University "Alexandru Ioan Cuza" of Iaşi
Faculty of Computer Science

T E C H N I C A L R E P O R T

Secrecy for Security Protocols

(Ph.D. Thesis)

Cătălin V. Bîrjoveanu

TR 07-02, January 2007

ISSN 1224-9327

Universitatea "Alexandru Ioan Cuza" Iaşi
Facultatea de Informatică
Str. Berthelot 16, 6600-Iaşi, Romania
Tel. +40-32-201090, email: bibl@infoiasi.ro

Secrecy for Security Protocols

Cătălin V. Bîrjoveanu
Department of Computer Science,
"Al.I.Cuza" University of Iaşi,
Iaşi, România

Thesis submitted to the "Al. I. Cuza" University of Iaşi
for the degree of Doctor of Philosophy
in Computer Science

October 2006

Cătălin V. Bîrjoveanu
Department of Computer Science,
"Al.I.Cuza" University of Iaşi,
Iaşi, România
E-mail: cbirjoveanu@infoiasi.ro

Prof. Dr. Gheorghe Grigoraş, Chair ("Al. I. Cuza" University of Iaşi)
Prof. Dr. Ferucio Laurenţiu Ţiplea, Supervisor ("Al. I. Cuza" University of Iaşi)
Prof. Dr. Adrian Atanasiu (University of Bucharest)
Prof. Dr. Victor Patriciu (Military Technical Academy of Bucharest)
Prof. Dr. Constantin Popescu (University of Oradea)

To my family

Contents

Preface

The information is a key element in our society, so its confidentiality (secrecy), authenticity and integrity must be ensured. Security protocols are prescribed sequences of interactions between entities, designed to achieve the above security goals across distributed systems. The environment in which security protocols take place is assume to be hostile, meaning that it may contain intruders who can read, modify, and delete informations. Security protocols are often wrong due to the extremely subtle properties they are supposed to ensure. The need of formal methods for analyzing the security protocols is thus obvious. Formal methods for security protocols analysis were first advocated by Needham and Schroeder in [48]. Attempts to develop frameworks and tools for security protocols analysis goes back over 20 years, but this field is still highly active in the research community.

Undecidability of secrecy for security protocols is a well-known issue. Yet, imposing various (reasonable) restrictions, subclasses of security protocols for which secrecy is decidable are obtained. This thesis focuses on the decidability and complexity of the secrecy problem for security protocols. Several results on this field have been developed in the literature, but some of them are wrong and, furthermore, it is hard to relate all results due to a large variety of formalisms used to develop them. The thesis corrects the wrong results identified and presents all the results obtained under the same formalism, providing a complete picture of the complexity of the secrecy problem for security protocols.

The thesis is structured in five chapters. In Chapter 1 we present a general overview of security protocols. In Chapter 2 we briefly look at the most popular models for security protocols and we describe the model that will be used in this thesis. Chapter 3 provides some undecidability results for the secrecy problem. In Chapter 4 we identify three subclasses of security protocols for which secrecy is decidable and provide the complexity results for the decision algorithms. Chapter 5 comes with the conclusions and future work.

First of all I would like to thank my supervisor Professor Dr. Ferucio Laurenţiu Ţiplea for his constant guidance and encouragement in the last five years. All the work presented here was done with him. I am especially in debited to him because he encouraged me to take up a career in research. Also, I would like to thank my colleagues for meaningful conversations we had.

Last and most I would like to thank my family for its continuous support.

Iaşi, October 2006
Cătălin V. Bîrjoveanu

Chapter 1

Introduction

”In this age of universal electronic connectivity, of viruses and hackers, of electronic eavesdropping and electronic fraud, there is no time at which security does not matter.”–William Stallings [61].

Motivations In many usual applications over the internet (e-commerce, e-payment, e-gambling, e-lotteries, e-auctions, e-elections, etc) security is a critical issue. Security protocols are a fundamental part of such applications. The first paper that introduces security protocols is that of Needham and Schroeder, in 1978 [48].

A *security protocol* specifies a set of rules under which a sequence of messages is exchanged between two or more parties in order to achieve security goals such as authentication or establishing new shared secrets. Depending on the application in which the protocol is use, achieving the security goals might require that some security properties to be hold: *secrecy, authentication, integrity, non-repudiation, fairness, anonymity,* etc. Examples of security protocols include the *Kerberos authentication scheme* [44] used to manage encrypted passwords on clusters of interconnected computers, *Secure Sockets Layer* (SSL) [30] used by internet browsers and servers to carry out secure internet transactions, *Secure Electronic Transaction* (SET) protocol [59] used to provide a standard for safe secure credit card transactions over the internet, etc. Since security protocols are a central part of the infrastructure for secure communications and networking, it is very important to analyze them in detail before they are deployed. When analysing security protocols we have to consider that they are taking place in a hostile environment, namely in the presence of an *intruder*. Usually, the set of agents engaging the protocol is divided in legitimate or honest agents and the intruder. The intruder is a malicious agent that can intercept, remove or change any message on the network and construct fake messages. The correctness of a protocol means that a

certain security goal is achieved even in the presence of the intruder. For example, *secrecy* means that the informations are accessible only to authorized parties, and *authentication* ensures that the origin of a message is correctly identified. In the context of intruder's presence, encrypted communications will be needed.

Cryptographic primitives Secure communications on the network relies on a set of basic functions that we will refer to as *cryptographic primitives*: symmetric and public-key encryption/decryption, digital signatures schemes, etc.

An encryption primitive uses an encryption key to produce an encoded message (ciphertext). The reverse primitive is called decryption and obtains the original message (plaintext) using the right decryption key. Once a message is encryped, it can be send over the network, such that its original meaning can be understood only by the agents that have the decryption key.

In this thesis we will treat the cryptographic primitives in an abstract way, without beeing interested in the way they are implemented.

In the *symmetric cryptography* the same key is used both for encryption and decryption. For two agents to communicate securely they need to share a secret: the cryptographic key. There are many protocols for establishing a shared secret (see for example Diffie-Hellman key exchange protocol [20]). The most recent well known symmetric cryptosystem is AES (Advanced Encryption Standard [28]).

Asymmetric or public-key cryptography has the advantage that for two agents to communicate, a protocol for key distribution is no longer needed. Each agent A has its own key pair, consisting of a public key K_A (used for encryption), and a private key $\overline{K}_A$ (used for decryption). Everybody is allowed to learn the public key K_A, but the private key $\overline{K}_A$ is known only to A. Any agent can encrypt a message using A's public-key, but only A can decrypt it using $\overline{K}_A$. The best known public-key cryptosystem is RSA [56].

