# Separating the Specification and Implementation Phases in Cryptology

(Extended Abstract)

Marie-Jeanne Toussaint[1]

**Abstract**

We propose to separate the specification and implementation phases in the conception of the cryptographic protocols. The specification phase describes the logic of the protocol. We develop a method for formally verifying this logic before the publication of the protocol. The implementation phase contains the choice of an appropriate cryptographic function.

**Key words** : cryptology, security, cryptographic protocol, formal verification, knowledge state, protocol execution tree.

## 1 Introduction

The security of the cryptographic protocols depends not only on the cryptosystem being used but also on the way these cryptosystems are used i.e. on the logic of the protocols. Some methods for formally verifying the logic of the protocols exist (see for examples [BAN89, Bie89, Tou91b, Tou91a]). But each of these methods requires that the analyzed protocol is translated in an appropriate syntax. Many of these methods do not worry about the particular cryptosystem being used: the protocol to be studied by the method has to be described in a relatively general way without specifying the particular cryptosystem to be used. These methods only verify the protocol logic and do not detect the weaknesses of the cryptosystems.

Many cryptographic protocols are described without specifying any cryptosystem. The methods of formal verification can thus be applied to them without any problem. But some cryptographic protocols (many recent ones) do not separate the protocol logic and the cryptosystem being used. This cryptosystem is specified in the description of the protocol. We say that these protocols are '*specific*'. Before being analyzed by the methods of formal verification, these protocols have to be generalized to be used with any

[1] E-mail : toussain@montefiore.ulg.ac.be
Address : 6, rue Lambert Delava
B-4530 Vaux et Borset; BELGIUM.
This research was made when the author was Research Assistant for the National Fund for Scientific Research (Belgium) and "S.P.P.S." Researcher at the University of Liège.
She is now working in the Scientific Computer Science Group of the Research Centre of Solvay (Brussel).

Y. Deswarte et al. (eds.), *Computer Security - ESORICS 92*

cryptosystem (or with any cryptosystem satisfying some properties). This generalization could seem tedious but it enables to deeply understand the basic mechanisms of the protocols and sometimes to find some inaccuracies in their description. In [Tou91b], we have been able to generalize two different protocols described in [BLY88] into a unique protocol using any cryptosystem having some required properties. The basic ideas of these two protocols are identical but are hidden by the use of different cryptographic functions.

Nevertheless, this generalization is an additional step. I think it would be more efficient that, in the future, the protocol designers specify their protocols in a syntax appropriate to directly apply a formal verification method. The designers would obtain the proof of the security of their protocols before publishing them. Thus, the weaknesses of the protocols would be found before their publication which would only happen when all the weaknesses would be removed. That would avoid some inconveniences for firms which use cryptographic protocols, believing (without proof) that they are secure and are victims of frauds from users who have discovered some weaknesses in these protocols.

We propose to separate the protocol specification phase from the implementation phase. The specification phase consists in describing the protocol in the appropriate syntax and in very precisely specifying the properties that the cryptosystem to be used has to satisfy. The implementation phase consists in choosing cryptosystems (for example based on the problems of computing square roots of quadratic residues or of computing discrete logarithms...) which satisfy these conditions. The word 'cryptosystem' is taken here in a very large sense: it can be a real cryptosystem or a one-way function enough secure in practice. The 'efficiency' factor strongly influences the choice of cryptosystems.

We adopt the [Tou91b] method for formally verifying the security of cryptographic protocols: it is a generalization of the method we presented at Asiacrypt'91 [Tou91a]. We analyze the logic of the protocols by assuming that the cryptosystems are perfect (as defined in [Mer83] and [MW85]). The method is based on a representation of the complete knowledge that the participants are able to obtain in a protocol execution [Tou92]. The main advantages of this method are its generality (it can be applied to (almost) every cryptographic protocol) and its probabilistic aspect (we can prove the probabilistic properties of cryptographic protocols or estimate the probabilities that the detected attacks succeed).

The method consists in analyzing the knowledge states of the participants and their evolution during the execution of the protocol. With each possible state of the protocol, we associate the representation of the participants' knowledge in that state. At a given time of the execution of a protocol, a participant or an opponent wants to discover the

current protocol state. For that, he analyzes the possible protocol states and reject the states which are not compatible with his knowledge.

The paper is organized as follows. In Section 2, we briefly describe our representation of knowledge of the participants in a cryptographic protocol and our model of such a protocol. In Section 3, we define our syntax to precisely describe the specifications of a cryptographic protocol. Afterwards (Section 4), we briefly explain our method for formally verifying the security of cryptographic protocols and for detecting the possible attacks. In Section 5, we show the advantages of separating the specification and implementation phases by considering an example of 'specific protocol'.

### 1.1 A Simple Illustrative Example

We use a common example to introduce and illustrate the various concepts: it is a simple coin-flip protocol which is also used as an example in [MW85] and [Bie89]. This protocol implements a coin flip by phone. We assume that two users $A$(lice) and $B$(ob) want to flip a coin by telephone. Each user ($A$ and $B$) does not trust the other. We consider the following protocol.

1. $A$ chooses randomly a key $k$ in $\mathcal{K}$ and different messages $m_1, m_2$ in $\{T, H\}$. $A$ sends $em_1 = E(k, m_1)$ and $em_2 = E(k, m_2)$ to $B$. ($E(k, *)$ denotes the enciphering transformation under the key $k$ in a secret key cryptosystem).
2. $B$ picks $u$ in $\{em_1, em_2\}$ and sends $u$ to $A$.
3. $A$ sends $k$ to $B$ and they both know the answer $a$ of the coin flip by applying the transformation $D$ to $u$. ($D(k, *)$ denotes the deciphering transformation under the key $k$).

## 2 Modeling the States of Knowledge and the Protocol

### 2.1 Modeling the Cryptosystem

A cryptosystem is constituted by a set of (clear or enciphered) messages, a set of keys, and enciphering and deciphering transformations. A cryptosystem can thus been seen as an algebraic system. The elements of this algebra are the (clear or enciphered) messages

and the keys. Its operators are the enciphering and deciphering functions (denoted respectively by $E$ and $D$). The cryptosystem seen as an algebra is here called the *crypto-algebra* $\mathcal{C}$.

