any possible run of protocol π, property φ holds. Other connectives are abbreviations such as $\wedge, \vee$ or $\langle\pi\rangle\varphi \equiv \neg[\pi]\neg\varphi$ which says that is it possible to run π and end in state where φ holds.

Protocols are formed as programs [15] from atomic actions: $A \rightarrow B : M$ which means that principal A attempted to sends a message M to B; A *makes* M when A tries to compose message M; and A *reads* M if A tries to cryptanalyze M.

The use of "attempt" in the intuitive explanations above is due to the particular nature of the problem (§2): A may try to send the message to B, but B may not actually receive it because the "enemy" intercepted it.

The (standard) operators are sequential composition ($\pi_1 ; \pi_2$), non deterministic choice ($\pi_1 \cup \pi_2$), iteration (π^*), converse (π^-) and test ($\varphi?$), where φ is a formula. Nondeterministic choice can be used to model the possibility of an opponent to replace some of the legitimate steps of the protocol and the converse operator π^- can be interpreted as "before running π".

Notice that the whole system could be reformulated using the notation and semantics of [28] in particular is sufficient to use the relations $\mathit{add} - r$ and $\mathit{delete} - r$ of [28] where r is instantiated to the relations $\cdot\,\texttt{made}\,\cdot$, $\cdot\,\texttt{recv}\,\cdot$ etc.

The semantics can be given by using Kripke models as done in [28] or using runs and interpreted systems [2,9,29]. There is not substantial difficulties except for the enforcement of the constraints due to cryptographic primitives.

4 The Design of the Planning Problem

To map our problem into a planning problem we need to:

- define explicitly the computational properties of principals i.e. define the actions for composition and (crypt)analysis of messages (and also what information the agents gain from these operations);
- represent the communications between agents by actions which can either satisfy general properties or be tailored to the particular protocol;

– formalize the ability and the behavior of a potential enemy (sometimes the network itself), which may interfere with the regular running of the communications protocols;
– identify the typology of security violations which we may consider as relevant goals G for planning attacks;
– finally the initial conditions IC will be typically determined by the (informal) assumptions of the protocol (e.g. private keys are secret, honest principals generates fresh nonces etc.).

Remark 1. Defining the (disruptive) behavior of the network or the "enemy" is essential for the problem to make sense and indeed any attacking plan to exists.

Indeed legitimate agents will try to follow the protocol as closely as possible unless they are mislead by somebody. Thus, for any (attacking) plan to exists we must devise this somebody which can deviate from the rules. The key point is that meaningful deviations must not be so arbitrary to be unmanageable: the attacker may destroy, replay or change messages but only to the extent that her messages still look like those of the original protocol.

The global properties of the network GC, viz. the computational properties, the communications possibilities and the behavior of the network (or the potential enemy), will be the classical post - and pre-conditions for the planning problem [25,28]. We sometimes also use the terminology of [24] and refer to constraints of the form $\varphi \Rightarrow [\pi]\psi$ as successor state axioms and to those of the form $\langle\pi\rangle\psi \ \Leftrightarrow\ \varphi$ as preconditions.

Once we have set the logical framework, the plan is the $\pi?$ such that $GC \models IC \Rightarrow \langle\pi?\rangle G$ as in [28].

4.1 Security Violations as Goals

A number of possible *planning goals*, i.e. security violations, can be devised in term of confidentiality, authenticity and freshness.

For instance a violation of *confidentiality* is represented by the goal $E\,\mathtt{has}\,M_s$ where M_s is a message supposed to be secret, such as the session key of a key K_{ab} of the Needham-Schroeder protocol (§2).

We can also represent *impersonation* either from the point of view of the initiator (A received a message apparently from B who is not there) or the respondent. In this case somebody can impersonate B and yet mislead A into thinking that everything went smoothly: she have sent all its messages and received all "right" messages "coming" from B, yet none of such message ever come from be. So she may simple have achieved the result of giving her credit card number to some villain... This state of affairs can be easily described with the following formula:

$$\bigwedge_i A\,\mathtt{sent}\,M_{ai} \wedge \bigwedge_j A\,\mathtt{recv}\,M_{bj} \wedge \bigwedge_j \neg B\,\mathtt{sent}\,M_{bj}$$

Where M_{ai} are the messages A is supposed to send B during a correct run of the protocol and M_{bj} are the one she is supposed to receive from B. We can weaken the bad effects we are planning by changing the $\bigwedge_k$ into a disjunctions or by dropping some conjuncts etc.