Most researches on security protocols (also called cryptographic protocols) are developed under *perfect encryption assumption* [22]. This assumption states that an agent can not learn anything about an encrypted message unless he knows decryption key. Also, an agent can not built an encryped message unless he knows the plaintext and the encryption key. Security of the protocols is not assured even if all cryptographic primitives used by the protocols are perfectly secure. This is because the protocols itself may have weaknesses due to the logical flaws in their design.

Standard notation Let us consider a simplified version of the Needham-Schroeder public-key protocol [48], one of the most faimous protocol from literature. For many years this protocol was considered correct, until 17 years after its appearence

was proved insecure by G. Lowe [38]. Using the standard notation, the protocol may be written as follows:

$$\begin{array}{lcl} 1. \quad A \rightarrow B & : & \{x, A\}_{K_B} \\ 2. \quad B \rightarrow A & : & \{x, y\}_{K_A} \\ 3. \quad A \rightarrow B & : & \{y\}_{K_B} \end{array}$$

The protocol consists from a list of communications of the form $A \rightarrow B : M$ meaning that the agent A sends the message M to the agent B, which receives the message. K_A and K_B are the public keys of the agent A, respectively B and the notation $\{x\}_K$ is used to denote the x encrypted with the key K. In the protocol we can distinguish two *roles*: the initiator role that describes the actions of the initiator (A) and the responder role that describes the actions of the responder (B). The names A, B, x, y from the protocol specification are abstract terms that are instantiated with concret terms when the protocol is run. The protocol specification describes the sequence of the communications and the syntax of the messages rather then its semantics.

In the first step, the agent A initiates the protocol by creating a new *nonce* x and encrypting it (along with his identity) with B's public key. A nonce stands for "number once used" and it is a newly generated, unguessable number. When B receives the message, he and only he, can decrypt it and learns x. B responds to A creating a new nonce y and encrypting it (along with x) with A's public key. When A receives the message, he checks that the nonce placed on the first field in the message matches with the nonce he sent in the first step. If those two are matching, then A is confident that he is talking to B, since only B could decrypt the first message to obtain x. A returns to B the message y encrypted with K_B. When B receives last message, he does the same verification as A with respect to his own nonce to be sure that is talking to A. After the three communication steps, the intended result of this protocol is that A and B know that they are talking with each other and they share acces to the values x and y, and no other agent knows this values. The agents A and B can use these values, either for later re-authentication or for use some hash value of them as a shared session key for later secret communications.

The protocol specification describes the intended behaviour of the honest agents and does not describe unexpected actions of the intruder. Therefore, it is not sufficient to fully describe the behaviour of the protocol.

Attack strategies Although the security protocols seems simple, and the literature is full of protocols that appear be secure, they fall prey to subtle attacks. As we stated above, G. Lowe discovered an attack on the Needham-Schroeder

public-key protocol. In what follows we will show Lowe's attack:

$$\begin{array}{lll} \alpha.1. & A \rightarrow I & : \{x, A\}_{K_I} \\ \beta.1. & I_A \rightarrow B & : \{x, A\}_{K_B} \\ \beta.2. & B \rightarrow I_A & : \{x, y\}_{K_A} \\ \alpha.2. & I \rightarrow A & : \{x, y\}_{K_A} \\ \alpha.3. & A \rightarrow I & : \{y\}_{K_I} \\ \beta.3. & I_A \rightarrow B & : \{y\}_{K_B} \end{array}$$

The attack involves two *interleaved sessions* of the protocol: the first, whose messages are labelled with α, in which A initiates a session with the intruder I and the second, whose messages are labelled with β, in which I initiates a session with B. We use the notations $I_A \rightarrow B$ meaning that the intruder sends a message to B impersonating A and $B \rightarrow I_A$ meaning that the intruder intercepts a message from B intended to A. The attack scenario is as follows: A starts a session with the intruder I thinking of him as an honest agent. The intruder impersonates A sending a message to B containing the nonce x obtained in previous step. B is thinking that the message came from A and he responds according to the protocol. The intruder intercepts the message from B, but he can't decrypt it, so he forwards the message to A using him as an oracle. A decrypts the message obtaining y and returns it to the intruder. I returns y to B, thus completing the session with B. At the end, authentication and secrecy goals fail: B is fooled to think that is talking to A, when in fact A is talking to the intruder; also B thinks that the nonce y is a secret known only to A and B, but in the message $\alpha.3$ the intruder learns y.

The Needham-Schroeder-Lowe protocol is proposed by Lowe [38] to fix the error from the above protocol:

$$\begin{array}{lll} 1. & A \rightarrow B & : \{x, A\}_{K_B} \\ 2. & B \rightarrow A & : \{x, y, B\}_{K_A} \\ 3. & A \rightarrow B & : \{y\}_{K_B} \end{array}$$

This protocol differs from the original Needham-Schroeder public-key protocol by including the responder's identity in the second message.

Another attack strategy is *replay attack*: a message from a previous session of the protocol is recorded by the intruder and is replayed as a message component in the current session. To illustrate such an attack, we will present the Needham-Schroeder secret-key protocol [48]:

$$\begin{array}{lll} 1. & A \rightarrow S & : A, B, x \\ 2. & S \rightarrow A & : \{x, B, K, \{K, A\}_{K_{BS}}\}_{K_{AS}} \\ 3. & A \rightarrow B & : \{K, A\}_{K_{BS}} \\ 4. & B \rightarrow A & : \{y\}_K \\ 5. & A \rightarrow B & : \{y - 1\}_K \end{array}$$

The goal of this protocol is to establish a new shared key between two agents A and B using a third trusted party, namely the server S. K_{AS} and K_{BS} are long-term keys shared between A and S, respectively B and S. A sends a message to S that includes a nonce x, requesting from the server a key to communicate with B. The server responds to A by a message that includes the nonce x, a new key K, and a certificate encrypted with K_{BS} intended to B. A decrypts the message, verifies that the nonce is matching with the one he sent in the first step and then can deduce that the key K is sent as a response to his request. A can't decrypt the certificate for B and forwards it to B. B decrypts his certificate and learns the key. Then, B sends to A a nonce y encrypted with K to be sure that A is present currently, since he certificate can be a replay. A responds to B, assuring B of his presence.