We want to model the notion of *perfect* cryptosystem. Like in [MW85], the cryptosystem is idealized by assuming that the crypto-algebra $\mathcal{C}$ is isomorphic to the free-algebra of the same type. This free-algebra is denoted by $\mathcal{F}$, the isomorphism between $\mathcal{F}$ and $\mathcal{C}$ by $\varphi$, and the operators of the free-algebra by $e$ and $d$ corresponding to the enciphering and deciphering operators of the crypto-algebra. In less formal terms, that amounts to assume that any stream of bits (corresponding to a clear or enciphered message) can be computed by only one way : it can thus be represented by only one expression in terms of the clear messages and the enciphering and deciphering operators. The cryptosystem being used has then no cycle and no collision.

Moreover, for some reasons that we will explain in Remark 2.1, we have also to assume that the set of the clear messages and the set of keys are infinite.

## 2.2 Modeling Participants' Knowledge

The cryptographic protocols are based on the secret of some information that some participants (a fortiori the opponents) are not allowed to know. In order to prove the security of these protocols, it is thus natural to analyze the knowledge of the participants and of the opponents in the different states of the protocol.

We assume that a state of knowledge about the cryptosystem is a subset of $\mathcal{F} \times \mathcal{C}$ which can be partitioned in three special finite sets $F$, $V$ and $SV$.

1. $F$ (for *F*ixed) is formed by pairs $(a, b)$ which define a one to one mapping from a subset of generators of $\mathcal{F}$ to a subset of generators of $\mathcal{C}$. That corresponds to generators of the crypto-algebra (i.e. clear messages or keys) that the participant has seen or that he knew at the start of the protocol and that he can label by a fixed element of the free-algebra: the participants thus knows the sequences of bits corresponding to these messages or keys and their meaning. The set $F$ will be described by specifying the free-algebra component of the pairs.

2. $V$ (for *V*ariables) is formed by pairs $(x, y)$ where $x$ is a fixed generator of the free-algebra whereas $y$ is a variable ranging over a subset of the generators of the crypto-algebra. The pairs of this type correspond to generators of the crypto-algebra that the participant has not seen but the existence of which he is aware of.

A pair $(x, y)$ of $V$ will be represented by a variable denoted by a tilded symbol, $\tilde{x}$, ranging over the image under the isomorphism $\varphi^{-1}$ of the domain of $y$.

3. $SV$ (for *S*emi-*V*ariables) is formed by pairs $(z, a)$ where $a$ is a fixed element of the crypto-algebra and $z$ belongs to a finite subset of $Cl(F \cup V) \setminus (Cl(F) \cup V)$ where $Cl(X)$ (for any subset $X$ of $\mathcal{F}$) denotes the closure of $X$ under the enciphering and deciphering operators. These pairs correspond to messages that the participant has seen but is unable to label. These messages are then the enciphering of some clear messages under unknown keys : they can be expressed in terms of these clear messages, these keys and the enciphering and deciphering operators. A pair $(z, a)$ of $SV$ is represented by a variable (denoted $z^*$) on the free-algebra with an inclusion constraint on this variable in the corresponding subset of $Cl(F \cup V) \setminus (Cl(F) \cup V)$ and occasionally inequality relations between variables.

The participants make some computations on their state of knowledge. We overvalue them by assuming that each participant is able to apply the enciphering and deciphering operators an infinite number of times but only to the messages and keys that he knows. We thus assume that each participant is able to compute the closure $Cl(F \cup SV)$: this closure is called the *seen fraction* of the participant.

**Remark 2.1** In the Subsection 2.1, we have assumed that the sets of messages and keys are infinite. We can now explain why we have made this hypothesis. Indeed, the different participants are assumed to know the sets of messages and keys and to be able to apply an infinite number of times the operators on any of their finite subsets. If the sets of messages and keys were finite, the participants would be able to compute the encryption of all the messages under all the keys and our model of a perfect cryptosystem would not be right. ∎

The participants are also able to draw some inferences from their computations. All these inferences are modeled in [Tou92] by unifications (i.e. restrictions of the domain of the variables so that two expressions become equal) and contra-unifications (i.e. eliminations of some values from the domain of the variables to prevent two expressions from becoming equal). We then obtain a representation of the complete knowledge (in terms of $F$, $V$, and $SV$) that the participants are able to deduce during an execution of the protocol.

**Example 2.1** We could build the knowledge states of the participants step by step in our coin-flip example introduced in Subsection 1.1. For example, the knowledge states

of participants $A$ and $B$ at the end of the first step are

$$F(A,1) = \{T, H, k\};\ \ V(A,1) = \emptyset;\ \ SV(A,1) = \emptyset$$

and

$$F(B,1) = \{T, H\};\ \ V(B,1) = \{\tilde{k}\};\ \ SV(B,1) = \{em_1^*, em_2^*\}$$

where

$$\begin{aligned} \tilde{k} \in \mathcal{K}\ ;\ \ em_1^*, em_2^* &\in \{e(\tilde{k}, T), e(\tilde{k}, H)\} \\ em_1^* &\neq em_2^*. \end{aligned}$$

$A$'s state of knowledge remains the same until the end of the protocol while $B$'s state of knowledge becomes at the end of Step 3 :

$$F(B,3) = \{T, H, k\};\ \ V(B,3) = \emptyset;\ \ SV(B,3) = \emptyset.$$

■

## 2.3 Modeling the Protocol

The participants try to use their knowledge to cheat. We have thus to study the evolution of the participants' knowledge during the execution of the protocol. We adopt the model of the protocol explained in [Tou91a] and [Tou91b].

A protocol is usually seen as a (finite) set of communicating processes. The *protocol execution tree* is the tree of the possible global states of the protocol. This tree represents the different transitions produced by the actions specified in the protocol. A *protocol state* is the instantiation of some cryptographic variables (i.e. variables of the protocol whose domains are included in the crypto-algebra). Note that here, we only consider the legitimate actions in the protocol without envisaging the illegitimate actions which could be executed by cheating participants or intruders. With each protocol state and participant, we associate the knowledge state of that participant in that state. This state of knowledge is represented as explained in Section 2.

# 3 Definition of a Syntax

In this section, we define a syntax to describe cryptographic protocols. In [Tou91b], this syntax enables us to systematize the construction of the protocol execution tree and the associated knowledge states. By using the syntax and defining an order relation

on the knowledge states, we also prove in [Tou91b] that the participants' knowledge is monotonously increasing during the execution of a protocol. This corresponds to the fact that we want to overvalue the participants' knowledge by assuming that they never forget anything. In this abstract, we only give the definition of the syntax. This syntax is very simple but is sufficient for rather general examples (Needham-Schroeder protocols, Kerberos, X509,... and all the examples considered in [Tou91b]).

We consider the following "actions - primitives".

