# Is it possible to decide whether a cryptographic protocol is secure or not ?

Hubert Comon and Vitaly Shmatikov

**Abstract —**

**We consider the so called "cryptographic protocols" whose aim is to ensure some security properties when communication channels are not reliable. Such protocols usually rely on cryptographic primitives. Even if it is assumed that the cryptographic primitives are perfect, the security goals may not be achieved: the protocol itself may have weaknesses which can be exploited by an attacker. We survey recent work on decision techniques for the cryptographic protocol analysis.**

## 1. Introduction

Security questions are not new. They become increasingly important, however, with the development of the Internet. For example, the classical access control problem, which has been studied in the context of operating systems (*e.g.*[Lam74, HRU76]), becomes more complex in a distributed environment where communication channels are not reliable.

How is it possible to secure communications on insecure channels? As we will see, (perfect) cryptographic primitives are a useful tool but security of the primitives does not guarantee security of the protocols. Several protocols had been thought to be secure... until a simple attack was found (see [CJ97] for a survey). Therefore, the question of whether a protocol indeed achieves its security goals becomes crucial.

Until recently, most of the research in protocol analysis was devoted to finding attacks on known protocols, but very few works addressed proof techniques for protocol correctness. This was partly due to the absence of adequate formal models for distributed communications in a hostile environment. In the past 5 years or so, there were proposed several formal models for security protocols (a rough description of the models can be found in section 2.). This opened the way for the use of formal methods and formal analysis of protocols. In this survey, we address the problem of effectiveness of such methods. What can we expect? For what class of protocols are there decision algorithms for security questions?

After explaining the communication and protocol models in section 2., we discuss the attacker model in section 3.. We then survey the techniques: general techniques (which do not necessarily yield decision algorithms) in section 4., finite state analysis (which is mainly useful for finding attacks, but does not yield correctness proofs) in section 5., and, finally, decision results are surveyed in the core of the paper, section 6..

## 2. Abstract protocol modeling

In the presence of insecure communication channels, an attacker may be able to observe network traffic and/or intercept messages, modify them in transit, and construct fake messages. In this context, securing communication relies on a set of basic functions that we will refer to as *cryptographic primitives*. For example, an encryption primitive can be used to encode messages prior to tranmission on an insecure channel in such a way that the original message content (*cleartext*) can only be retrieved by recipients who possess the "right" decryption key. A number of cryptographic primitives have been designed to achieve information security goals such as secrecy, integrity, authentication, etc.

The analysis techniques discussed in this survey assume *perfect cryptography*. This means that cryptographic primitives are considered as black boxes satisfying certain properties, as described in section 2.1. below. This assumption by itself does not ensure security of the protocols. Even if all cryptographic primitives used by the protocol are perfectly secure, the protocol itself may have weaknesses which can be exploited by an attacker, as described, *e.g.*, in the Clark and Jacob survey [CJ97]. Typically, an attacker can observe and/or participate in some of the protocol sessions and use the knowledge obtained from these sessions when acting as a participant in subsequent sessions, perhaps impersonating some of the agents. We will give examples of this below.

This paper considers the following problem: is it possible to decide, assuming perfect cryptography, whether a given protocol is secure or not?

We must, of course, be more precise about what is a protocol and what is meant by "secure." Informally, a *protocol* is a conversation between two or more agents (also called *principals*) that aims to guarantee certain *security properties* even if a malicious party has access to the communication channel. A more formal definition is given in section 2.3.. In section 2.4., we describe common security properties such as *secrecy* and *authentication*, and give some examples.

### *2.1. Cryptographic primitives*

In this section, we discuss abstract modeling of cryptographic primitives such as encryption and one-way functions. Other primitives such as digital signatures can be modeled in a similar way.

**Symmetric encryption.** Suppose that Alice and Bob share a secret value $K$, which is not known to anybody else. The problem of establishing such a shared secret is beyond the scope of this survey —there are many protocols for achieving this, going back to the Diffie-Hellman key exchange protocol [DH76]. If Alice wants to communicate privately with Bob, she encrypts her messages with the secret $K$, producing a (symmetrically encrypted) ciphertext, which we will write as $\{m\}_K^{\leftrightarrow}$. As part of the perfect encryption assumption, we assume that the attacker cannot learn *anything* about $m$ from $\{m\}_K^{\leftrightarrow}$ unless he knows $K$. In particular, an attacker cannot learn anything by comparing ciphertexts.

Moreover, the attacker cannot construct $\{m\}_K^{\leftrightarrow}$ unless he holds $K$ and $m$. The attacker may be able, however, to obtain $\{m\}_K^{\leftrightarrow}$ from messages sent by other participants and replay it without learning $m$. Note that, in theory, the attacker can build all possible keys given a particular key length, and try to decrypt the message with every possible key. The perfect encryption assumption is an idealization of the fact that the attacker has only a very low probability of obtaining the cleartext of an encrypted message within a reasonable time.

**Public-key encryption.** The symmetric encryption scheme is not practical in many situations since every pair of principals willing to communicate must share a secret. This is the motivation for public-key encryption schemes, of which RSA [RSA78] is the best known. In a public-key encryption scheme, every principal has its own key pair, consisting of a public key $K$ (used for encrypting messages), and a private key $K^{-1}$ (used for decryption). Everybody is allowed to learn the public key $K$, but the private key is known to its owner only. Therefore, any principal can encrypt messages with $K$, producing ciphertext $\{m\}_K^{\rightarrow}$, but only the principal who knows $K^{-1}$ can decrypt $\{m\}_K^{\rightarrow}$ to retrieve $m$. The perfect encryption assumption in this case states that, again, it is impossible to learn $m$ from $\{m\}_K^{\rightarrow}$ without knowing $K^{-1}$.

**One-way functions.** Suppose an agent wants to send a long file and be sure that the file is not altered during communication. This can be achieved by sending a digest of the file in a secure way (*e.g.*, digitally signed by the sender). The recipient can then check the integrity of the received message by computing its digest and comparing it with the sender's digest. For this purpose, many protocols make use of *one-way functions*, also called digest functions or *hash functions*. The most widely used hash functions are MD5 [R92] and SHA-1 [NIST94]. It is assumed that one-way functions cannot be inverted in the sense that it is computationally infeasible to compute $m$ given $h(m)$, or find $m'$ such that $h(m') = h(m)$.