Denning and Sacco [19] showed that the protocol is flawed. The attack is because there is nothing in the message three to indicate that it was actually created by S as a part of the current session of the protocol. The attack is described below:

$$
\begin{array}{lll}
\beta.1 \quad A \rightarrow S & : & A, B, x \\
\beta.2 \quad S \rightarrow A & : & \{x, B, K, \{K, A\}_{K_{BS}}\}_{K_{AS}} \\
\beta.3 \quad A \rightarrow I_B & : & \{K, A\}_{K_{BS}} \\
\alpha.3 \quad I_A \rightarrow B & : & \{K', A\}_{K_{BS}} \\
\beta.4 \quad B \rightarrow I_A & : & \{y\}_{K'} \\
\beta.5 \quad I_A \rightarrow B & : & \{y-1\}_{K'}
\end{array}
$$

We assume that in a previous session α of the protocol, the agents A and B shared a key K' and that the key K' has been compromised (for example by cryptanalysis). In the current session β of the protocol, the agent A sends a new request for a key to communicate with B. The intruder intercepts the current certificate for B and replaces it with an old certificate that contains K'. As we stated above, B has no way of knowing that this is an old certificate. Because the intruder knows K' he sends the message that B is expecting in the fifth step of the current session. B is fooled thinking that K' is the current key shared with A. Therefore, using the key K' the intruder can engage in a further communication with B masqueradeing as A. The above attack emphasizes the need of an information to indicate the freshness for the third message in the protocol. This can be done either adding a nonce, or a timestamp in the message. The Needham-Schroeder secret-key protocol is a base for the Kerberos protocol that eliminates this kind of attack by including timestamps.

A different attack strategy is a *type flaw attack*: a field that was originally intended to have one type is interpreted as having another type. To illustrate this

type of attack, we will present the Π^1 Woo and Lam authentication protocol [70]:

$$\begin{array}{lcl} 1. & A \rightarrow B & : \quad A \\ 2. & B \rightarrow A & : \quad x \\ 3. & A \rightarrow B & : \quad \{A, B, x\}_{K_{AS}} \\ 4. & B \rightarrow S & : \quad \{A, B, \{A, B, x\}_{K_{AS}}\}_{K_{BS}} \\ 5. & S \rightarrow B & : \quad \{A, B, x\}_{K_{BS}} \end{array}$$

Initially, the agent A knows only the long-term key K_{AS} shared by A and the server S, and the name of the agent B, while B only knows the long-term key K_{BS} shared by B and S. Woo and Lam stated that the protocol is correct if whenever the responder (B) finishes the execution of the protocol, the initiator of the protocol is in fact the agent A claimed in the initial message .

As we can see, B cannot decrypt the message in the third communication step, but simply includes it in the fourth step to the server. The following type flaw attack exploits this:

$$\begin{array}{lcl} 1. & I_A \rightarrow B & : \quad A \\ 2. & B \rightarrow I_A & : \quad x \\ 3. & I_A \rightarrow B & : \quad x \\ 4. & B \rightarrow I_S & : \quad \{A, B, x\}_{K_{BS}} \\ 5. & I_S \rightarrow B & : \quad \{A, B, x\}_{K_{BS}} \end{array}$$

The intruder impersonate both A and S; it replays the nonce x at step 3, which B accepts it and thinks it is of the form $\{A, B, x\}_{K_{AS}}$. Therefore, B sends back $\{A, B, x\}_{K_{BS}}$, which is precisely the form of message that the intruder requires to fake the message at step 5. So, B is fooled thinking that he ended a protocol session with A, whereas A did'nt participated in this session of the protocol. A solution for this kind of attack is given by tagging scheme: for example, by adding some extra information to each field of the message indicating their intended type. Some tagging scheme will be discussed later in this thesis.

Difficulties in modelling and analyzing security protocols Standard notation it is not sufficient to fully describe the behaviour of a protocol. We will need a formal model in which we express the protocol, the desired security goals and the environment in which the protocol take place. Then, we need to apply a suitable technique to prove that the resulting model has desirable security properties. The difficulty of modelling and analyzing security protocols relies on the following considerations:

- Despite its simplicity, security protocols deploys in an hostile environment. To evaluate them properly, we need to describe and model the capabilities of the agents deliberately trying to undermine the protocol;

- The protocols are specified in terms of abstract agent names A, B, S etc., abstract nonces x, y, etc., and abstract keys K_{AS}, K_{BS} etc. In a run, abstract names are instantiated with concrete principals, nonces, keys, and other message terms. Many such instantiations may be interleaved in a single run (see Lowe's attack);

- An agent can be involved simultaneously in many instantiations of one or more roles of the protocol;

- There is no bound on the number of run's sessions. As Denning and Sacco's attack illustrates, many protocols need fresh nonces in each session that leads to unbounded number of nonces in a run;

- The intruder might force the principals to generate unboundedly long messages (as in type flaw attacks);

Due to this considerations, the set of runs of the protocol is infinite, although the protocol's specification is finite. Moreover, a security protocol satisfies a desired property if all runs of the protocol satisfy it.

Formal methods for security protocols analysis The attacks illustrated above show the need for some guidelining principles in design of the protocols and the need of formal analysis methods. In [2] some principles to design the protocols are stated, but they are not sufficent for correctness.

As Needham and Schroeder pointed out in [48], "security protocols are prone to extremely subtle errors that are unlikely to be detected in normal operation". The formal methods for security protocols analysis focuses on two complementary directions: one is to prove the correctness of protocols and the other is to find flaws in the protocols.

Much work has been done to develop methods to ensure the protocols correctness, starting with Dolev and Yao [22] that proposed a model for adversary capabilities adopted by most succeding models. The seminal work of Burrows, Abadi, and Needham who developed the BAN authentication logic [9] was one of the first attempts to make reasoning about security protocols more systematic. Meadows [42] developed the NRL Protocol Analyzer, a special-purpose verification tool for the analysis of security protocols. Paulson [50] introduced a verification technique for security protocols based on inductive proofs with automated support provided by Isabelle/HOL which works with higher-order logic. The CSP approach can also be used to construct models of security protocols, which consists of a number of communicating components and are thus well suited to analysis [58]. In [1] protocols are represented as processes in spi-calculus and their security properties can be expressed as observational equivalence of two processes. Durgin and

Mitchell [23] introduced a formalism based on multiset rewriting to model and analyze security protocols. The strand spaces approach [66] associates strands to model the behavior of the participants. Ramanujam and Suresh [52] proposed a formalism, that will be used with slight modifications in this thesis.

Another important direction in analyzing security protocols is focusing on finding the flaws in the protocols, using model checking approach. The basis of this approach is to produce a finite state system of a protocol and to use a state exploration tool (model-checker) to discover if the system reaches an insecure state. Model-checkers can be used to find attacks, but in case no attack is detected, this only means that there is no attack on the finite state system; there may be an attack on some larger system runing the same protocol. Model checking based systems include tools such as Lowe's FDR [39], Murϕ [46] and Brutus [14].

The abstraction techniques have also been succesfuly used to analyze security protocols. This approach imposes a bounded number of the protocol's sessions and uses a finite representation of the unbounded capabilities of the intruder. An early work is that in [8] which uses an abstract interpretation to define a finite system from a protocol. In [47, 32] tree automata techniques have been proposed to efficiently represent the intruder's state.

Contributions of the thesis

One of the main question one may ask when dealing with security protocols is the following one: How difficult is to prove that a security protocol assures secrecy? The work from this thesis give some answers to this question, which is a central problem for a couple of years and, in the meantime, several results have been developed.