- choose-key($A$,$k$,$\mathcal{K}_k$,$t$): user $A$ instantiates at step $t$ the cryptographic variable $k$, i.e. $A$ chooses at step $t$ the instantiation of $k$ in the set of keys corresponding to $\mathcal{K}_k$ in the crypto-algebra $\mathcal{C}$.
- choose-message($A$,$m$,$M_m$,$t$): user $A$ chooses at step $t$ the instantiation of the cryptographic variable $m$ in the set of messages corresponding to $M_m$ in the crypto-algebra $\mathcal{C}$.
- send-clear($A$,$B$,$m$,$t$): at step $t$, user $A$ sends message $m$ (in fact its instantiation in the run) to user $B$.
- send-cipher($A$,$B$,$k$,$m$,$ekm$,$t$): at step $t$, user $A$ sends to user $B$ the message $ekm$ but this cryptographic variable has a special instantiation: it is the result of the encryption of the instantiation of $m$ under the instantiation of the enciphering key $k$. $A$ sends thus $E(run(k), run(m))$ to $B$ at step $t$ (where $run(cv)$ denotes the instantiation of the cryptographic variable $cv$ in the run). Briefly, we say that $A$ sends the encryption of $m$ under $k$.
- send-decipher($A$,$B$,$k$,$m$,$dkm$,$t$): at step $t$, user $A$ sends to user $B$ the instantiation of the message $dkm$ i.e. intuitively, the message $m$ that $A$ has decrypted under the deciphering key $k$. In other words, $A$ sends $D(run(k), run(m))$ to $B$ at step $t$.
- cipher($A$,$k$,$m$,$ekm$,$t$): user $A$ enciphers the instantiation $run(m)$ of the message $m$ under the enciphering key $run(k)$ at step $t$ and the result $E(run(k), run(m))$ is the instantiation of the cryptographic variable $ekm$.
- decipher($A$,$k$,$m$,$dkm$,$t$): user $A$ deciphers the message $run(m)$ under the deciphering key $run(k)$ at step $t$ and the result $D(run(k), run(m))$ is the instantiation of the cryptographic variable $dkm$.

**Remark 3.1**

- A 'send-cipher' action is simply a succinct writing of the succession of 'cipher' and 'send-clear' actions and we have a similar property for a 'send-decipher' action.

- The different actions only have a meaning if their authors are able (i.e. have enough knowledge) to execute them: we say that 'the actions have to be *meaningful*'. In [Tou91b], we give conditions for each action described in the syntax to be meaningful.

∎

**Example 3.1**

The description of the coin-flip protocol in terms of the introduced primitives becomes:

- choose-key$(A, k, \mathcal{K}, 1)$ ($\mathcal{K}$ denotes the set of all the keys)
- choose-message$(A, m_1, \{T, H\}, 2)$ $(m_2 \in \{T, H\} \setminus \{m_1\})$
- send-cipher$(A, B, k, m_1, em_1, 3)$
- send-cipher$(A, B, k, m_2, em_2, 4)$
- choose-message$(B, u, \{em_1, em_2\}, 5)$ (the output $a$ of the coin flip is $d(k, u)$)
- send-clear$(B, A, u, 6)$
- send-clear$(A, B, k, 7)$

Note that the choice actions are explicitly specified in the description of the protocol: that enables us in [Tou91b] to systematize the construction of the knowledge states of the participants and the protocol execution tree. ∎

# 4 Method for Formally Verifying the Security of Cryptographic Protocols

Our method can be applied for verifying two aspects of the security of cryptographic protocols.

- Firstly, we can verify the probabilistic properties of cryptographic protocols. In this case, the method consists in grouping the protocol states with the same properties and in analyzing if some participants have enough knowledge to modify the probabilistic distributions in the protocol. The verification of the probabilistic properties is explained in detail in [Tou91a]. Here we will not consider it.

- Secondly, we can model the possible attacks of some participants who could send messages not in accordance with the specifications of the protocol and the possible attacks of intruders who could intercept, delete or replace some messages exchanged on the communication channel. We want to describe here this method. However, for a better understanding, we will omit some details which are very precisely and formally developed in [Tou91b].

## 4.1 Extended Protocol Execution Tree

The protocol execution tree that we defined in Subsection 2.3 only represents the protocol states allowed by the specifications of the protocol. Here, we want to detect the attacks which lead the protocol to a state unallowed by these specifications : in these attacks, the participants send unallowed messages or intruders (who do normally not intervene in the protocol) intercept and replace some messages. We define the notion of *extended protocol execution tree* which represents the protocol states obtained by assuming that, at every time, not only the actions specified by the protocol but also any (nonspecified) action can be executed in the protocol.

## 4.2 Attacks without Detection

When is an intruder able to cheat? We assume that the communication channel is completely open. An intruder is thus always able to intercept any message. However, the noncheating participants know the specifications of the protocol and are waiting for some messages to be exchanged. If they detect any abnormal behavior in the protocol, the noncheating participants are assumed to directly stop the protocol execution. Thus an attack of an intruder only succeeds if the intruder has enough knowledge to replace some messages without detection by the noncheating participants. (For a first approach, we do not consider here the problem of the coalition of several cheating users).

**Example 4.1** Let us consider a first attack on our coin-flip protocol.

1. At the first step, $A$ sends $em_1 = E(k, m_1)$ and $em_2 = E(k, m_2)$ to $B$. An intruder $I$ can intercept these messages, choose another key $k'$, other messages $m'_1$ and $m'_2$ in $\{T, H\}$, and send $em'_1 = E(k', m'_1)$ and $em'_2 = E(k', m'_2)$ to $B$. $B$ does not detect any abnormal behavior if intruder $I$ is able to intercept and replace the messages in the next steps.

2. At the second step, $B$ picks $u$ in $\{em'_1, em'_2\}$ and sends $u$ to $A$. Intruder $I$ can intercept the message, pick $u'$ in $\{em_1, em_2\}$, and send $u'$ to $A$.

3. $A$ sends $k$ to $B$. $I$ can intercept the message and send $k'$ to $B$. $A$ and $B$ obtain the answer of the coin-flip by applying the transformation $D$ but $A$ applies it to $u'$ and $B$ to $u$. They can thus obtain different answers.

This example is very simple and we directly see that this attack will not be detected. Why? Intruder $I$ is able to choose a key $k'$ (different from $k$) and the messages $m'_1$ and $m'_2$ (he is assumed to know the specifications of the protocol and thus to know the messages $T$ and $H$). Then $I$ has no difficulty to intercept and replace the exchanged messages without detection by $A$ and $B$. He can each time replace these messages by messages wanted by the receiver: $I$ has enough knowledge to find messages that do not seem in contradiction with the knowledge that the receiver $A$ or $B$ has about the current state of the protocol i.e. $I$ has enough knowledge to find messages such that some legal states of the protocol (some nodes of the nonextended protocol execution tree) seem still possible.