**Nonces.** To prevent an attacker from recording messages transmitted as part of one protocol session and replaying them in another session, messages often include nonces. A *nonce* is a value used no more than once for the same purpose [HAC]. We assume that a nonce is a randomly generated value that satisfies the following properties:

**Fresh** If two nonces are generated by different principals or at different times, then they are different.

**Unpredictable** An agent or the attacker cannot guess the value of a nonce generated by another agent (although it may able to learn it by analyzing protocol messages).

Many protocols only require freshness, in which case nonces can be replaced by *time stamps*, which we will not consider here.

The decision techniques surveyed in this paper assume, unless explicitly stated otherwise, that neither encryption, nor one-way functions satisfy any algebraic properties. If viewed as term constructors, cryptographic operators form a free term algebra. This assumption does not hold for many functions used in cryptographic applications. For example, `xor` is self-canceling ($\mathtt{xor}(\mathtt{x},\mathtt{xor}(\mathtt{x},\mathtt{y})) = \mathtt{y}$), and basic RSA satisfies $\mathrm{sig}_{\mathrm{pk}(A)}(\{m\}^{\rightarrow}_{\mathrm{pk}(A)}) = m$ where $\mathrm{sig}_{\mathrm{pk}(A)}(x)$ is agent $A$'s public-key signature of $x$. There is a wide class of encryption schemes and hash functions, however, for which the free algebra assumption is realistic.

To summarize our view of cryptography, we consider cryptographic functions as abstract black boxes satisfying certain properties. In our model, there is no notion of probability or partial data - the attacker either does not know a value, or knows all bits with 100% certainty. Cryptanalysis attacks that rely on probabilistic properties of cryptographic functions are beyond the scope of the methods considered in this survey. In section 3.3., we briefly mention recent work on more realistic formal models of cryptography.

### *2.2. Term algebra*

In our abstract model, protocol messages are terms constructed out of:

- Plaintext messages $m$;
- Nonces;
- Pairing of two messages $\langle M_1, M_2\rangle$ (or, more generally, tupling);
- One-way, unary functions applied to messages $h(M)$;
- Encrypted messages constructed from plaintext $M$ and key $k$. In general, for symmetric encryption we can view $k$ as an arbitrary term, which provides support, *e.g.*, for symmetric *session keys*, *i.e.*, keys which are generated as part of each instance of the protocol. We will distinguish between public-key and symmetric encryption by using two distinct notations $\{M\}_k^{\rightarrow}$ and $\{M\}_k^{\leftrightarrow}$, respectively. Terms are constructed in the same way in both cases, the only difference is decryption: to decrypt $\{M\}_k^{\leftrightarrow}$, it is necessary to know $k$, whereas to decrypt $\{M\}_k^{\rightarrow}$, it is necessary to know $k^{-1}$.

### *2.3. Protocol specification*

A protocol is a process parametrized by a (fixed and finite) set of agents who act as participants. Their names are given as distinct variables $(A, B, \ldots)$. Protocol specification consists of a finite sequence of rules of the form $A \rightarrow B : M$ where message $M$ is syntactically constructed as described in section 2.2.. The intended (informal) meaning is that $A$ sends to $B$ message $M$ on a public, insecure channel. The names which are used in the $i$th rule of the protocol refer to the names used in previous steps of the protocol, often in a somewhat ambiguous way, which has to be made precise in the formal models. An *instance* of the protocol, also called a *session* is the image of the protocol by a substitution assigning concrete values to all variables.

### *2.4. Security properties*

While there are many properties that a security protocol may aim to guarantee, in this survey we will be concerned mainly with *secrecy* and *authentication*.

**Secrecy** There are many definitions of secrecy, and the relationship between them is not clear [Aba99]. For the purposes of this survey we will say that a protocol preserves secrecy of a datum $d$ is the attacker cannot learn the value of $d$ by interacting with the protocol within the framework of the conventional Dolev-Yao model as described in section 3.. The goal of protocol analysis is then to determine if there exists a protocol trace in which the attacker learns the value of $d$.

It is worth observing that this notion of secrecy is not adequate for, *e.g.*, electronic voting, where possible values of the vote are known in advance and the goal of the protocol is to preserve the confidentiality of the association between a voter and his/her chosen value.

**Authentication** There are also many definitions of authentication (see, *e.g.*, [Low97]). In a nutshell, an event $e$ authenticates agent $A$ if $e$ can occur only if a previous message originated from $A$. The purpose of authentication is to ensure another agent $B$ that he is indeed talking with $A$.

Both secrecy and authentication are *trace properties*, *i.e.*, their violations can be found by looking at a single execution trace of the protocol. If the protocol process running in parallel with the attacker process is viewed as a state transition system, the protocol analysis problem for trace properties can be stated as a *reachability* problem, *i.e.*, the problem of determining if the state in which the property is violated is reachable from the protocol's initial state.

There exist security protocols designed to achieve other properties such as fairness, anonymity, non-repudiation, no denial of service, among others, but they are beyond the scope of this survey.

### *2.5. Example*

The following protocol is perhaps the most (in)famous one in the literature on formal analysis of security protocols. It's the (simplified) version of the Needham-Schroeder public-key mutual authentication protocol [NS78]:

$$
\begin{array}{lllll}
1. & A \rightarrow B & : & \{A, N_A\}_{\vec{K_B}} \\
2. & B \rightarrow A & : & \{N_A, N_B\}_{\vec{K_A}} \\
3. & A \rightarrow B & : & \{N_B\}_{\vec{K_B}}
\end{array}
$$

In the first message agent $A$ (Alice) sends to agent $B$ (Bob) her name together with a nonce $N_A$, encrypting the pair with Bob's public key $K_B$. Bob replies by sending back nonce $N_A$, together with his own nonce $N_B$, encrypting the pair with Alice's public key $K_A$. Finally, Alice sends back Bob's nonce encrypted with $K_B$.

The goal of the protocol is mutual authentication. After completing the protocol, Alice and Bob should be confident that they are talking to each other. More formally, Alice, upon receiving the second message, should be confident that this message was indeed sent by Bob (since only Bob could decrypt Alice's first message and learn the value of $N_A$). Bob, upon receiving the third message, should be confident that it was Alice who sent message $\{A, N_A\}_{\vec{K_B}}$ in the first step, since nobody but Alice could decrypt Bob's message and learn the value of $N_B$. A related goal of the protocol is to preserve the secrecy of nonces $N_A$ and $N_B$.