In Chapter 2, we will describe the formalism used to develop the decidability and complexity results from this thesis. We distinguish between the *initial secrecy problem* (where the secrets are given from the beginning), mostly considered in the literature on security protocols and the *secrecy problem*, where the secrets are generated in the course of runs. For instance, in [57, 25], the authors say that, in order to study the complexity of the secrecy problem, one may consider that the secrets can be only provided from the beginning.

Secrecy for security protocols is in general undecidable. Many proofs have been proposed to this, based on the halting problem for Turing/counter machines, Post's correspondence problem, the implication problem for existential Horn clauses, the reachability problem for reset Petri nets, etc. In Chapter 3 we will provide undecidability results for the secrecy problem using reductions from the halting problem for two-counter machine and Post's correspondence problem. Two cases are taken into consideration: infinitely many nonces but bounded-length messages, and finitely many nonces but arbitrary-length messages. We identify

an error in the reduction from Post's correspondence problem shown in [25] and correct it.

By imposing different bounds on security protocols, we identify some subclasses of protocols for which secrecy problem became decidable. The first subclass of such protocols is that of *bounded protocols*, where both message length and the number of nonces are bounded. The complexity of the secrecy problem for bounded protocols has been the subject of several papers, starting with [24]. Thus, using a multiset rewriting formalism, it has been shown in [24] that the initial secrecy problem for protocols in the restricted form, with bounded length messages, no existentials, and an intruder with no existentials is DEXPTIME-complete.

Using a formalism and a terminology slightly different than the one in [24], we show that the initial secrecy problem for bounded protocols whithout freshness check is DEXPTIME-complete (Section 4.1.1). The main differences between the proof in [24] and the one in this thesis are:

- precise bounds proving the membership to DEXPTIME are provided;
- DEXPTIME-hardness is showed by exhibiting a simpler reduction than the one in [24].

We also point out the main difficulty which appears when one wants to extend this result to the secrecy problem for bounded protocols without freshness check.

Using a multiset rewriting formalism, it has been claimed in [25] (Table 9, page 282) that the following problems are DEXPTIME-complete:

- the initial secrecy problem for protocols in the restricted form, with bounded length messages, bounded existentials, without disequality tests, and an intruder with no existentials;
- the initial secrecy problem for protocols in the restricted form, with bounded length messages, bounded existentials, disequality tests, and an intruder with no existentials;
- the initial secrecy problem for protocols in the restricted form, with bounded length messages, bounded existentials, without disequality tests, and an intruder with existentials.

Unfortunately, all these statements are wrong because allowing bounded existentials with honest agents requires much more effort to decide the initial secrecy problem, making it complete for NEXPTIME. In Section 4.1.2, under the formalism used in thesis, we obtain the main result: the (initial) secrecy problem for bounded protocols with freshness check is NEXPTIME-complete. To prove NEXPTIME-hardness we provide a new style of reduction from Turing machine.

Relating the formalism in this thesis with the multiset rewriting formalism we obtain that the three problems mentioned above are NEXPTIME-complete (Section 4.1.4). These results also holds for the protocols in unrestricted form. Moreover, the initial secrecy problem for protocols in the restricted or unrestricted form, with bounded length messages, bounded existentials, disequality tests, and an intruder with bounded existentials, is NEXPTIME-complete too. If "an intruder with bounded existentials" is replaced by "an intruder with existentials" and the protocols are in the unrestricted form, this last problem becomes undecidable. Therefore, the problem mention in Table 9 of [25] still remains open.

Another restriction that leads to decidability is the bound on the number of sessions that can occur in any run of the protocol as in [57, 43, 3, 37, 6, 27]. In [57, 3] several NP-completeness results for classes of protocols under bounded number of sessions were obtained. This results holds also for bounded protocols. In these papers, the authors implicitly assume that secrets can only be provided from the beginning. Thus, the results obtained in [57, 3] hold true for the initial secrecy problem. However, trying to reduce the secrecy problem to the initial secrecy problem in deterministic polynomial time has proved to be quite hard. We will provide the secrecy problem with similar complexity results as the initial secrecy problem for the case of bounded protocols under bounded number of sessions (Section 4.1.3). We prove NP-hardness by exhibiting a reduction from 3-SAT with major differences from the one previously shown in [57]:

- our reduction works for secrets generated in the course of runs and not only for initial secrets;

- in [57] the protocol is considered leaky only if some "big secrets" are revealed, while we treat all secrets equally.

All the results mentioned above have been developed under different formalisms which are not necessarily equivalent (each of them has specific constructs that do not have faithful equivalents in the others). By presenting all the results under the same formalism, together with the existing results, this thesis provides a complete picture of the complexity of the initial secrecy problem and the secrecy problem for bounded protocols.

Decidability of secrecy also can be obtained imposing restriction on the way messages can be constructed [3, 21, 17].

Using semantic criteria, a new subclass of protocols for which the secrecy problem is decidable is identified: *normal protocols*. The subclass of normal protocols in [53] is based on a very interesting equivalence relation on terms but, unfortunately, the results are seriously flawed. In Section 4.2 we show decidability of secrecy for normal protocols, correcting the flawed statements from [53].

The subclass of *tagged protocols* [55] is the last one considered in the thesis, for which decidability of secrecy under infinitely many nonces and arbitrary-length messages is obtained.

The work in this thesis is based on the papers [67, 68, 69].

Chapter 2

Modelling Security Protocols

2.1 Background

Many models for security protocols have been proposed in the literature. An early model, described by Dolev and Yao in [22], proposes an intruder with the ability to manipulate messages passing over the system by deleting, replaying, facking, and so on. This assumptions about intruder's capabilities are used by most models for security protocols and are commonly refered as "Dolev-Yao model". Before presenting the model used in this thesis, we will briefly described the most popular models proposed for security protocols.

BAN logic The BAN logic [9] focuses on the beliefs of the legitimate agents involved in the protocol and on the way this beliefs evolves as new informations are received. Any protocol step is transformed into a logic formulae by idealization. For example, the second step of the Needham-Schroeder secret-key protocol is idealized as follows:

$$S \rightarrow A : \{x, A \overset{K}{\longleftrightarrow} B, fresh(A \overset{K}{\longleftrightarrow} B), \{A \overset{K}{\longleftrightarrow} B\}_{K_{BS}}\}_{K_{AS}}$$

, where the formulae $A \overset{K}{\longleftrightarrow} B$ means that the agents A and B may use the key K to communicate, and $fresh(A \overset{K}{\longleftrightarrow} B)$ that the formulae $A \overset{K}{\longleftrightarrow} B$ is fresh. The idealization step eliminates communications with unencrypted components. The goals of protocols are also translated into logical formulae. A protocol is correct if the protocol's goals can be achieved by applying the inference rules to the formulae representing the initial assumptions and the protocol messages.