Let us consider another attack on the coin-flip protocol.

1. At the first step, intruder $I$ does not intercept the messages. Thus, $B$ receives the messages $em_1 = E(k, m_1)$ and $em_2 = E(k, m_2)$ from $A$.

2. At Step 2, intruder $I$ decides to cheat and intercepts the message $u$ ($u$ chosen by $B$ in $\{em_1, em_2\}$ and sent by $B$). Here, we have two scenarii.

   - In the first scenario, $I$ chooses a new key $k'$, other messages $m'_1$ and $m'_2$ in $\{T, H\}$, and computes $em'_1 = E(k', m'_1)$ and $em'_2 = E(k', m'_2)$. Then, he chooses $u'$ between these two messages, and sends $u'$ to $A$.
   - In the second scenario, $I$ replaces the message $u$ by the other message in $\{em_1, em_2\}$ and sends $u'$ ($u' \in \{em_1, em_2\} \setminus \{u\}$) to $A$.

3. At Step 3, $A$ sends the key $k$ to $B$.

   - In the first scenario, $I$ replaces the key $k$ by $k'$.
   - In the second scenario, $I$ does not intercept the key $k$. $B$ receives it but obtains an answer different from $A$'s one. $I$ has thus falsified the answer of the coin-flip protocol in the second scenario.

If we examine the two scenarii above, we immediately see that the first one does not really constitute an attack against the protocol. Indeed, $A$ directly detects an inconsistency at the end of the second step. $A$ knows the specifications of the protocol, he

knows that the message that he has to receive at the end of the second step is one of the two messages he sent at Step 1. These two messages are in his state of knowledge; when he compares the message $u'$ received at the end of the second step with his state of knowledge, he remarks that any legal state of the protocol is impossible.

The second scenario really constitutes an attack. The protocol state at the end of the second step is illegal (i.e. it is a state of the extended protocol execution tree not belonging to the nonextended one) but $A$ receives a message $u'$ that is compatible with his state of knowledge. A legal protocol state (i.e. belonging to the nonextended protocol tree) is possible for $A$ and he thinks that the protocol is in that state. ∎

We have considered obvious attacks on the coin-flip protocol. In general, we can see that an attack will not be detected if a legal protocol state "seems possible" for the noncheating participants from the illegal protocol state. We have thus to define possibility relations between the states of the extended protocol execution tree. "A state seems possible from another state" will mean that the first state is not in contradiction with the knowledge state of the participant in the second state. We have thus to consider the participants' knowledge states in the different states of the extended protocol execution tree, especially in the illegal states.

## 4.3 Knowledge States Associated with the States of the Extended Protocol Execution Tree

If each participant or intruder can try to cheat by sending messages not matching the description of the protocol, the question is "what are the participants' knowledge states?".

We could be tempted to assume that at every step, every participant can send any message (that he knows or can choose). However, this hypothesis would imply that any received message (i.e. any received sequence of bits) could correspond to any element of the free-algebra and thus the reception of any message would bring no information to the receiver. More precisely, when the received message is a primitive element of the crypto-algebra, the receiver could recognize it and label it but he would not know who is the sender whereas when the message is not primitive, it could correspond to any nonprimitive element of the free-algebra and would thus bring no information. For example, in the coin-flip protocol, when $B$ receives two messages at the end of the first step, he knows that these messages are the encryption of $T$ and $H$ under the same key if the protocol is correctly executed but an intruder could have intercepted these messages and replaced them by any messages (not necessarily the encryption of $T$ and $H$). If $B$ considers that at any step an attack is possible and that the messages he receives can be

any messages sent by an intruder, he adds nothing in his state of knowledge and never obtains any information (when the received messages are not clear).

Each participant in a protocol knows its description and tries to deduce from an execution as much information as possible. The only way for him to obtain some information is to assume nevertheless that the participants follow the protocol and send messages of the form specified in its description. So, when a user receives a message, if nothing refutes that fact, he is forced to assume that this message is of the form specified by the protocol.

To summarize, the knowledge states associated with a state of the extended protocol execution tree for a given participant will be built by considering the actions really executed when this participant is the author of these actions or by considering the actions specified in the description of the protocol when this participant is not their author. In the coin-flip example, we can see that the states of knowledge are represented in the same way for any protocol states of the nonextended protocol execution tree corresponding to a same step.

**Remark 4.1** Let us consider the first attack on the coin-flip protocol described in Example 4.1. We have seen that this attack succeeds. We remark that, at each step of the protocol, the knowledge states of participants $A$ and $B$ are consistent i.e. if they analyze their seen fraction (as defined in Subsection 2.2), these fractions are one to one mappings. But if we change a bit this attack, it does no more succeed. Indeed, let us assume that in Step 3, intruder $I$ does not succeed to replace the key $k$ and $B$ receives this key (and not the key $k'$). When he adds $k$ to his state of knowledge and analyzes his seen fraction, $B$ remarks that the encryption of $T$ and $H$ under $k$ does not correspond to the messages he received at Step 1. As we saw in Subsection 2.1, these messages are represented in the set $SV$ of $B$'s state of knowledge by $em_1^*, em_2^* \in \{e(\widetilde{k}, T), e(\widetilde{k}, H)\}$. We thus obtain that in this case $B$'s state of knowledge is inconsistent. $B$ detects the attack. He stops the protocol execution and the attack does not succeed. We see that an additional condition for an attack to succeed is that the knowledge states of the noncheating participants are and remain consistent during the whole execution of the protocol.

Note that the condition of the consistency of the knowledge states of the noncheating participants is necessary but not sufficient for an attack to succeed: even if these knowledge states are consistent, some attacks can be detected by the noncheating participants. Indeed, in the first scenario of the second attack in Example 4.1, we have seen that this attack cannot succeed because the message that $A$ obtains at the end of the second step has not the form that $A$ is waiting for: thus, $A$ remarks that the instantiation of the variable $u$ does not correspond to a legal protocol state. $A$'s seen fraction contains the

knowledge that $A$ is able to obtain about the crypto-algebra but not about the instantiations of the variables of the protocol. Thus, in this case, $A$'s state of knowledge is consistent; nevertheless, $A$ detects the attack. As we have already seen in Example 4.1, another condition for an attack to succeed is that some legal protocol states seem possible for the noncheating participants during the whole execution of the protocol; $A$'s state of knowledge has also to be compatible with some legitimate protocol states. ∎

## 4.4 Possibility Relations between the Protocol States

At a given time of the execution of the protocol, a participant analyzes the protocol execution tree to reject all the protocol states which are not compatible with his knowledge and to discover (if it is possible) the current protocol state. The problem is to know which protocol states are possible for a participant in the current protocol state.