In some cases we may require something stronger than simply sending a messages: for instance, for *cash protocols*, it is not only important that A sent a message but she has actually made it (in practice has signed the main message). In this case we can replace $A\,\mathtt{sent}\,M$ with $A\,\mathtt{sent}\,M \wedge A\,\mathtt{made}\,M$ and similarly replace $\neg B\,\mathtt{sent}\,M$ with $\neg B\,\mathtt{made}\,M$.

Another issue is *freshness*: we may reach a state where everything is fine for A, who receives a new key, while B received an old compromised key.

$$A\,\mathtt{recv}\,M_a \wedge \neg A\,\mathtt{had}\,M_a \wedge B\,\mathtt{recv}\,M_b \wedge B\,\mathtt{had}\,M_b$$

It is worth noting that also here we have to cope with the frame problem. Moreover we can also use a formalization based on planning as model generation rather than deduction as in [14]. We leave further details to the full paper.

5 Modeling the Communication Environment

We inherit directly all the axioms and properties of standard dynamic logic with converse [15]. If we wanted to redefine the basic primitives in terms of the *add* and *delete* operator of [28] then we would have to import also their axioms.

5.1 Computational Abilities of Agents: Synthesis

The next step is the definition of the computational properties of the principals involved in the communication wrt the synthesis of messages.

In traditional security analysis, either based on authentication logics such as [6] or traces and states enumeration [20,26], it is assumed that all principals, including the potential enemy E, have the same computational power w.r.t. cryptographic primitives. For simplicity we follow the same approach here but we could also devise differentiate principals with different computational power by changing the applicable axioms.

The first part regards the ability of *composing messages*: we have traditional successor state axioms of the form $\varphi \Rightarrow [\pi]\psi$ [24,25] and local constraints:

$$\begin{aligned} \neg\, A\,\mathtt{has}\,M &\Rightarrow [A\ \mathit{makes}\ M]A\,\mathtt{made}\,M \\ A\,\mathtt{made}\,M &\Rightarrow A\,\mathtt{has}\,M \end{aligned}$$

Notice that we put the negation of $\cdot\,\mathtt{has}\,\cdot$ in the precondition of the successor state axioms because we want to be conservative: we want to derive that A "constructed" M only if this was really necessary.

The next set of axioms depends on the particular function we need to use in our protocols and are just preconditions:

$$\langle A\ \mathit{makes}\ f(M_1,\ldots,M_n)\rangle\top \;\Leftrightarrow\; \Phi^f_{make}(A\,\mathtt{has}\,M_1,\ldots,A\,\mathtt{has}\,M_n)$$

Typically Φ^f will be a boolean conjunction of its arguments, thus matching the intuition that for constructing a function we need to have all its arguments.

For instance in the case of encryption and concatenation:

$$\begin{aligned}\langle A\ \mathit{makes}\ \{M\}_K\rangle\top &\Leftrightarrow A\,\mathtt{has}\,M \wedge A\,\mathtt{has}\,K\\ \langle A\ \mathit{makes}\ \{M_1,M_2\}\rangle\top &\Leftrightarrow A\,\mathtt{has}\,M_1 \wedge A\,\mathtt{has}\,M_2\end{aligned}$$

This is equivalent to say that to encrypt a message one, of course, need both the message and the key.

Atomic messages, such as nonces and keys, may have particular properties. For instance we may want private key of the challenge response protocols (§2) to be unguessable by the enemy:

$$\langle E\ \mathit{makes}\ PK_B\rangle\top \Leftrightarrow -$$

If we use deductive planning just to look for the existence of plans, we can drop the axiom altogether.

On the contrary the nonces of the Crypto-Nonce challenge response protocol may or may not be guessable according the axiom we choose:

$$A\,\mathtt{made}\,N_a \Rightarrow \langle E\ \mathit{makes}\ N_a\rangle\top$$

implies that it is possible for E to guess the nonce generated by A. For instance this may happen if A uses simply a counter to generate her nonces.