The BAN logic is simple and easy to apply. Using BAN logic, flaws in some security protocols were discovered (for example in the CCITT X.509 protocol [11]), but it didn't discovered the errors in some known flawed protocols from literature (see Lowe's attack on the Needham-Schroeder public-key protocol). A

weak point in BAN logic is that it does not attempt to model knowledge, it can't be used to prove results about secrecy and can be used only to reason about authentication. This weakness of BAN is exploited by Nesset [49] that provides an protocol example that is obviously insecure but the flaw is not detected by BAN analysis. An early semantics for BAN logic is due to Abadi and Tuttle [4]. To improve the BAN logic and its semantics some extensions were developed: GNY and SVO logics [31, 65]. Automated tools have been provided for BAN and derivated logics (for example in [10] a HOL-based tool).

The inductive approach In this approach, a protocol is formalized as the set of all possible traces, where a trace is a sequence of events that occur in some run of the protocol. The events are of the form: *Says* $A\ B\ X$ that means that A sends X to B and *Notes* $A\ X$ that A learns X. Bella [5] extends the original approach to explicitely model the reception of a message, introducing the event *Gets* $A\ X$ representing the fact that A receives X. Traces are defined inductively, starting from empty trace, and adding a set of rules that corresponds to the possible actions of the agents (including the intruder).

A protocol is specified in Isabelle/HOL by a set of rules, each protocol step having a corresponding rule. For example, the first two steps of the Needham-Schroeder public-key protocol are specified as follows:

NS1 $(t_1 \in ns \ \wedge\ A \neq B \ \wedge\ nonce(x) \notin used(t_1)) \Rightarrow$
$Says\ A\ B\ crypt(pubk(B), \{nonce(x), agent(A)\})\ \#\ t_1 \in ns$

NS2 $(t_2 \in ns \ \wedge\ A \neq B \ \wedge\ nonce(y) \notin used(t_2) \ \wedge$
$Gets\ B\ crypt(pubk(B), \{nonce(x), agent(A)\}) \in set(t_2)) \Rightarrow$
$Says\ B\ A\ crypt(pubk(A), \{nonce(x), nonce(y)\})\ \#\ t_2 \in ns$

In Isabelle/HOL datatypes are introduced : *agent, pubk, nonce,* etc, to define agents, public keys, nonces, respectively. The operators on traces are also used:*set* that returns the set of events of a trace and *used* to verify whether a data is fresh. ns is the set of protocol's traces and $e\#tr$ represents that the event e extends the trace tr. The precondition of the **NS1** rule specifies that t_1 is the current trace of the protocol, A and B are two distinct agents and the nonce x is fresh. The conclusion of the rule is that the trace t_1 can be extended with an event that corresponds to the first step of the protocol. Similarely, the rule **NS2** defines the second protocol step: its precondition requires that the first event to be already in the trace. As we observe, sending a message depends on receiving a corresponding one. A important point of Paulson's approach is that all possible actions of the intruder are taking in consideration by the **Fake** rule.

Security properties such as secrecy and authentication can be stated as predicates over the trace in a manner similar to the CSP approach [58]. For example, secrecy for the initiator in the Needham-Schroeder public-key protocol, states that for any trace tr of the protocol, the nonce x is not in the set $analz(spies(tr))$ of knowledge deduced by the intruder from the trace tr. A protocol satisfies a security property if all its traces are satisfying that property. This verification is done inductively using the Isabelle theorem prover. If a property fails to be prove, then the machine proof leads to an attack scenario.

Strand spaces The strand space model has been introduced by Thayer, Herzog and Guttman [66]. A strand is a sequence of events; it represents either the actions that an honest agent can realize in a protocol role (regular strand), or an atomic action of the intruder (penetrator strand). For example, in the Needham-Schroeder-Lowe protocol, the initiator strand is:

$$\langle +\{x, A\}_{K_B}, -\{x, y, B\}_{K_A}, +\{y\}_{K_B} \rangle$$

, where $+, -$ signifies the transmition, respectively the reception of a term. More complex actions of the intruder are modeled conecting a number of penetrator strands.

A node is an element of a strand (for example, the initiator strand above has three nodes). Two kinds of relations are defined between nodes: $\rightarrow$ that represents the communication between strands and $\Rightarrow$ that connects succesive nodes of a strand. A strand space associated to a protocol is a graph whose nodes are all the nodes of the protocol and the edges are defined by that two kinds of relations above.

The main concept of this approach is the *bundle*: a set of nodes causally conected; that is whenever a node is in a bundle, its causal past is also in bundle and whenever a bundle contains a reception node, then it contains the corresponding sending node. The analysis of security protocols properties is carry out on bundles. For example, a protocol preserves secrecy if for all nodes from any bundle, a secret is not revealed to the intruder. The significant feature of this model is that the runs of a protocol are formalized as partially ordered sets, unlike the most other models for security protocols [23, 50, 52, 39]. A tool for automatic analysis based on the strand space model, called Athena, has been developed [60]. Strand spaces can also be used to provide semantics for BAN logic [64]. A connection between strand space model and multiset rewriting formalism ([23]) was established in [13].

Process algebra models There are mainly two kinds of process algebra models for security protocols: based on CSP model and spi calculus model. CSP

(Communicating Sequential Processes) is a notation to describe systems of parallel agents that communicate by passing messages between them and was first introduced by Hoare [34]. There are many approaches based on CSP [39, 41, 58]. CSP is provided with an automated tool: the model-checker FDR (Failures Divergence Refinement) [29]. To model security protocols using CSP, Lowe [39] defines processes for each agent, including the intruder. The processes are described in terms of the possible events that they may engage in. For example, the initiator role in the Needham-Schroeder public-key protocol is modeled by the following process:

$$
\begin{aligned}
&INITIATOR(A,x)\widehat{=}\\
&\quad user.A?B \rightarrow I\_running.A.B \rightarrow\\
&\quad comm!Msg1.A.B.Encrypt.key(B).x.A \rightarrow\\
&\quad comm.Msg2.B.A.Encrypt.key(A)?x'.y \rightarrow\\
&\quad \text{if} \quad x = x'\\
&\quad \text{then} \quad comm!Msg3.A.B.Encrypt.key(B).y \rightarrow\\
&\qquad\quad I\_commit.A.B \rightarrow session.A.B \rightarrow Skip\\
&\quad \text{else} \quad Stop.
\end{aligned}
$$

A request from the user for the initiator A to connect with the responder B is represented by the event $user.A?B$ and the resulting session by the event $session.A.B$. The event $comm!Msg1.A.B.Encrypt.key(B).x.A$ models sending the first message in the protocol, where $comm$ is the standard communication channel. The reception of the second message by the initiator is represented by the event $comm.Msg2.B.A.Encrypt.key(A)?x'.y$. The initiator checks the value of the received nonce and if it is matching with the one he sended in the first message, then commits to the session, and carries out the session. A similar process is used for the responder role. The intruder's capabilities are described by a process that can perform any attack. The process corresponding to the protocol is defined as a parallel composition of the processes for the roles and the intruder process.