A participant $A$ is assumed to know the specification of the protocol : he is thus able to build the (extended) protocol execution tree and to know the representation of his knowledge in each protocol state. A protocol state $CS'$ will be *possible for a participant $A$ in a given protocol state $CS$*

- if the variables instantiations known by $A$ are identical in $CS$ and $CS'$ and
- if $A$'s knowledge is represented in the same way in $CS$ and $CS'$.

In [Tou91b], we have proved that the relation of possibility considered between cryptographic protocol states of a fixed level of the nonextended protocol execution tree is an equivalence relation.

## 4.5 Model of the Attacks

A participant or an intruder will be able to cheat in a given action if he has enough knowledge to find an instantiation of at least one cryptographic variable which does not belong to the domain of this variable and such that the noncheating participants do not remark any unusual behavior in the execution of the protocol. Because we also consider the attacks of the intruders who could want to impersonate some participants in the protocols, our method is able to detect failures of authentication as well as secrecy in the cryptographic protocols.

We define a '*protocol history*' as an execution of the protocol that each noncheating

participant thinks compatible with the specifications of the protocol i.e. such that

- the knowledge states of the noncheating participants have to be consistent during the whole execution of the protocol;
- there have to be some legal protocol states which seem possible for the noncheating participants.

When a protocol history contains some cheatings, the noncheating participants are not able to detect them; only a cheating user is then conscious of his cheating.

By definition, a cryptographic protocol will be *secure* if any protocol history corresponds to a legitimate execution of the protocol i.e. to a branch of the (nonextended) protocol execution tree; thus if the extended protocol tree is reduced to the nonextended one by the elimination of the branches which do not correspond to protocol histories. In fact, we could build the extended protocol execution tree by only considering the instantiations of the variables possible for the noncheating participants. If some branches can be added by this way to the nonextended tree, the protocol is insecure and these branches correspond to attacks.

In [Tou91b], we have studied by this method the security of other protocols like Needham-Schroeder key distribution protocols, the Kerberos protocol, and the X.509 standard. We have found the already known attacks and also some new attacks.

# 5 Specification and Implementation Phases

If we consult the literature, we notice that some protocols (and among others many zero-knowledge protocols) are described by using very specific cryptographic functions. The analysis of their security is closely connected with the security of these functions: we call these protocols 'specific'. However, our method analyzes the security of protocols by assuming that the cryptosystems being used are perfect. The security of the protocol is thus completely separated from the security of the cryptosystem.

So, our method cannot directly be applied to specific protocols. One could say that this is a limitation of our method. We do not think so. The application of our method to specific protocols commits us, at first, to generalize them by precisely specifying the required properties for the cryptosystem. This generalization can seem tedious but enables us to deeply understand the basic mechanisms of protocols and sometimes to find some inaccuracies in their description. In fact, this generalization inverses the steps of the

designer of a protocol who first thinks of a general scheme and then tries to implement it by taking advantage of the properties of some specific cryptosystem.

For the existing specific protocols, this generalization is necessary. But we think that the syntax (possibly completed by other actions) defined in Section 3 could provide a language for describing the cryptographic protocols. The designer of protocols could directly write their protocol in terms of this syntax. That would have three important advantages.

1. The designer could directly apply the method of formal verification described in Section 4 and thus he would only publish a protocol when he is sure that it is secure.
2. The required properties for the cryptosystem to be used would precisely be specified.
3. The imprecisions in the description of a protocol would directly be detected.

The specification phase would contain the conception and description of the protocol in terms of the syntax, the formal proof of its security, and the specification of the properties that the cryptosystem to be used has to satisfy. The implementation phase would contain the choice of the cryptosystem satisfying all these properties. This choice takes in account all the practical requirements (performance, security level, ...).

## 5.1 Example

We present here a signature protocol inspired from the Fiat-Shamir scheme ([FS87]). Brickell, Lee and Yacobi give in [BLY88] two specific signature schemes for secure audio teleconferencing between $N$ participants connected to a central facility called a '*bridge*'. One of these protocols is based on quadratic residues and the other on discrete logarithms. These two protocols seems rather different. We show that if the authors had followed our method of separating the two phases of specification and implementation, they would only obtain one protocol. The two protocols only differ in the choice of the cryptosystems but in [BLY88], the specific cryptosystems completely hide the similarities between the two protocols. Note that the authors were conscious that the two protocols were based on the same idea when they designed these protocols but this idea does not clearly appear when we consider the two published protocols ([BLY88]).

In the protocol, three entities are considered:

- a central authority
- a central facility called *bridge* which selects $L$ active speakers, adjusts the volume, adds their (encrypted or not) signals and broadcasts the result. $L$ is an integer larger than 2.
- different participants who want to jointly sign a message $m$ or the contents of the conference.

### 5.1.1 The Two Protocols Presented in [BLY88]

We give here the specifications of the two protocols given in [BLY88].

**First Specific Protocol: a Quadratic Residue Signature Protocol**
Let $n$ be a public *Blum integer* (i.e. the product of two large primes which are congruent to 3 modulo 4). $n$ is public but its factorization is known only by the central authority. $P$ denotes the list of participants who want to sign a message $m$: each user $u$ can be identified by some public data $I_u$ (e.g., his name, address, social security number,...).

Let $h$ be a cryptographically secure pseudo random function known by every participant. The numbers $v_{u_j} = h(I_u, j)$ $(j = 1, \ldots, l$; for $l$ an integer sufficiently large for the protocol to be secure) can be computed by every user. By renaming if necessary, we can assume that $v_{u_1}, \ldots, v_{u_l}$ all have the Jacobi symbol $+1$ modulo $n$ and thus, $+v_{u_i}$ or $-v_{u_i}$ $(i = 1, \ldots, l)$ is a quadratic residue modulo $n$.

For each user $u$, the central authority computes $s_{u_1}, \ldots, s_{u_l}$ such that for $j = 1, \ldots, l$

$$s_{u_j}^2 = \pm v_{u_j}^{-1} \pmod{n}$$

(the inverse is computed in arithmetic modulo $n$) and transmits to $u$ the $\{s_{u_j}\}$ in a secret and (assumed) quite secure way. Let also $g$ be a one-way function which maps very long messages to sequences of $l$ bits and $m$ the message to be signed. The quadratic residue protocol is described in [BLY88] as follows.