Gavin Lowe [Low96] discovered that the protocol fails to achieve secrecy and authentication due to the following (by now very well-known) attack:

1.1. $A \rightarrow I$ : $\{A, N_A\}_{\vec{K_I}}$
The attacker, acting as a legitimate participant in the protocol, is contacted by Alice

1.2. $I \rightarrow B$ : $\{A, N_A\}_{\vec{K_B}}$
The attacker starts a new session of the protocol with Bob, pretending to be Alice

2.2. $B \rightarrow (A)$ : $\{N_A, N_B\}_{\vec{K_A}}$
Bob replies to message 1.2 according to the protocol specification (Bob thinks that message 1.2 came from Alice). Message 2.2 is intercepted by the attacker

2.1. $I \rightarrow A$ : $\{N_A, N_B\}_{\vec{K_A}}$
The attacker replies to message 1.1 using the intercepted message 2.2. At this point Alice, who only observed messages 1.1 and 2.1, believes that $N_B$ has been generated by $I$.

3.1 $A \rightarrow I$ : $\{N_B\}_{\vec{K_I}}$
Alice replies to $I$'s message 2.1 according to the protocol specification

3.2 $I \rightarrow B$ : $\{N_B\}_{\vec{K_B}}$
The attacker, again impersonating Alice, sends the expected answer to message 2.2

The authentication goals fail as follows:

- Upon reception of message 2.1, Alice should be confident that the message has been constructed by $I$, which is not the case.
- Upon reception of message 3.2, Bob should be confident that $A$ sent the message $\{A,N_A\}^{\rightarrow}_{K_B}$, which is not the case.

In other words, Bob thinks he is talking with Alice, while he is talking with the attacker.
The secrecy goal fails as follows: $N_B$ should be a secret shared by Alice and Bob only, while message 3.1 allows the attacker to learn it.

# 3. Attacker model

It is typically assumed that the set of principals consists of two disjoint sets: the honest principals and the attackers. The attackers may include dishonest protocol participants. For most protocols, it is sufficient to analyze the security of the protocol against a single attacker that combines the knowledge and abilities of all dishonest principals.
The honest participants are assumed to follow the rules of the protocol as defined in the protocol specification in a mechanistic way. What they do when they receive a message which does not match their expectation is left unspecified. It is assumed that they do not keep track of previously completed sessions and, more generally, that they do not play an active role in detecting or tracing possible attacks.

### *3.1. Dolev-Yao model*

A common attacker model used in formal analysis of security protocols is the so called *Dolev-Yao model*, inspired by [DY83]. Following the convention, we used the term *Dolev-Yao* somewhat loosely. Some of the attacker models described below are in fact richer than the original Dolev-Yao model.
We assume that the attacker can eavesdrop on, remove, and arbitrarily schedule messages sent on public communication channels. It can also create new messages from the pieces of messages it already observed and insert them into the channels. The attacker can split unencrypted messages into pieces and decrypt encrypted terms if it knows the correct decryption key. It is assumed that messages contain enough redundancy so that the recipient can always determine if decryption was successful (*e.g.*, when the attacker decrypts an encrypted nonce $\{N\}^{\leftrightarrow}_{K}$ with a key $K'$, he can always tell whether $K = K'$). In the Dolev-Yao model, the attacker has the choice to intercept any message transmitted on a public communication channel and possibly replace it with a message constructed from his a priori knowledge and parts of the messages previously sent by any participant in this or other session of the protocol.
The steps taken by honest participants following the protocol specification and (non-deterministic) actions of the attacker give rise to an abstract model of the protocol as a state transition system (*e.g.*, [MP95]). The general approach taken in formal analysis of security protocols is to analyze all feasible traces of the state transition system and determine for each trace whether all of the desired security properties are preserved. This task is complicated by the following considerations:

- There can be arbitrarily many sessions (also known as *instances*) of the protocol which can be interleaved in an arbitrary way.
- One agent can participate in arbitrarily many sessions at the same time. The memory of each agent is, therefore, unbounded (as has been mentioned, an agent's memory is limited to uncompleted sessions).
- The attacker can generate an unbounded number of messages.
- Nonces have limited scope: honest principals forget nonces as soon as the corresponding instance of the protocol has completed.

Among the formal models for protocol traces, the most widely used are CSP [H85, Ros94, Ros95, Low96, Sch96], higher-order logic [Pau98], multiset rewriting [CDL+99, CDL+00], and strand spaces [THG99]. For information about the relation between different models, see [CDL+00].

### *3.2. Spi-calculus*

In the spi-calculus [AG99] the behavior of honest protocol participants is formalized as a process in a special-purpose process calculus (basically, an extension of $\pi$-calculus [Mil92] with cryptographic operations). This process can be replicated any number of times to model several instances of the protocol running concurrently. The attacker can observe and participate in any communication in any possible way. The model, however, also relies on the perfect cryptography assumption.
Protocol security can then be expressed as *observational equivalence* of two systems. In the first system, an arbitrary process $A$ (which models the public network controlled by the attacker) is run concurrently with the process modeling the actual protocol. In the second system, $A$ is run concurrently with a process modeling an idealized specification of the protocol which is secure by design. If the two systems are observationally equivalent in the process-calculus sense (taking into account cryptographic operations, *e.g.*, $\{N\}^{\rightarrow}_{K}$ and $\{N'\}^{\rightarrow}_{K}$ may not be distinguishable by the attacker who does not know $K$), then the protocol is secure.
For example, secrecy can be modeled by considering an attacker $A$ that outputs a message on a designated channel when it learns the secret. When run concurrently with the ideal version of the protocol, $A$ cannot possibly learn the secret and thus never outputs on the channel. If $A$ cannot learn the secret from the actual protocol, it will not be able to output on the channel when run concurrently with the protocol, and the two systems will be observationally equivalent, *i.e.*

$$P[\text{secret}] \mid A \approx_{obs} P[\text{any}] \mid A.$$

In addition to the notions of security supported by the Dolev-Yao model, spi-calculus can be used to analyze implicit flows since the attacker may perform comparisons between observed messages and produce output depending on the comparison results. For instance, the processes may test encrypted (unknown) values for equality and perfom actions depending on the result of the test.