Model checking of the whole system is performed automatically by comparing the CSP specification of a protocol with another specification that captures the desirable properties only. For example, the authentication of the initiator is described using the events $I\_running.A.B$ and $R\_commit.A.B$:

$$
\begin{aligned}
AI\widehat{=}&I\_running.A.B \rightarrow\\
&R\_commit.A.B \rightarrow AI
\end{aligned}
$$

that states that the responder B commits to a session with the initiator A only if A has participated in the session of the protocol. Then, the process that captures the authentication for the initiator is obtain by parallel composition of the process AI and the process corresponding to the protocol. Similarely, for secrecy, an event

$Claim\_secret$ will be inserted in a point of a process role where we believe that the intruder can not obtain a secret value.

Abadi and Gordon [1] introduced spi-calculus: an extension of π-calculus [45] designed to deal with cryptographic primitives. In spi-calculus a process is associated to each role in the protocol, but unlike CSP, there is no need to explicitely define an intruder process. Security properties can be expressed in terms of the observational equivalence between processes. Two processes P and Q are observationally equivalent if a third process R can not distinguish running in parallel with P from running in parallel with Q. The protocol preserves secrecy if for a secret x, the protocol running with x is observationally equivalent with the protocol running with x', for every x'. That means that if the secret is not revealed, an arbitrarely intruder can't see any diference between a run using x and a run using x'. For example, $\{x\}_K$ and $\{x'\}_K$ are not distinguishable to an intruder that does'nt know K. The authentication is given as an equivalence between a process that describes the actual protocol and a process that describes a version "obviously correct" of the protocol.

Multiset rewriting Multiset rewriting (MSR) formalism for security protocols was introduced in [23]. Decidability and complexity results for security protocols using this framework were developed in [12, 24, 25]. A detailed description of MSR formalism and relations with the formalism used in the thesis will be given in Section 4.1.4.

2.2 A model for security protocols

In what follows we will adopt the model proposed in [52] with slight modifications, and we will use it in order to discuss the decidability and complexity of the secrecy problem for various classes of security protocols. The model used in the thesis is close to the inductive approach from [50] in some respects.

Protocol signature and terms

A *protocol signature* consists of three sets $\mathcal{S} = (\mathcal{A}, \mathcal{K}, \mathcal{N})$, where $\mathcal{A}$ is a finite set of *agent names* (or shortly, *agents*), $\mathcal{K}$ is an at most countable set of *keys*, and $\mathcal{N}$ is an at most countable set of *nonces*. It is assumed that:

- $\mathcal{A}$ contains a special element denoted by I and called the *intruder*. All the other elements are called *honest agents* and we denote by Ho their set. We consider the *Dolev-Yao model of the intruder*, stated in [22], that assumes an all-powerful intruder who:

  - can read any message and block further transmission;
  - can impersonate any honest agent;
  - can decompose messages into parts and remember them, including decrypting any message for which the intruder has obtained the key;
  - can generate fresh data as needed, and
  - can compose new messages from known data and send.

  However, the intruder cannot generate the honest agents' secrets, nor it can break encryption. Even if we consider a group of Dolev-Yao intruders colluding with one another, the power of the intruder is not increasead. As it has been pointed out in [63], any group of Dolev-Yao intruders colluding with one another cannot cause more attacks than a single intruder acting alone;

- $\mathcal{K} = \mathcal{K}_0 \cup \mathcal{K}_1$, where $\mathcal{K}_0$ is a set of *short-term keys* and $\mathcal{K}_1$ is a finite set of *long-term keys* shared by agents. K_{AB} will denote the long-term key shared by two distinct agents A and B and we assume that any agent A knows initially a set of long-term keys $\mathcal{K}_A$ (i.e., keys shared by A with other participants, *public keys* and A's *private key*);

- $\mathcal{N}$ is a set of nonces that are unguessable randomly generated numbers. As we will see later, depending on the requirements of the computation rule, we will test the freshness of the generated nonces by honest agents.

- some honest agents are provided from the beginning with some *secret information*, not known to the intruder, and given as nonces or short-term keys. Therefore, we denote by $Secret_A$ the set of secret information the agent A is provided from the beginning. $Secret_A$ does not contain long-term keys because such keys will be used only for encryptions and never communicated by agents during the runs;

- the intruder is provided from the beginning with a set of nonces $\mathcal{N}_I \subseteq \mathcal{N}$ and a set of short-term keys $\mathcal{K}_{0,I} \subseteq \mathcal{K}_0$. This sets play the role of a set of new data the intruder may generate in the course of a run, and as we will develop the model, we ensure that they cannot be generated by any honest agent. Unlike other models (for example multiset rewriting [23]), we do not explicitely model new data generation of the intruder.

The set of *basic terms* is $\mathcal{T}_0 = \mathcal{A} \cup \mathcal{K} \cup \mathcal{N}$. The set $\mathcal{T}$ of *terms* is defined inductively by:

- every basic term is a term;
- if t_1 and t_2 are terms, then (t_1, t_2) is a term;
- if t is a term and K is a key, then $\{t\}_K$ is a term.

The notation (t_1, t_2) denotes the term obtained by *pairing* of the terms t_1 and t_2. We extend the construct (t_1, t_2) to $(t_1, \ldots, t_n)$ as usual by letting

$$(t_1, \ldots, t_n) = ((t_1, \ldots, t_{n-1}), t_n),$$

for all $n \geq 3$. Sometimes, parenthesis will be omitted. The term $\{t\}_K$ is a abstract notation where we make no cryptographic assumptions about the algorithm used to form $\{t\}_K$ from t and K. We will allow only terms constructed using atomic keys and we will disallow encryption with *composed keys* (non-atomic keys) that can be formed with pairing or encryption of terms. In same spirit with [22], we will assume that the terms came from a free algebra of terms with pairing and encryption operators.

Given a term t, the set of all *subterms* of t, denoted by $Sub(t)$, is defined to be the least set of terms that satisfies the following properties:

- $t \in Sub(t)$;
- if $(t_1, t_2) \in Sub(t)$ then $t_1, t_2 \in Sub(t)$;
- if $\{t_1\}_K \in Sub(t)$ then $t_1, K \in Sub(t)$.

$ESub(t)$ denotes the set of all *encrypted subterms* of t. That is,

$$ESub(t) = \{t' \in Sub(t) | (\exists t'' \in \mathcal{T})(\exists K \in \mathcal{K})(t' = \{t''\}_K)\}.$$

These two notations are extended to sets of terms by union.