1. Each participant $1 \leq u \leq N$ picks a random $0 \leq r_u \leq n$, computes $x_u = r_u^2 \pmod{n}$ and sends $x_u$ to the bridge.
2. The bridge computes $X = \prod_{u=1}^{N} x_u \pmod{n}$ and broadcasts it.
3. Each participant $u$ computes $g(m, X, P) = (e_1, \ldots, e_l)$.

4. Each participant $u$ computes $y_u = r_u \prod_{j=1}^{l} s_{u_j}^{e_j}$ and transmits $y_u$ to the bridge.
5. The bridge computes $Y = \prod_{u=1}^{N} y_u \pmod{n}$ and broadcasts it.
6. Each participant computes

$$Z = Y^2 \prod_{j=1}^{l} V_j^{e_j} \pmod{n}$$

where $V_j = \prod_{u=1}^{N} v_{u_j} \pmod{n}$ and $v_{u_j}^{-1} = \pm s_{u_j}^2 \pmod{n}$.

7. $Y$ is a valid signature if and only if $Z = \pm X \pmod{n}$ and the list $P$ of parties matches the list of parties signing the documents.

**Second Specific Protocol: a Discrete Logarithm Signature Protocol**
Let $T$ be a prime power, $\alpha$ a generator of the multiplicative group of the Galois field $GF(T)$ and $\equiv$ the congruence in $GF(T)$. Each user $u$ ($1 \leq u \leq N$) has $l$ secrets $s_{u_j}$ ($1 \leq j \leq l$) ($l$ is an integer large enough for the protocol to be secure) and publishes

$$w_{u_j} \equiv \alpha^{-s_{u_j}}.$$

This protocol based on the problem of computing discrete logarithms is described in [BLY88] as follows.

1. $u$ picks random $r_u \in [0, T[$, computes $a_u \equiv \alpha^{r_u}$ and transmits it to the bridge.
2. The bridge computes $X \equiv \prod_{u=1}^{N} a_u$ and broadcasts it.
3. Let $(e_1, \ldots, e_l)$ be the first $l$ bits of $\alpha^{\sigma}$ in $GF(T)$, where $\sigma$ is the concatenation $(m, X, P)$. $u$ computes $y_u = r_u + \sum_{e_j=1} s_{u_j} \pmod{T-1}$ and transmits it to the bridge.
4. The bridge computes $Y = \sum_{u=1}^{N} y_u \pmod{T-1}$ and broadcasts it.
5. $u$ computes $W \equiv \prod_{u=1}^{N} \prod_{e_j=1} w_{u_j}$ and then $Z \equiv \alpha^Y W$.
6. o.k. iff $Z \equiv X$.

The specifications of the protocols are different but seem similar on some points. Let us now explain our method to describe the protocol. We begin by the specification phase to define the structure of the protocol.

### 5.1.2 Specification Phase

We directly give the description of the (generalized) protocol expressed in terms of our syntax defined in Section 3 and the properties required for the cryptosystem to be used.

#### Required Properties for the Cryptosystem to be used

We assume that

- the crypto-algebra corresponds to a public-key system.
- there is a public pseudo-random function $g$ of which the domain is the set of possible messages of the crypto-algebra and the codomain the set of the different sequences of $l$ bits.
- There are two binary operations $op_1$ and $op_2$, publicly known, defining abelian groups structures on the set $\mathcal{M}$ of messages of the crypto-algebra, and such that

$$E(k_p, op_2(m_1, m_2)) = op_1(E(k_p, m_1), E(k_p, m_2)) \ \forall m_1, m_2 \in \mathcal{M} \qquad (1)$$

  where $E$ is the enciphering function. Corresponding operations are defined on the free-algebra.

#### Preliminary choices

Each user $u$ ($u = 1, \ldots, N$) chooses $l$ messages in $\mathcal{M}$ ($l$ is a parameter to be fixed according to the desired degree of security). These messages are denoted

$$s_{u_1}, \ldots, s_{u_l}$$

and remain secret. User $u$ computes and publishes the inverse (for the operator $op_1$ ) of the encryption of these messages under the public key $k_p$ of the central authority

$$v_{u_1} = (e(k_p, s_{u_1}))^{-1_{(op_1)}}, v_{u_2} = (e(k_p, s_{u_2}))^{-1_{(op_1)}}, \ldots, v_{u_l} = (e(k_p, s_{u_k}))^{-1_{(op_1)}} :$$

each user is assumed to know the elements $\{v_{u_i}, i = 1, \ldots, l\}$ corresponding to the other users.

### Description of the Protocol

1. $\bigwedge_{u=1}^{N}$ (choose-message$(u, m_u, \mathcal{M}, 1) \wedge$ send-cipher$(u, B, k_p, m_u, em_u, 1)$)
   if $B$ denotes the bridge.
2. $\bigwedge_{u=1}^{N}$ send-clear$(B, u, X, 2)$ where $X = op_1(em_1, \ldots, em_N)$.
3. Let $g_1, \ldots, g_r$ be the positions of the bits of $g(m, op_1(em_1, \ldots, em_N), P)$ equal to '1' (assuming their number is $r$ ($r \geq 0$)) where $P$ denotes the list of participants who want to sign the message $m$.
   $\bigwedge_{u=1}^{N}$ send-clear$(u, B, op_2(m_u, s_{ug_1}, \ldots, s_{ug_r}), 3)$
4. $\bigwedge_{u=1}^{N}$ send-clear$(B, u, Y, 4)$ where

$$Y = (op_2(op_2(m_1, s_{1g_1}, \ldots, s_{1g_r}), op_2(m_2, s_{2g_1}, \ldots, s_{2g_r}), \ldots, op_2(m_N, s_{Ng_1}, \ldots, s_{Ng_r})).$$

We will assume that the seen fraction of the participants (as defined in Subsection 2.2) is closed under the operators $op_1$, $op_2$, and the enciphering function $E$. The signature will be verified if at the end of the protocol execution, the participants' state of knowledge is still consistent. Indeed, in this case,

$$op_1(E(k_p, Y), op_1(v_{1g_1}, \ldots, v_{1g_r}, v_{2g_1}, \ldots, v_{2g_r}, \ldots, v_{Ng_1}, \ldots, v_{Ng_r}) = X. \qquad (2)$$

Note that this protocol uses very little cryptography: one public transformation is known but it is always used with the same public key $k_p$ and the corresponding deciphering function is not specified because the central authority does not really participate in the protocol. Thus, the states of knowledge do not need to be assumed closed under the decryption function.