On the contrary the formula below implies that nobody can guess A nonces[4].

$$A\,\mathtt{made}\,N_a \Rightarrow \neg\langle E\ \mathit{makes}\ N_a\rangle\top$$

5.2 Computational Abilities of Agents: Cryptanalysis

The ability of *analyzing messages* is essential. We have some general successor state axioms and some local conditions such as

$$\begin{aligned}\top &\Rightarrow [A\ \mathit{reads}\ f(M_1,\ldots,M_n)](A\,\mathtt{has}\,M_1\wedge\ldots\wedge A\,\mathtt{has}\,M_n)\\ A\,\mathtt{recv}\,M &\Rightarrow A\,\mathtt{has}\,M\end{aligned}$$

Notice that the axiom just state *if* A can analyze a function f then she can recover its components and we can well have functions which she cannot analyze. The precondition axioms for the cryptanalysis of a functions depends of course on the function itself and has the general form:

$$\langle A\ \mathit{reads}\ f(M_1\ldots M_n)\rangle\top \Leftrightarrow A\,\mathtt{has}\,f(M_1\ldots M_n)\wedge\Phi^f_{read}(A\,\mathtt{has}\,M_1\ldots A\,\mathtt{has}\,M_n)$$

For the limited version of public key cryptography we used in our challenge response protocol (§2) we can defined the following properties:

$$\begin{aligned}\langle A\ \mathit{reads}\ eSK\{M\}\rangle\top &\Leftrightarrow A\,\mathtt{has}\,eSK\{M\}\wedge A\,\mathtt{has}\,PK\\ \langle A\ \mathit{reads}\ ePK\{M\}\rangle\top &\Leftrightarrow A\,\mathtt{has}\,ePK\{M\}\wedge A\,\mathtt{has}\,SK\end{aligned}$$

[4] In theory the possibility of guessing the right answer is never zero but, for practical purposes, we may be satisfied that it is negligible.

We have the property of agents to *remember* the data they have got. We can represent it as follows (π_0 is any atomic action):

$$\begin{array}{ll} A\,\texttt{has}\,M \Rightarrow [\pi_0]A\,\texttt{has}\,M & \quad A\,\texttt{made}\,M \Rightarrow [\pi_0]A\,\texttt{made}\,M \\ A\,\texttt{had}\,M \Rightarrow [\pi_0]A\,\texttt{had}\,M & \quad A\,\texttt{recv}\,M \Rightarrow [\pi_0]A\,\texttt{recv}\,M \end{array}$$

Remark 2. Properties of key and nonces must be carefully added since they may easily lead to inconsistency.

We can also assume that all principals have the names of other principals,i.e. that $A\,\texttt{has}\,B$ is an axiom.

5.3 Modeling the Communication, the Network and the Enemy

Some postconditions and some preconditions are independent of the protocol. The first axioms provide us with a simple mechanism for handling freshness:

$$\begin{array}{r} A\,\texttt{has}\,M' \Rightarrow [A \to B\colon M]A\,\texttt{had}\,M' \qquad\qquad\qquad\quad \\ A\,\texttt{has}\,M' \Rightarrow [B \to A\colon M](A\,\texttt{recv}\,M \Rightarrow A\,\texttt{had}\,M') \quad\; \\ \neg\, C\,\texttt{had}\,M' \Rightarrow [A \to B\colon M]\neg\, C\,\texttt{had}\,M' \quad \text{for } C \neq A, B, E \end{array}$$

In the second implication we impose that A must have received M. If M is going to be intercepted by E then A may not realize that "time is passing" (i.e. some external event is happening).

Next we have *non-interference* among principals (only the intended recipient of a message can receive it) and *sending*, where $C \neq B, E$:

$$\begin{array}{r} \neg\, C\,\texttt{recv}\,M \Rightarrow [A \to B\colon M]\neg\, C\,\texttt{recv}\,M \\ \top \Rightarrow [A \to B\colon M]A\,\texttt{sent}\,M \qquad\quad \end{array}$$

The other *communication properties* depends on the protocol and the choice of the explicit or implicit encoding of the enemy as the network.

We can choose to *explicitly model the enemy as she were in charge of the network*. This is usually done in traced based systems [17,19,26] and is common in the informal analysis of protocols. In this case, for each step $A \to B : M$ we must add the following preconditions

$$\begin{array}{rcl} \langle E \to B\colon M\rangle\top & \Leftrightarrow & E\,\texttt{recv}\,\{B, M\} \\ \langle A \to E\colon \{B, M\}\rangle\top & \Leftrightarrow & A\,\texttt{has}\,M \wedge \Phi_{send}(A, M, protocol) \end{array}$$

In a nutshell, the enemy (the network) can forward any message that she has received, while "good" principals should follow the protocol (or at least believe that they are following it).

The formula $\Phi_{send}(A, M, protocol)$ states that this is the correct time for A to send M. So, if M is supposed to be the n-th message in the "normal"protocol run then $\Phi_{send}(A, M, protocol)$ describes A's viewpoint up to the $n-1$th message: A has sent all message she was supposed to sent and she has received all messages she was supposed to receive (if the protocol was followed by all parties involved).