The *length* of terms is inductively defined, by taking into consideration that pairing and encryption are operations:

- $|t| = 1$, for any $t \in \mathcal{T}_0$;
- $|(t_1, t_2)| = |t_1| + |t_2| + 1$, for any terms t_1 and t_2;
- $|\{t\}_K| = |t| + 2$, for any term t and key K.

We adopt the *perfect encryption assumption* [22] that states that a message encrypted with a key K can be decrypted only by an agent who knows the corresponding inverse of K [1], and the only way to compute $\{t\}_K$ is by encrypting t with K.

[1]The inverse of a key K, denoted by K^{-1} is defined by: if K is a symmetric key then $K^{-1} = K$ and, if $(K, \overline{K})$ is an asymmetric key then $K^{-1} = \overline{K}$.

Actions

Security protocols are usually specified as sequences of communications of the form $A \rightarrow B : t$. Such communications say that A sends t to B, and B receives t from A.

For analysis of security protocols, a third party should be taken into consideration, namely the intruder. In such a case, a communication $A \rightarrow B : t$ says that A sends t to B, but it is not guaranteed that B will receive it (B may receive a different message composed by the intruder). From this point of view, a decomposition of the communication $A \rightarrow B : t$ into two actions, has to be taken into consideration:

- a *send action*, performed by A and denoted $A!B : t$, where B is the intended receiver, and
- a *receive action*, performed by B and denoted $B?A : t$, where A is the supposely sender.

From a technical point of view, it is useful to specify, along with any send action, the set of nonces and short-term keys freshly generated by A to compose t. Therefore, a *send action* will be of the form $A!B : (M)t$, while a *receive action* will be of the form $A?B : t$. In both cases,

- $A \in Ho$,
- $B \in \mathcal{A}$,
- $A \neq B$,
- $t \in \mathcal{T}$ is the *term of the action*, and
- $M \subseteq Sub(t) \cap (\mathcal{N} \cup \mathcal{K}_0)$ is the *set of new terms of the action* [2].

Note that, in this formalism, we do not explicitly model the intruder's actions. As we will see from the computation rule, any message sent by an honest agent is intercepted by the intruder and any message received by an honest agent comes from the intruder.

Given an action a, we shall denote by $M(a)$ the set

$$M(a) = \begin{cases} M, & \text{if } a = A!B : (M)t \\ \emptyset, & \text{if } a = A?B : t. \end{cases}$$

Moreover, $t(a)$ stands for the term of a. We will simply write $A!B : t$ whenever $M = \emptyset$.

[2] The terminology will be clear when the computation rule is discussed.

Denote by Act the set of all actions, and by $Act(A)$ the set of all actions performed by A, that is, actions where A is involved as a sender or as a receiver:

$$Act(A) = \{A!B : (M)t | B \in \mathcal{A}\} \cup \{A?B : t | B \in \mathcal{A}\}.$$

For a sequence of actions $w = a_1 \cdots a_l$ and an agent A, we define the *restriction of w to A* as being the sequence obtained from w by removing all actions not in $Act(A)$. Denote this sequence by $w|_A$. The notations $M(a)$ and $t(a)$ are extended to sequences of actions by union.

Protocols

Definition 2.2.1 A *protocol* is a triple $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$, where

- $\mathcal{S}$ is a protocol signature;
- $\mathcal{C}$ is a subset of $\mathcal{T}_0$, called the set of *constants* of $\mathcal{P}$;
- w is a non-empty sequence of actions, called the *body* of the protocol.

Constants are publicly known elements in the protocol that cannot be re-instantiated (as it will be explained a little bit later). As usual, $\mathcal{C}$ does not include private keys, elements in $Secret_A$ for any honest agent A, or elements in $\mathcal{N}_I$, $\mathcal{K}_{0,I}$ and $M(w)$.

Any non-empty sequence $w|_A$, where A is an honest agent, is called a *role* of the protocol. A role specifies the actions which an agent should perform in a protocol, and the order of these actions. We denote by *Roles($\mathcal{P}$)* the set of all roles of the protocol $\mathcal{P}$.

Example 2.2.1 Using this model, the Needham-Schroeder public-key protocol presented in Chapter 1, is given by $\mathcal{P}_{NS} = (\mathcal{S}, \mathcal{C}, w)$, where:

- $\mathcal{S}$ consists of two honest agents A and B, their public and private keys $(K_A, \overline{K}_A)$ and $(K_B, \overline{K}_B)$, and two nonces x and y;
- $\mathcal{C} = \emptyset$;
- w is the following sequence:

$$\begin{array}{lcl} A!B & : & (\{x\})\{x, A\}_{K_B} \\ B?A & : & \{x, A\}_{K_B} \\ B!A & : & (\{y\})\{x, y\}_{K_A} \\ A?B & : & \{x, y\}_{K_A} \\ A!B & : & \{y\}_{K_B} \\ B?A & : & \{y\}_{K_B} \end{array}$$

$Roles(\mathcal{P}_{NS}) = \{r_1, r_2\}$, where:

- $r_1 = w|_A$ is the *initiator role* given by:

$$\begin{array}{lcl} A!B & : & (\{x\})\{x, A\}_{K_B} \\ A?B & : & \{x, y\}_{K_A} \\ A!B & : & \{y\}_{K_B} \end{array}$$

- $r_2 = w|_B$ is the *responder role* given by:

$$\begin{array}{lcl} B?A & : & \{x, A\}_{K_B} \\ B!A & : & (\{y\})\{x, y\}_{K_A} \\ B?A & : & \{y\}_{K_B} \end{array}$$

As a general remark, all protocol specifications will implicitly omit I and the elements associated to it, and $Secret_A$ will be the empty set whenever it is not explicitly specified.

Substitutions and events

A security protocol specifies the legal actions to be performed between generic participants in the protocol. The application of the protocol in various real cases is done by instantiating the protocol. Moreover, in a real environment, multiple instantiations of the same protocol may run.

Instantiations of a protocol are given by *substitutions*, which are functions which map agents to agents, short-term keys to short-term keys, and nonces to arbitrary terms. A substitution which maps nonces to nonces is called a *well-typed substitution*.

Substitutions are homomorphically extended to terms, actions, and sequences of actions. For instance, if σ is a substitution and $t = \{t'\}_{K_{AB}}$ is a term, then $\sigma(t) = \{\sigma(t')\}_{K_{\sigma(A)\sigma(B)}}$.

Definition 2.2.2 A substitution σ is called:

- *suitable for an action* $a = AxB : y$ if
  - $\sigma(A)$ is an honest agent, that means that each action is instantiated into a legitimate action;
  - $\sigma(A) \neq \sigma(B)$;
  - σ maps distinct nonces from $M(a)$ into distinct nonces, distinct short-term keys into distinct short-term keys;

- *suitable for a sequence of actions* if it is suitable for each action in the sequence;

- *suitable for a subset* $C \subseteq \mathcal{T}_0$ if it is the identity on C.