### States of Knowledge

Let $i$ be a user who wants to cheat: in order to prepare his cheating, he has stored every message exchanged during several executions (let us assume $n$ ($n \geq 1$) executions) of the protocol. The other users suspect nothing and thus are only conscious of the messages that they receive themselves and forget them when the corresponding execution is finished.

Thus $i$'s knowledge state before the start of the $(n+1)^{\text{st}}$ execution of the protocol is $K(i,0) = F(i,0) \cup V(i,0) \cup SV(i,0)$ where

$$F(i,0) = \{s_{i_j} : j = 1, \ldots, l; k_p, m_i^{(1)}, \ldots, m_i^{(n)}\},$$

$$V(i,0) = \{\widetilde{s_{v_j}} : j = 1,\ldots,l,; \widetilde{m_v^{(k)}} : k = 1,\ldots,n; v \in P \setminus \{i\}\},$$

$$\begin{aligned} SV(i,0) \;=\; & \{(E(k_p, \widetilde{s_{v_j}}))^{-1_{(op_1)}} : j = 1,\ldots,l; em_v^{(k)*} = E(k_p, \widetilde{m_v^{(k)}}); \\ & op_2(\widetilde{m_v^{(k)}}, \widetilde{s_{v_{g_1^{(k)}}}}, \ldots, \widetilde{s_{v_{g_{r(k)}^{(k)}}}}) : k = 1,\ldots,n; v \in P \setminus \{i\}\}. \end{aligned}$$

The preliminary states of knowledge of the other participants ($u \in P \setminus \{i\}$) are

$$F(u,0) = \{s_{u_i} : i = 1,\ldots,l, k_p\}, V(u,0) = \{\widetilde{s_{v_i}} : i = 1,\ldots,l; v \in P \setminus \{u\}\},$$

$$SV_{u,0} = \{(E(k_p, \widetilde{s_{v_i}}))^{-1_{(op_1)}} : i = 1,\ldots,l; v \in P \setminus \{u\}\}.$$

The bridge is assumed trusted and we do not consider its states of knowledge. After the first step, we have

$$\forall u \in P : F(u,1) = F(u,0) \cup \{m_u\}, V(u,1) = V(u,0) \cup \{\widetilde{m_v} : v \in P \setminus \{u\}\},$$

$$\forall u \in P \setminus \{i\} : SV(u,1) = SV(u,0),$$

$$SV(i,1) = SV(i,0) \cup \{E(k_p, \widetilde{m_v}) : v \in P \setminus \{i\}\}.$$

After the second step,

$$\forall u \in P \setminus \{i\} : SV(u,2) = SV(u,1) \cup \{op_1(E(k_p, \widetilde{m_1}), \ldots, E(k_p, \widetilde{m_N}))\}.$$

After the third step,

$$SV(i,3) = SV(i,2) \cup \{op_2(\widetilde{m_v}, \widetilde{s_{vg_1}}, \ldots, \widetilde{s_{vg_r}}) : v \in P \setminus \{i\}\}.$$

At the end of the protocol, $i$'s knowledge state has not changed anymore because the seen fraction of each user is assumed to be closed under the encryption function but also under the operations $op_1$ and $op_2$ (considered here in the free-algebra). But the states of knowledge of the other users become

$$\begin{aligned} \forall u \in P \setminus \{i\} : SV(u,4) \;=\; & SV(u,3) \cup \{op_2(op_2(\widetilde{m_1}, \widetilde{s_{1g_1}}, \ldots, \widetilde{s_{1g_r}}), \ldots, \\ & op_2(\widetilde{m_u}, \widetilde{s_{ug_1}}, \ldots, \widetilde{s_{ug_r}}), \ldots, op_2(\widetilde{m_N}, \widetilde{s_{Ng_1}}, \ldots, \widetilde{s_{Ng_r}}))\}. \end{aligned}$$

**Analysis of the Protocol**

At first glance, this signature scheme seems very secure but our method detects a possibility of cheating. Indeed, there is a protocol history which does not correspond to the normal execution of the protocol: the intruder has a probability larger than zero to manage to impersonate any other user. Even if the probability of cheating is very very

small, as soon as it is larger than zero, the possibility of cheating is detected by our method.

If an intruder wants to impersonate a user $u$ and to give the impression that $u$ has really signed the message $m$ while $u$ does not, he can make the first step instead of $u$: $i$ chooses a message $m_u$ and sends $B$ the encryption of this message under the public key $k_p$. The second step is performed by $B$ and thus does not constitute any difficulty for the intruder. The delicate point is Step 3. The intruder can compute $g(m, op_1(em_1, \ldots, em_N), P)$ but has to send a message which has the same value as $op_2(m_u, s_{u_{g_1}}, \ldots, s_{u_{g_r}})$ without knowing $s_{u_{g_1}}, \ldots, s_{u_{g_r}}$. We can assume that the intruder $i$ has discovered the values of $m_u^{(1)}, \ldots, m_u^{(n)}$ which do not need to be well protected because they are normally used only once. Thus, the intruder knows '$op_2(\widetilde{s_{u_{g_1^{(j)}}}}, \ldots, \widetilde{s_{u_{g_{r(j)}^{(j)}}}})$' for each previous execution ($j = 1, \ldots, n$). If we assume that the operation $op_2$ is such that $op_2(a, b) = op_2(c, d) \Leftrightarrow a = c, b = d$ for any elements $a, b, c, d$ of the domain of $op_2$ (in fact, it is sufficient that all the applications of $op_2$ on different combinations of $\{s_{u_j} : j = 1, \ldots, l\}$ are different),

$$op_2(s_{u_{g_1}}, \ldots, s_{u_{g_r}}) = op_2(\widetilde{s_{u_{g_1^{(j)}}}}, \ldots, \widetilde{s_{u_{g_{r(j)}^{(j)}}}})$$

if and only if

$$s_{u_{g_1}} = \widetilde{s_{u_{g_1^{(j)}}}} \wedge \ldots \wedge s_{u_{g_r}} = \widetilde{s_{u_{g_r^{(j)}}}}. \tag{3}$$

The probability to have Relation (3) is the probability that

$$g(m, op_1(em_1, \ldots, em_N), P) = g(m^{(j)}, op_1(em_1^{(j)}, \ldots, em_N^{(j)}), P) : \tag{4}$$

this probability is $\frac{1}{2^l}$. If $l$ is large enough, the protocol is secure in practice.

### 5.1.3 Implementation Phase

We have to choose a cryptosystem which satisfies the properties previously specified. In this protocol, only an encryption function is used with always the same public key. In [BLY88], the authors do not choose real cryptosystems but two one-way functions (one for each specific protocol) that they estimate appropriate (enough secure and efficient) for the practice.