### *3.3. Probabilistic models*

Recently, attempts have been made to develop analysis techniques for more realistic formal models of cryptography that go beyond the Dolev-Yao abstraction described in section 3.1.. The goal is to replace "black-box" abstractions of cryptographic primitives with probabilistic models. These models include probabilistic polynomial-time process calculus [LMMS98, LMMS99] and more traditional (in the cryptographic sense) simulatability-based models [PW00a, PW00b, PW01, Can00]. No tools have been developed so far for the mechanized analysis of realistic formal security models.

## 4. General techniques

In general, there is no algorithm which takes a cryptographic protocol as input and always outputs either "yes, the protocol is secure," or "the protocol is insecure and here is an attack." Both secrecy and authentication are undecidable in the Dolev-Yao model, and so are probably all the interesting properties one might want to check [DLMS99, AL00, CCM01]. We give more details on the sources of undecidability below.

Despite this limitation, there exist semi-decision techniques which can be automated in various ways. First, observe that it is possible to design an algorithm which will always find an attack (by the Dolev-Yao attacker) in finite time if an attack exists and may not terminate if the protocol is correct. This can be done by simply enumerating all traces of the protocol's state transition system. Then, in each state, it can be decided if secrecy has been violated, as explained in section 6.1..

Other semi-decision techniques and tools include, but are not limited to, Paulson's inductive method [Pau98], NRL Protocol Analyzer [Mea96], Athena [Son99], and abstraction-based techniques by Bolignano [Bol97, Bol98] (this is by no means a comprehensive list).

There are several sources of undecidability. First, the protocol itself can simulate one step of computation for a universal computation model (*e.g.*, a Turing machine): each state of the machine is an agent who, upon reception of a configuration, sends the next configuration to the appropriate state. The attacker only has to bridge two successive sessions forwarding the last message of one session to the appropriate principal as the first message of the next session. That is why decision methods have to either impose a bound on the number of instances as in [AL00], or restrict manipulation of the messages (*e.g.*, impose a "single reference to previous messages" restriction [CCM01]).

The second source of undecidability is the ability to generate nonces, which may be used, roughly, to simulate arbitrarily many memory locations and therefore encode machines with unbounded memory [DLMS99]. Again, if the number of protocol instances is bounded in advance, this cannot occur. In fact, it is sufficient to bound the total number of nonces which are generated in any trace.

Even if it is assumed that there is a bounded number of instances, it is not yet easy to design a decision algorithm since, according to the Dolev-Yao model, the attacker still has an unbounded number of possible choices at any point. In particular, the number of messages that can be created by the attacker is unbounded. An additional restriction bounding the attacker's memory allows development of finite-state decision techniques.

## 5. Finite-state analysis

Bounding the number of instances and the number of times each cryptographic operation can be applied by the attacker yields finite-state analysis, which terminates. In this case the protocol can be described by a finite state machine and reachability properties such as secrecy and authentication can be expressed formally, *e.g.*, in some temporal logic. This enables the use of finite-state *model checking* tools such as FDR [Ros95, Low96], Mur$\varphi$ [MMS97], and Brutus [CJM].

Lowe [Low99] gave a syntactic characterization of a class of protocols such that, for every insecure protocol in the class, there is an attack using a bounded number of sessions and a bounded number of applications of cryptographic primitives (therefore, there is a bound on the number of attacker operations and on the size of terms that the attacker may have to construct). For this class, both the attacker's memory and the number of sessions can be bounded without sacrificing completeness. This enables application of model checking. Moreover, the bounds are quite small in practice.

This result can be seen as a decidability result for the class of protocols which satisfy the assumptions in Lowe's paper [Low99]. Many of these assumptions are among prudent engineering practices for security protocols proposed by Abadi and Needham [AN96], but it is not realistic to assume that they are satisfied by a particular cryptographic protocol. Following are some of the requirements defining the class:

- The intended recipient of a message should be able to decompose the message into atomic pieces. This means, for example, that he cannot use part of the message as a black box to be included in the reply, as done, *e.g.*, in Kerberos [Kerb].

- Every message must contain (under encryption) the name of the supposed sender.

- Message fields must contain enough redundancy so that it is always possible to determine the type of the field. Type confusion between keys, names, nonces, etc. should not be possible.

- There are no temporary secrets. The protocol should be secure under the assumption that everything which is sent in the clear is part of the attacker's initial knowledge.

Among Lowe's requirements is also the restriction of protocols to atomic encryption keys which are either nonces, or basic constants. It should not be possible to build new keys out of existing ones. This assumption, however, is too restrictive for the modeling of "real-world" key exchange protocols such as SSL 3.0 [SSL] where it is typical for the parties to compute symmetric keys as functions of the shared secret material. In fact, reachability is decidable even in the presence of constructed keys assuming the number of protocol instances is bounded (see section 6.3.).
Stoller [Sto00] demonstrated a more general class $\mathscr{C}$ of protocols for which it is possible to derive, from the protocol specification, a theoretical upper bound on the number of cryptographic function applications that have to be made by the attacker. Stoller also gives a decision algorithm for membership in $\mathscr{C}$. The algorithm is complicated due to the lack of syntactic characterization of the protocol class.

# 6. Decision results for infinite-state analysis

The protocol analysis techniques surveyed in this section assume that there is a bounded number of protocol sessions, but attacker computations are unbounded. In particular, there are no limits on the depth of terms that can be constructed by the attacker. In all of the techniques, the subject of the analysis is an idealized Dolev-Yao model of the honest protocol participants running in parallel with an attacker who controls the public communication channels. This models execution of the protocol in a hostile environment (we will thus use terms "attacker" and "environment" interchangeably). Therefore, every input to the honest processes from the environment can be viewed as constructed by the attacker.
Typically, specifications of protocol participants' roles contain variables. Variables represent data that the participant does not possess prior to starting the protocol and receives from the environment as part of the protocol. For example, after initiating a key exchange with Bob, Alice may receive a term encrypted with her public key. Since Alice does not know the value of the term before receiving it, it will be denoted by a variable in the specification of Alice's role in the protocol.
For instance, in the Needham-Schroeder example from section 2.5., Bob (*i.e.*, any agent playing the role of Bob), upon reception of message $\{X,Y\}_{\vec{K_B}}$ will send back $\{N_B,X\}_{\vec{K_Y}}$. Here $X,Y$ are variables since, from Bob's viewpoint, they originated from the environment and their values are not known to Bob apriori. $X$ ranges over arbitrary data and $Y$ ranges over principal names (under the assumption that the agents are able to distinguish principal names from other data). In such a formulation, $X$ could be, for instance, a name or a key or a nonce. Some formalisms assume that each piece of data comes annotated with its type, preventing type confusion attacks [CJ97]. In any case, Bob cannot check that $X$ has been generated by the agent whose name is $Y$.