For instance in the Crypto-Nonce challenge (§2) we have:

$$\langle B \to A \colon N_a \rangle \top \;\Leftrightarrow\; B \,\texttt{has}\, N_a \wedge B \,\texttt{recv}\, ePK\{N_a\}$$

We must also add the following state successor axioms:

$$\top \Rightarrow [A \to E \colon M] E \,\texttt{recv}\, M \qquad \top \Rightarrow [E \to B \colon M] B \,\texttt{recv}\, M$$

In this case we can assume that, once that the enemy has decided to deliver the message, she will deliver it, and similarly that all messages sent to the network at least arrive there. Then the enemy may decide not to pass them forward.

If we choose the *implicit modeling* of the enemy then our formalization requires more care. At first we may assume that the enemy is *tapping the lines* the network and *sending spoof messages*:

$$\top \Rightarrow [A \to B \colon M] E \,\texttt{recv}\, M \qquad E \,\texttt{has}\, M \Rightarrow \langle E \to A \colon M \rangle \top$$

Remark 3. The axioms for spoof messages is too general. Indeed we are only interested in M which respects the format of the messages of the desired protocols. It can be replaced by a series of more definite axioms.

With this replacement, if we limit the number of nonces, keys and agents, we can map the system into PDL and make it decidable. We must transform the precondition axioms for the steps of the protocol into a set of axioms of the form:

$$\langle A \to B \colon M \rangle \Phi^i_{rec}(A, M, \mathit{protocol}) \;\Leftrightarrow\; A \,\texttt{has}\, M \wedge \Phi_{send}(A, M, \mathit{protocol})$$

The different $i = 1, \ldots m$ will determine the possible outcomes of the actions: they could be $A \,\texttt{recv}\, M$ or $\neg A \,\texttt{recv}\, M$. If also the enemy may and may not receive messages then we may have more cases.

In alternative we can assume that messages are always delivered, replace Φ^i_{rec} with $\top$, add then axiom $[A \to B \colon M] B \,\texttt{recv}\, M$ and then model the "blocking" of messages as agents that do not respond[5].

Finally, following an idea of [23], we can also add *oops messages* where a principal "accidentally" looses the session key at the end of the protocol.

6 Discussion

In this paper we have shown how the formal verification of security protocols can be naturally represented as an interesting AI planning problem.

Using a dynamic logic one can represent the communication and computing actions which can be used by legitimate and illegitimate participants in a protocol. Then finding an attack is equivalent to find a plan which leads to a state which violates some security requirement. In the present formalization all conditions are (implicitly) universally quantified and, by setting the number

[5] Notice that we have a possibility formula for the precondition axiom so we may validate Φ_{send} without actually choosing the subsequent step.

of different nonces, keys and principals we could transform it into a problem in dynamic propositional logic by grounding the properties on a subset of the Herbrand universe, in a fashion similar to the technique used in [14], and thus getting a decidable problem.

For a full modeling of protocols with potentially infinite participants or runs, such as [23] further developments are necessary. The simplest way is to introduce actions for generating new nonces. Such actions would also make the model closer to the semantics of [2,29]. The (necessary) trade off is that "term-creating" actions will affect the decidability of the planning problem.

A further step towards a full-fledge modeling is the introduction of time: a number of protocols use time-stamps rather than nonces and reasoning about actions and time becomes essentials. Also in this direction AI techniques are available [3] and the corresponding planning methods could be applied.

Acknowledgments

A large part of this work has been carried while the author was at the Computer Laboratory, University of Cambridge (UK). Discussions with A. Gordon, R. Needham and L. Paulson on security protocols and L. Carlucci Aiello, G. De Giacomo and D. Nardi on planning were invaluable. A special thank to the Computer Laboratory and Larry Paulson for their hospitality in Cambridge. This work have been partly supported by MURST-40% grant.