1. The first function is

   $$f(n, m) = m^2 \pmod{n}$$

   where $n$ is publicly known but its factorization is only known by the central authority. The operators $op_1$ and $op_2$ are then the multiplication in arithmetic modulo $n$.

2. The second function is

$$f(T, m) = \alpha^m$$

in the arithmetic of $GF(T)$ where $T$ and $\alpha$ are the public information. The operator $op_1$ is then the multiplication in $GF(T)$ and $op_2$ the addition modulo $(T-1)$.

# 6 Conclusions

In the design of the cryptographic protocols, we propose to separate the specification and implementation phases.

The specification phase consists in describing the protocol in terms of the syntax introduced in Section 3 and in very precisely specifying the properties that the cryptosystem to be used has to satisfy. The implementation phase consists in choosing cryptosystems (for example based on the problem of computing square roots of quadratic residues or of computing discrete logarithms...) which satisfy these conditions. The word 'cryptosystem' is taken here in a very wide sense: it can be a real cryptosystem or a one-way function enough secure in practice. The 'efficiency' factor strongly influences the choice of cryptosystems.

The main advantage of this approach is that the security of the protocol can be formally verified before its publication. This verification can be made by applying the method we describe in [Tou91b] which is a generalization of [Tou91a]. Our method is conceptually simple and very close to the reasoning of a participant or of an intruder who wants to cheat. This method is also very general: it is able to detect any attack on the cryptographic protocols and to verify the probabilistic properties of these protocols. It can be applied in public or private key cryptography, to protocols preserving the secret or the authenticity of some data, providing digital signatures,... In [Tou91b], we have studied by this method the security of other protocols like Needham-Schroeder key distribution protocols, the Kerberos protocol, and the X.509 standard. We have found the already known attacks and also some new attacks. The other methods often study the security of only one category of cryptographic protocols and do not consider the probabilistic aspect.

Another advantage of separating the specification and implementation phases is that the properties the cryptosystems have to satisfy are precisely specified. In the implementation phase, these properties can be lightened if the security is estimated sufficient in practice. Note that the properties can be very general. An extension of our method could be the definition of languages to specify the properties of the protocols and of the

cryptosystems.

## Acknowledgements

I would like to thank Professors P. Wolper, T.A. Banh, J.-J. Quisquater, M. Merritt, A. Danthine, D. Ribbens for many helpful discussions about this work and many encouragements. I also address my thanks to F.N.R.S. and S.P.P.S. (Belgium) for financial support.

# References

[BAN89] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. Technical Report 39, Digital — Systems Research Center (SRC), 1989.

[Bie89] P. Bieber. *Aspects Epistémiques des Protocoles Cryptographiques.* PhD thesis, Université Paul-Sabatier de Toulouse (Sciences), October 1989.

[BLY88] E.F. Brickell, P.J. Lee, and Y. Yacobi. Secure Audio Teleconference. In C. Pomerance, editor, *Lecture Notes in Computer Science. Advances in Cryptology — CRYPTO'87, #293*, pages 418–426. Springer-Verlag, 1988.

[BM84] M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SIAM Journal on Computing*, 13(4):850–864, 1984.

[CCI88] CCITT. CCITT blue book, Recommendation X.509, The Directory - Authentication Framework , November 1988.

[Cop89] Don Coppersmith. Analysis of ISO/CCTTI Document X.509 Annex D. IBM Thomas J. Watson Research Center, Yorktown Heights, June 1989.

[FS87] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In A. M. Odlyzko, editor, *Lecture Notes in Computer Science. Advances in Cryptology — CRYPTO'86, #263*, pages 186–194. Springer-Verlag, 1987.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof-Systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[GNY90] L. Gong, R. Needham, and R. Yahalom. Reasoning about Belief in Cryptographic Protocols. In *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 234–248. IEEE Computer Society Press, 1990.

[IM90] C I'Anson and C. Mitchell. Security Defects in CCITT Recommendation X.509 - The Directory Authentication Framework. *Computer Communication Review*, 20(2):30–34, 1990.

[ISO89] ISO. 7498-2. Information processing systems - Open Systems Interconnection - Basic Reference Model - Part 2: Security Architecture , 1989.

[Kem89] R. A. Kemmerer. Analyzing Encryption Protocols Using Formal Verification Techniques. *IEEE Journal on Selected Areas in Communications*, 7(4):448–457, 1989.

[MCF87] J. K. Millen, S. C. Clark, and S. B. Freedman. The Interrogator: Protocol Security Analysis. *IEEE Transactions on Software Engineering*, 13(2):274–288, 1987.

[Mea90] C. Meadows. Representing Partial Knowledge in an Algebraic Security Model. In *Proceedings of the Computer Security Foundations Workshop III*, pages 23–31. IEEE Computer Society Press, 1990.

[Mer83] M. J. Merritt. *Cryptographic Protocols.* PhD thesis, Georgia Institute of Technology, 1983.

[MW85] M. Merritt and P. Wolper. States of Knowledge in Cryptographic Protocols (extended abstract). Unpublished Manuscript, 1985.

[Syv91] P. Syverson. The Use of Logic in the Analysis of Cryptographic Protocols. In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*, pages 156–170. IEEE Computer Society Press, 1991.

[Tou89] M-J. Toussaint. Reasoning about Probabilistic Properties of Cryptographic Protocols (extended abstract). Abstract of the talk at the F.N.R.S. day on Computer Security, May 1989.

[Tou91a] M-J. Toussaint. Formal Verification of Probabilistic Properties in Cryptographic Protocols (Extended Abstract). in the proceedings of ASIACRYPT'91, November 1991.

[Tou91b] M.-J. Toussaint. *Verification of Cryptographic Protocols.* PhD thesis, Université de Liège (Belgium), 1991. in the Publications Collection (to appear).

[Tou92] M-J. Toussaint. Deriving the Complete Knowledge of Participants in Cryptographic Protocols (Extended Abstract). In J. Feigenbaum, editor, *Lecture Notes in Computer Science. Advances in Cryptology — CRYPTO'91, #576*, pages 24–43. Springer-Verlag, 1992.

[TW91] M-J. Toussaint and P. Wolper. Reasoning about Cryptographic Protocols (Extended Abstract). In Joan Feigenbaum and Michael Merritt, editors, *Distributed Computing and Cryptography (October 1989)*, pages 245–262. DIMACS - Series in Discrete Mathematics and Theoretical Computer Science (AMS - ACM), 1991. Volume 2.