### *6.1. Symbolic protocol models*

All of the analysis techniques considered in this section have two main components:

**Symbolic reduction** The basic idea behind symbolic reduction is to avoid instantiating variables in the protocol specification unless necessary. This is done by defining a symbolic state transition relation which gives rise to the (finite) symbolic state space of the honest protocol participants. Each symbolic state summarizes an infinite number of concrete states that can be obtained by instantiating variables in the symbolic state specification. Protocol correctness conditions are represented by constraints. A typical constraint is the requirement that every input term received by the honest participants from the environment must be derivable from the environment's initial knowledge combined with the terms sent by the participants on public channels up to that point.

**Knowledge analysis** Each technique defines a deduction system for determining whether a particular term can be derived from a given set of terms. Obviously, the deduction system depends on the chosen attacker model. In the Dolev-Yao model, even though the set of terms that can be constructed by the attacker from a given finite initial knowledge is infinite, it is possible to effectively compute a finite tree automaton which accepts this set of terms. This is also true if the initial knowledge of the attacker is a regular set of terms [Mon99, Gou00, AL00]. We will use notation $\mathscr{F}(T)$ for the infinite set of terms that can be derived by the attacker from a particular set $T$ of ground terms.

The protocol analysis problem is then reduced to deciding whether the attacker can instantiate a protocol trace that violates one of the protocol correctness conditions, *i.e.*, if there exists an instantiation of variables computable by the attacker that satisfies the constraints implied by the faulty trace.

### *6.2. Constructed vs. atomic keys*

In the simplest Dolev-Yao-style model for symmetric encryption, it is assumed that all symmetric keys are atomic - either constants, or variables that can be instantiated only to constants. This simplifies knowledge analysis, since the set of terms $\mathscr{F}(T)$ that can be derived by the Dolev-Yao attacker from a given term set $T$ is equal to $\mathsf{synth}(\mathsf{analz}(T))$

where synth and analz are Paulson's synthesis and analysis closures of term sets. Roughly, $\mathsf{analz}(T)$ is the set of all terms that can be obtained by breaking up and decrypting terms in $T$, and $\mathsf{synth}(T)$ is all terms that can be obtained by combining, encrypting, and hashing terms in $T$. With atomic keys, analysis of a term is linear in the depth of the term's structure.

To analyze "real-world" protocols, it is often necessary to extend the model with constructed symmetric keys. In a typical key exchange scenario, two parties exchange a secret, then each derives the shared symmetric key by hashing parts of the shared secret together with nonces and other data. An example of this is master key computation in the SSL 3.0 handshake protocol [SSL].

### *6.3. Symbolic decision techniques*

In this section, we describe several symbolic decision techniques for security protocols and the assumptions they make about protocols. Unless stated otherwise, all of the methods assume a bounded number of protocol instances but impose no bounds on the attacker's knowledge set $\mathscr{F}$. All results described below hold for the scenarios in which a principal is involved in several parallel sessions. Though only [Gou00] explicitly considers an infinite initial knowledge of the attacker, most of the results described below also apply in this case.

**Huima.** The origin of symbolic protocol analysis can be traced to the seminal work of Dolev and Yao [DY83] which applied to a very restricted class of protocols. Huima's paper [Hui99] was the first to present a decision technique for secrecy in cryptographic protocols without a bound on attacker operations. The class of protocols considered in [Hui99] is very general. Protocols are defined using an adhoc process algebra formalism, somewhat similar to untyped spi-calculus. Both symmetric and public-key encryption are supported, and constructed keys are allowed.

A standard term rewrite system is defined, representing the attacker's ability to manipulate terms by splitting, decrypting with a known key, encrypting, etc. User-defined symbols are given "semantics" by instantiating one of the pre-defined relation templates. For example, after declaring symbols $e$ and $d$, the user can declare $P_{\mathrm{symenc}}(e,d)$, meaning that for any terms $t_1$ and $t_2$, $d(t_1, e(t_1,t_2)) \to t_2$. While the templates support explicit decryption operators (and, therefore, a limited equational theory associated with the term algebra), there is no support for commutative and associative operators. See also [Mon99].

For each protocol participant, its local state is defined as $\langle p, Y, c\rangle$ where $p$ is the process representing the corresponding protocol role, $Y$ is a partial variable instantiation function from variable names to terms, and $c$ is a counter used to keep track of fresh values. A symbolic state of the entire protocol is defined as a triple $\langle \mathscr{L}, \mathscr{M}, \mathscr{C}\rangle$ where $\mathscr{L}$ is a function from participant names to their local states, $\mathscr{M}$ is the set of terms known to the environment (attacker), and $\mathscr{C}$ is a list of constraints that must be satisfiable in order for the state to be reachable. Each constraint has one of the following forms: $Eq(t,t')$, $Ineq(t,t')$, or $InClos(t,M)$ where terms $t,t'$ and term set $M$ may involve variable names. An *InClos* constraint represents the requirement that ground instances of term $t$ must be derivable, using the rewrite system, from the ground instances of terms in $M$. Such a constraint is satisfiable *iff* there exists a substitution $\sigma$ such that $\sigma t \in \mathscr{F}(\sigma M)$. (We write $\sigma t$ for the term $t$ in which all variables are replaced according to $\sigma$.)