References

1. M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Trans. on Software Engineering*, 22(1):6–15, 1996.
2. M. Abadi and M. Tuttle. A semantics for a logic of authentication. In *Proc. of the 10th ACM Symp. on Principles of Distributed Computing*, pp. 201–216, 1991.
3. J. Allen. Towards a general theory of action and time. *AIJ*, 23:123–154, 1984.
4. R. Anderson. Why cryptosystems fail. In *Proc. of the 1st ACM Conf. on Communications and Computer Security*, pp. 217–227. ACM Press, 1993.
5. S. Biundo. Present-day deductive planning. In C. Bäckstrom and E. Sandewall, editors, *Current Trends in AI Planning*, pp. 1–5. ISO Press, 1994.
6. M. Burrows, M. Abadi, and R. Needham. A logic for authentication. *ACM Trans. on Computer Systems*, 8(1):18–36, 1990. Also available as Res. Rep. SRC-39, DEC - System Research Center, 1989.
7. G. De Giacomo and M. Lenzerini. PDL-based Framework for Reasoning about Actions In *Proc. of the Italian Conf. on Artificial Intelligence (AI*IA-95)*, vol. 992 of *LNAI*, pp. 103–114. Springer-Verlag, 1995.
8. R. De Millo, L. Lynch, and M. Merrit. Cryptographic protocols. In *Proc. of the 14th ACM Symp. on Theory of Computing (STOC-82)*, pp. 383–400, 1982.
9. R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge.* The MIT Press, 1995.
10. M. Fitting. *Proof Methods for Modal and Intuitionistic Logics.* Reidel, 1983.
11. D. Gollmann. What do we mean by entity authentication. In *Proc. of the 15th IEEE Symp. on Security and Privacy*, pp. 46–54. IEEE Comp. Society Press, 1996.

12. L. Gong, R. Needham, and R. Yahalom. Reasonign about belief in cryptographic protocols. In *Proc. of the 9th IEEE Symp. on Security and Privacy*, pp. 234–248. IEEE Comp. Society Press, 1990.
13. C. Green. Application of theorem proving to problem solving. In *Proc. of the 1st Internat. Joint Conf. on Artificial Intelligence (IJCAI-69)*, pp. 219–239, 1969.
14. H. Kautz and B. Selman. Planning as satisfiability. In *Proc. of the 10th European Conf. on Artificial Intelligence (ECAI-92)*, pp. 359–363. John Wiley & Sons, 1992.
15. D. Kozen and J. Tiuryn. Logic of programs. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, vol. II, chap. 14, pp. 789–840. Elsevier Science, 1990.
16. A. Liebl. Authentication in distributed system: A bibliography. *Operating Systems Review*, 27(4):31–41, October 1993.
17. G. Lowe. Some new attacks upon security protocols. In *Proc. of the 10th IEEE Computer Security Foundations Workshop*, pp. 162–169. IEEE Comp. Society Press, 1996.
18. Z. Manna and R. Waldinger. How to clear a block: Plan formation in situational logic. *J. of Automated Reasoning*, 3:343–377, 1987.
19. C. Meadows. Formal verification of cryptographic protocols: A survey. In *Advances in Cryptology - Asiacrypt 94*, vol. 917 of *LNCS*, pp. 133–150. Springer-Verlag, 1995.
20. C. Meadows. Analyzing the Needham-Schroeder public key protocol: A comparison of two approaches. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *Proc. of the 4th European Symp. on Research in Computer Security*, vol. 1146 of *LNCS*, pp. 351–364. Springer-Verlag, 1996.
21. R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.
22. L. Paulson. *Isabelle: A Generic Theorem Prover*, vol. 828 of *LNCS*. Springer-Verlag, 1994.
23. L. Paulson. Proving properties of security protocols by induction. Technical Report TR409, Computer Laboratory, Univ. of Cambridge (UK), 1996.
24. R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, ed., *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pp. 359–380. Academic Press, 1991.
25. S. Rosenschein. Plan synthesis: A logical perspective. In *Proc. of the 7th Internat. Joint Conf. on Artificial Intelligence (IJCAI-81)*, pp. 331–337, 1981.
26. S. Schneider. Security properties and CSP. In *Proc. of the 15th IEEE Symp. on Security and Privacy*, pp. 174–187. IEEE Comp. Society Press, 1996.
27. B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, 1994.
28. W. Stephan and S. Biundo. A new logical framework for deductive planning. In *Proc. of the 13th Internat. Joint Conf. on Artificial Intelligence (IJCAI-93)*, pp. 32–38. Morgan Kaufmann, 1993.
29. P. Syverson and P. van Oorschot. On unifying some cryptographic protocols logics. In *Proc. of the 13th IEEE Symp. on Security and Privacy*. 1994.
30. G. Wedel and V. Kessler. Formal semantics for authentication logics. In Elisa Bertino, Helmut Kurth, Giancarlo Martella, and Emilio Montolivo, editors, *Proc. of the 4th European Symp. on Research in Computer Security*, vol. 1146 of *LNCS*, pp. 219–241. Springer-Verlag, 1996.
31. J. Zhou and D. Gollmann. Observations on non-repudiation. In Kim Kwangjo and Matsumoto Tsutomu, editors, *Advances in Cryptology - Asiacrypt 96*, vol. 1163 of *LNCS*, pp. 133–144. Springer-Verlag, 1996.