Symbolic reduction is handled by defining a transition relation for symbolic global states that generates a finite symbolic state space with associated constraints (*e.g.*, if a participant receives term $x$, then $InClos(x,M)$ is added to the constraint list, because the state can only be reached if the environment is capable of generating $x$). Protocol correctness conditions are also formulated as constraints (*e.g.*, secrecy of term $t$ can be expressed as $\neg InClos(t,M)$), and the two constraint lists are merged. Finally, each terminal symbolic state is transformed in a certain way in order to decide whether there exists a instantiation of variables that satisfies all constraints simultaneously. Note that deciding the existence of an instantiation that satisfies an $InClos(t,M)$ constraint requires deciding the knowledge analysis problem as explained in section 6.1..

The paper contains no details of the algorithm used to decide the constraint satisfaction problem apart from the list of high-level properties that are supposed to guarantee termination, and the claim that the method is sound and complete.

**Amadio-Lugiez-Vanackère.** Amadio *et al.* [AL00, ALV01] use a untyped process algebra formalism similar to the spi-calculus [AG99] for specifying protocols. Only symmetric-key encryption with atomic keys is considered. Variables in key positions are handled by brute-force enumeration of all possible substitutions. The decision algorithm is proved NP-hard.

In this approach, symbolic reduction is combined with knowledge analysis. A symbolic state of the protocol is a triple $(P,T,E)$ where $P$ is the state of the process representing the honest participants, $T$ is the finite set of terms representing the attacker's knowledge, and $E$ is an ordered list of constraints $x_1 : T_1, \ldots, x_n : T_n$ such that $T_1 \subseteq \ldots \subseteq T_N$. Each $x_i : T_i$ constraint corresponds to a point in the protocol execution where the accumulated knowledge of the environment consists of terms in set $T_i$. The values of $x_i$ are the terms which are sent to the honest protocol participants in a trace. Such constraints are equivalent to Huima's $InClos(x_i, T_i)$ constraints and are satisfiable *iff* there exists a substitution $\sigma$ such that $\sigma x_i \in \mathsf{synth}(\mathsf{analz}(\sigma T_i))$, *i.e.*, if, after $\sigma$ instantiates all free variables, $x_i$ is derivable from $T_i$ using operations available to the Dolev-Yao attacker (see section 3.1.). It is worth noting that the characterization of $\mathscr{F}(\sigma T_i)$ as $\mathsf{synth}(\mathsf{analz}(\sigma T_i))$ is only valid if decryption keys are atomic. For each symbolic reduction step, the algorithm checks if the substitution required for the step is compati-

ble with previous substitutions. The algorithm thus decides whether there exists a single substitution that solves all $x_i : T_i$ constraints simultaneously.
As in Huima's approach, to account for the conditional it is necessary to accumulate a separate set of equality constraints as symbolic reduction progresses. Equality constraints are solved by a separate set of rules.

**Boreale.** Boreale [Bor01] also formalizes abstract models of protocols in a variant of spi-calculus, considering only symmetric-key encryption. Only variables or atomic terms may appear in key positions. The original paper [Bor01] only deals with authentication properties, but the general method can also be used to analyze any reachability property, including secrecy. There is a publicly available analysis tool (STAP), which also handles constructed keys.
The knowledge analysis problem for ground terms is decided using a standard Dolev-Yao deduction system. Protocol execution by honest participants is modeled by a symbolic transition relation that allows messages to contain free variables. As in other methods, exhaustive enumeration of all symbolic traces produces a finite symbolic state space of the protocol.
Protocol analysis is then equivalent to deciding, for each symbolic trace, if it can be solved, *i.e.*, if there exists an instantiation of variables in messages such that in the resulting concrete trace every ground term sent by the environment is derivable from the environment's knowledge at that point using the deduction system. This is the same as deciding the satisfiability of Huima's *InClos* constraints for a particular symbolic trace.
The paper gives a *refinement* decision procedure that works by gradually instantiating variables in the symbolic trace until a *solved form* is obtained in which every sent term is derivable by the environment. The key idea is that any symbolic term can be decomposed into a finite number of irreducible components by splitting pairs and decrypting if the correct key is known to the environment. Therefore, for each message sent by the environment, it is possible to (i) split the sent term into its irreducible components, (ii) split all symbolic terms known to the environment at that point into their components. Since both sets are finite, the symbolic knowledge analysis problem can be decided by checking that the component set of the term is included, modulo unification, in the component set known to the environment.
The refinement process is non-deterministic and may lead to several different solved forms for the same symbolic trace. Completeness is proved by demonstrating that every solution of the symbolic trace is a solution of at least one of the solved forms produced by the algorithm.

**Fiore-Abadi.** Fiore and Abadi [FA01] are similar to Amadio *et al.* and Boreale in that they use a variant of untyped spi-calculus with symmetric-key encryption and decryption and a free term algebra. The analysis method supports constructed keys, but completeness is proved only for atomic keys.
The method creates a symbolic computation graph of the honest protocol processes. Paths in the graph represent all possible execution traces of the protocol, and some of them may violate the desired security properties. To determine if there exists a concrete execution trace of the protocol corresponding to the violating path, the paper gives an algorithm for deciding the existence of *realisers* for all symbolic inputs (*i.e.*, message sends) to the process from the environment. A realiser is a substitution for variables such that every resulting ground input term can be derived by the environment from the terms it already knows at that point. Once again, this is equivalent to deciding the satisfiability of all $InClos(t_i, M_i)$ constraints, or finding a substitution $\sigma$ such that $\sigma t_i \in \mathscr{F}(\sigma M_i)$ for all terms $t_i$ sent by the environment at a point where it knows $M_i$.

**Rusinowitch-Turuani.** Rusinowitch and Turuani [RT01] extend the work by Amadio *et al.* [AL00] in two directions. First, their model supports public keys as well as constructed symmetric keys. Second, they show that the symbolic knowledge analysis problem is NP-complete for the Dolev-Yao attacker as long as the number of sessions is bounded.
The main result of the paper is a polynomial bound on the number of attacker operations that may be needed in order to construct the substitution that realizes the attack. If $t \in \mathscr{F}(M)$, *i.e.*, if the term that must be sent by the environment can be derived from the term set representing the environment's knowledge, then there exists a *normal* derivation of $t$ from $M$ that has a polynomial size. This is similar in spirit to Lowe's "small system" result [Low99], but with significantly fewer restrictions on the protocol.
The polynomial bound on normal derivations is then used to construct an NP-complete procedure for deciding the protocol insecurity problem. The procedure works by guessing a ground substitution $\sigma$ for all variables such that the size of the $\sigma x$ term has a polynomial upper bound, then guessing a polynomial sequence of attacker operations $l_1, \ldots, l_N$, and finally checking that $\sigma t \in l_N(\ldots l_1(\sigma M))$. Such a procedure is obviously impractical for real protocol analysis, but in addition to establishing complexity of the problem, the existence of polynomial normal attacks supports the empirical observation that all Dolev-Yao attacks on cryptographic protocols that have been discovered so far are relatively simple.

**Comon-Cortier-Mitchell.** Comon *et al.* [CCM01] consider the Dolev-Yao protocol model with support for public keys and constructed symmetric keys. There are two main assumptions. The first one slightly relaxes the finite-sessions requirement by assuming that only a bounded amount of fresh data is generated in all sessions. This means that either there is a finite number of sessions, or else the protocol does not contain any nonce generation steps. This restriction alone is not sufficient for decidability; it is still possible to build a protocol simulating one transition step of a universal computation model. The second restriction states, roughly, that an agent can copy only one piece of any message he

receives into any message he sends. This rules out, for instance, simulation of two-stack machines.
The decision technique is based on a reduction to set constraints (*e.g.*[CP97]), which in turn are reduced to an automata-theoretic question. The resulting algorithm runs in doubly exponential time.

**Millen-Shmatikov.** Millen and Shmatikov [MS01] present a decision technique for reachability properties based on constraint solving. Each honest protocol participant is specified as a *semi-bundle* in the strand space model [THG99]. A semi-bundle is a strand (*i.e.*, a protocol role) parameterized with variables. The Prolog implementation automatically generates all possible interleavings of the semi-bundles.
Using parameterized strands to represent symbolic traces of the protocol achieves a clean separation between the symbolic reduction problem and the knowledge analysis problem. As in other approaches, deciding the latter is equivalent to solving a system of constraints of the form $t_i : T_i$, where $t_i$ is a term, possibly containing variables, sent by the attacker to the honest processes, and $T_i$ is the set of terms available to the attacker. These constraints are equivalent to Huima's *InClos* constraints, and are satisfiable if $\exists\sigma$ such that $\forall i\ \sigma t_i \in \mathscr{F}(\sigma T_i)$, *i.e.*, every term needed to the stage an attack can be generated by the attacker.
The resulting constraint system is solved by applying a set of constraint reduction rules. The constraint solving procedure is terminating, sound, and complete even in the presence of constructed keys. Unlike the Rusinowitch-Turuani procedure [RT01], the algorithm is useful in practice and can be applied to the analysis of real protocols.

## 7. Conclusion

Protocol analysis is a model checking problem [LSV01]: given a model (the protocol) and a property, we want to decide whether the model satisfies the property. As we have seen, however, the model is an infinite state system, and classical model checking techniques can only be used to verify an approximate model. Nevertheless, as with infinite-state model checking techniques, symbolic representation of infinite sets of states (*e.g.*, using constraints) and reasoning about such representations may yield interesting decision results, some of which have been sketched above.
There are still a number of open questions, and more generally, several open areas of research. Let us mention two of them as a conclusion:

- We considered only two particular security properties: secrecy and authentication. While the described techniques may work for other trace properties, there are several security goals which are not trace properties (for instance, anonymity and fairness). There is currently no specification language (such as temporal logic for reactive systems), which is rich enough to express all desired security properties. Design of decision algorithms for such properties is an open problem.
- We assumed that terms and messages are generated by a free algebra. As mentioned above, this is an approximation since most cryptographic primitives satisfy some algebraic properties. Which properties can be supported by the model while preserving decidability is an open question.

## Acknowledgements

The authors are grateful to Jean Goubault-Larrecq and Alexandre Boisseau for their comments on an early version of this paper.
This work was partly supported DoD MURI "Semantic Consistency in Information Exchange," ONR Grant N00014-97-1-0505, and NSF CCR-9629754.

## References

[Aba99] M. Abadi. Security protocols and their properties. In *Foundations of Secure Computation* (F.L. Bauer and R. Steinbrueggen, eds.), NATO Science Series, IOS Press (2000), pages 39–60. Volume for the 20th International Summer School on Foundations of Secure Computation, held in Marktoberdorf, Germany (1999).

[AG99] M. Abadi and A. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[AN96] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996.

[AL00] R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *Proc. CONCUR*, vol. 1877 of *Lecture Notes in Computer Science*, pages 380–394, 2000.

[ALV01] R. Amadio, D. Lugiez and V. Vanackère. On the symbolic reduction of processes with cryptographic functions. Technical Report 4147, INRIA, March 2001.

[Bol97] D. Bolignano. Towards a mechanization of cryptographic protocol verification. In *Proc. 9th International Conference on Computer Aided Verification (CAV)*, pages 131–142, 1997.

[Bol98] D. Bolignano. Integrating proof-based and model-checking techniques for the formal verification of cryptographic protocols. In *Proc. 10th International Conference on Computer Aided Verification (CAV)*, pages 77–87, 1998.

[Bor01] M. Boreale. Symbolic Trace Analysis of Cryptographic Protocols In *Proc. 28th International Conference on Automata, Languages and Programming (ICALP)*, vol. 2076 of *Lecture Notes in Computer Science*, Springer-Verlag, pages 667–681, 2001.

[Can00] R. Canetti. A unified framework for analyzing security of protocols. IACR Cryptology ePrint Archive 2000/067, December 2000, `http://eprint.iacr.org`

[CDL+99] I. Cervesato, N. Durgin, P.D. Lincoln, J.C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *Proc. 12th IEEE Computer Security Foundations Workshop*, pages 55–69, 1999.

[CDL+00] I. Cervesato, N. Durgin, P.D. Lincoln, J.C. Mitchell, and A. Scedrov. Relating strands and multiset rewriting for security protocol analysis. In *Proc. 13th IEEE Computer Security Foundations Workshop*, pages 35–51, 2000.

[CJ97] J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. Draft paper, 1997. Available at `//www-users.cs.york.ac.uk/~jac`

[CJM] E.M. Clarke, S. Jha and W. Marrero. Verifying security protocols with Brutus. To appear in *ACM Transactions in Software Engineering Methodology.*

[CCM01] H. Comon, V. Cortier, and J.C. Mitchell. Tree automata with memory, set constraints and ping pong protocols. In *Proc. 28th International Conference on Automata, Languages and Programming (ICALP)*, vol. 2076 of *Lecture Notes in Computer Science*, Springer-Verlag, pages 682–693, 2001.

[CP97] W. Charatonik and A. Podelski. Set constraints with intersection. In *Proc. IEEE Symposium on Logic in Computer Science*, Varsaw, 1997.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DLMS99] N. Durgin, P.D. Lincoln, J.C. Mitchell and A. Scedrov. Undecidability of bounded security protocols. In *Proc. FLOC Workshop on Formal Methods in Security Protocols*, Trento, Italy, 1999.

[DY83] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[FA01] M. Fiore and M. Abadi. Computing symbolic models for verifying cryptographic protocols. In *Proc. 14th IEEE Computer Security Foundations Workshop*, pages 160–173, 2001.

[Gou00] J. Goubault-Larrecq. A method for automatic cryptographic protocol verification. In Dominique Méry, Beverly Sanders, editors, *Fifth International Workshop on Formal Methods for Parallel Programming: Theory and Applications (FMPPTA 2000)*, vol. 1800 of *Lecture Notes in Computer Science*, Springer-Verlag, 2000, pages 977–984.

[HAC] A.J. Menezes, P.C. van Oorshot and S.A. Vanstone. Handbook of Applied Cryptography. CRC Press, 1997.

[H85] C.A.R. Hoare. Communicating Sequential Processes. Prentice-Hall, 1985.

[HRU76] M. Harrison, W.L. Ruzzo and J. Ullman Protection in operating systems Communications of the ACM, pages 461–471, Aug 1976.

[Hui99] A. Huima. Efficient infinite-state analysis of security protocols. In *Proc. FLOC Workshop on Formal Methods in Security Protocols*, Trento, Italy, 1999.

[Kerb] J.T. Kohl and B.C. Neuman. The Kerberos network authentication service (version 5). Internet Request For Comments RFC-1510, September 1993.

[Lam74] B. Lampson. Protection. Proc. 5th Princeton Conf. on Information Sciences and Systems, Princeton, 1971. Reprinted in ACM Operating Systems Rev. 8, 1 (Jan. 1974), pp 18-24.

[LMMS98] P.D. Lincoln, J.C. Mitchell, M. Mitchell and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proc. 5th ACM Conference on Computer and Communications Security*, pages 112–121, 1998.

[LMMS99] P.D. Lincoln, J.C. Mitchell, M. Mitchell and A. Scedrov. Probabilistic polynomial-time equivalence and security analysis. In *Proc. Formal Methods '99*, vol. 1708 of *Lecture Notes in Computer Science*, Springer-Verlag, pages 776–793, 1999.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS)* (Margaria and Steffen, eds.), vol. 1055 of *Lecture Notes in Computer Science*, Springer-Verlag, pages 147–166, 1996.

[Low97] G. Lowe. A hierarchy of authentication specifications. In *Proc. 10th IEEE Computer Security Foundations Workshop*, pages 31–43, 1997.

[Low99] G. Lowe. Towards a completeness result for model checking of security protocols. *Journal of Computer Security*, 7(2–3):89–146, 1999.

[LSV01] B. Bérard, M. Bidoit, F. Laroussinie, A. Petit, L. Petrucci, P. Schnoebelen. Systems and Software Verification. Model-Checking Techniques and Tools. *Springer Verlag*, 2001.

[Mea96] C. Meadows. The NRL protocol analyzer: an overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[MP95] Z. Manna and A. Pnueli Temporal Verification of reactive systems Springer Verlag, 1995.

[MS01] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. To appear in *Proc. 8th ACM Conference on Computer and Communications Security*, 2001.

[Mil92] R. Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2:119–141, 1992.

[MMS97] J.C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murphi. In *Proc. IEEE Symposium on Security and Privacy*, pages 141–153, 1997.

[Mon99] D. Monniaux. Abstracting cryptographic protocols with tree automata. In *Proc. 6th International Static Analysis Symposium (SAS'99)*, vol. 1694 of *Lecture Notes in Computer Science*, Springer-Verlag, 1999.

[NIST94] Announcement of Weakness in the Secure Hash Standard. *National Institute of Standards and Technology (NIST)*, 1994.

[NS78] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–9, 1978.

[Pau98] L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[PW00a] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. 7th ACM Conference on Computer and Communications Security*, pages 245–254, 2000.

[PW00b] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. IBM Research Report RZ 3304, December 2000.

[PW01] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. IEEE Symposium on Security and Privacy*, pages 184–200, 2001.

[R92] R.L. Rivest. RFC 1321: The MD5 Message-Digest Algorithm. Internet Engineering Task Force, 1992.

[RSA78] R.L. Rivest, A. Shamir and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[Ros94] A.W. Roscoe. Model-checking CSP. In *A Classical Mind: Essays in Honor of C.A.R. Hoare*, Prentice-Hall, 1994.

[Ros95] A.W. Roscoe. Modeling and verifying key-exchange protocols using CSP and FDR. In *Proc. 8th IEEE Computer Security Foundations Workshop*, pages 98–107, 1995.

[RT01] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proc. 14th IEEE Computer Security Foundations Workshop*, pages 174–190, 2001.

[Sch96] S. Schneider. Security properties and CSP. In *Proc. IEEE Symposium on Security and Privacy*, pages 174–187, 1996.

[Son99] D. Song. Athena: A new efficient automatic checker for security protocol analysis. In *Proc. 12th Computer Security Foundations Workshop*, pages 192–202, 1999.

[SSL] A. Freier, P. Karlton and P. Kocher. The SSL protocol. Version 3.0. `http://home.netscape.com/eng/ssl3/`

[Sto00] S.D. Stoller. A bound on attacks on payment protocols. To appear in *Proc. IEEE Logic in Computer Science*, 2001. Available as Technical Report 537, Computer Science Dept., Indiana University, February 2000.

[THG99] F.J. Thayer Fabrega, J.C. Herzog, and J.D. Guttman. Strand Spaces: Proving security protocol correct. *Journal of Computer Security*, 7:191–230, 1999.