Example 2.2.2 Two well-typed substitutions σ_1 and σ_2, suitable for the role r_1, respectively r_2 of the protocol $\mathcal{P}_{NS}$ presented in Example 2.2.1 are illustrated:

- $\sigma_1(A) = A$, $\sigma_1(B) = I$, $\sigma_1(x) = m$, $\sigma_1(y) = n$, and

- $\sigma_2(A) = A$, $\sigma_2(B) = B$, $\sigma_2(x) = m$, $\sigma_2(y) = n$.

Remark 2.2.1 Using suitable substitutions, the model allows only to the intruder to map nonces to arbitrary terms, leaving the possibility for type-flaw attacks.

Definition 2.2.3 An *event* of a protocol $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$ is any triple $e_i = (r, \sigma, i)$, where:

- $r = a_1 \cdots a_l$ is a role of $\mathcal{P}$,

- σ is a substitution suitable for r and $\mathcal{C}$, and

- $1 \leq i \leq l$.

$\sigma(a_i)$ is the *action of the event* e_i. As usual, $act(e_i)$ ($t(e_i)$, $M(e_i)$) stands for the action of e_i (term of e_i, set of new terms of e_i). The notations above are extended to sequences of events by union. The event $e_i = (r, \sigma, i)$ is called *well-typed* if the substitution σ is well-typed.

The *local precedence relation* on events is defined by

$$(r, \sigma, i) \rightarrow (r', \sigma', i')$$

if and only if $r' = r$, $\sigma' = \sigma$, and $i' = i + 1$, provided that $i < |r|$. $\stackrel{+}{\rightarrow}$ is the transitive closure of $\rightarrow$. Given an event e, $LP(e)$ stands for the *set of all local predecessors of* e, i.e.,

$$LP(e) = \{e' | e' \stackrel{+}{\rightarrow} e\}.$$

For any sequence of events ξ, we shall denote by $Events(\xi)$ the set of all events of ξ.

Message generation rules

In [50], Paulson introduced the *analz* and *synth* operators to model deducing, respectively constructing new messages from old ones. For a given set X of terms denote by $analz(X)$ the least set of terms that satisfies the following properties:

1. $X \subseteq analz(X)$;
2. if $(t_1, t_2) \in analz(X)$, then $t_1, t_2 \in analz(X)$; *(decomposition)*
3. if $\{t\}_K, K^{-1} \in analz(X)$, then $t \in analz(X)$; *(decryption)*
4. if $\{\{t\}_K\}_{K^{-1}} \in analz(X)$, then $t \in analz(X)$. (*simplification)*

By the *simplification* rule, we identify $\{\{t\}_K\}_{K^{-1}}$ with t. The terms in $analz(X)$ are obtained from those in X by decomposition and decryption with known keys.

Example 2.2.3 Let $X = \{\{(\{m\}_{K'}, n)\}_K, K^{-1}\}$ where $K^{-1} \neq K'^{-1}$. Then $n \in analz(X)$ because $K^{-1} \in analz(X)$, but $m \notin analz(X)$ because $K'^{-1} \notin analz(X)$.

For a given set X of terms denote by $synth(X)$ the least set of terms that satisfies the following properties:

1. $X \subseteq synth(X)$;
2. if $t_1, t_2 \in synth(X)$, then $(t_1, t_2) \in synth(X)$; *(pairing)*
3. if $t, K \in synth(X)$, then $\{t\}_K \in synth(X)$. *(encryption)*

The terms in $synth(X)$ are obtained from those in X by pairing and encryption with known keys.

Example 2.2.4 Let $X = \{\{m\}_K, n, \{K\}_{K'}, K\}$, where $K \neq K'$.
Then $\{(\{m\}_K, n)\}_K \in synth(X)$, but $\{(\{m\}_K, n)\}_{K'} \notin synth(X)$ because $K' \notin synth(X)$.

Moreover, $\overline{X}$ stands for $synth(analz(X))$. The following proposition states the basic properties of these operators.

Proposition 2.2.1 Let X, X' the sets of terms and σ be a substitution. Then, the following properties hold true:

(1) $X \subseteq analz(X) \cap synth(X)$;

(2) If $X \subseteq X'$ then $analz(X) \subseteq analz(X')$ and $synth(X) \subseteq synth(X')$;

(3) $analz(analz(X)) = analz(X)$ and $synth(synth(X)) = synth(X)$;

(4) $\overline{\overline{X}} = analz(\overline{X}) = \overline{X}$;

(5) $\sigma(analz(X)) \subseteq analz(\sigma(X))$;

(6) $\sigma(synth(X)) \subseteq synth(\sigma(X))$;

(7) $\sigma(\overline{X}) \subseteq \overline{\sigma(X)}$.

States and runs

A *state* of a protocol gives information about the knowledge of all participants at a given moment in the evolution of the protocol. Therefore, it is defined as an indexed set $s = (s_A | A \in \mathcal{A})$, where $s_A \subseteq \mathcal{T}$, for any agent A. By $Sub(s)$ we will denote the set $Sub(s) = Sub(\bigcup_{A \in \mathcal{A}} s_A)$.

The *initial state* of a protocol $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$ is $s_0 = (s_{0A} | A \in \mathcal{A})$, where:

- $s_{0A} = \mathcal{A} \cup \mathcal{C} \cup \mathcal{K}_A \cup Secret_A$, for any honest agent A;
- $s_{0I} = \mathcal{A} \cup \mathcal{C} \cup \mathcal{K}_I \cup \mathcal{N}_I \cup \mathcal{K}_{0,I}$.

Actions in a protocol define a transition relation between states of the protocol as follows. Given two states s and s' of $\mathcal{P}$ and an action a, we write $s[a\rangle s'$ if and only if:

1. if a is of the form $A!B : (M)t$, then:
   (a) $t \in \overline{s_A \cup M}$ and $M \cap Sub(s) = \emptyset$; (*enabling condition*)
   (b) $s'_A = s_A \cup M \cup \{t\}$, $s'_I = s_I \cup \{t\}$, and $s'_C = s_C$ for any $C \in \mathcal{A} - \{A, I\}$;
2. if a is of the form $A?B : t$, then:
   (a) $t \in \overline{s_I}$; (*enabling condition*)
   (b) $s'_A = s_A \cup \{t\}$ and $s'_C = s_C$, for all $C \in \mathcal{A} - \{A\}$.

The requirement "$M \cap Sub(s) = \emptyset$" in the above definition is usually called the *freshness check requirement*. The freshness check requirement means that only fresh nonces and short-term keys are generated during any run of the protocol. When this requirement is present, the protocol is called *with freshness check*. If the freshness check requirement is replaced by "$M \cap Sub(s_{0I}) = \emptyset$" then we will say that the protocol is *without freshness check*. This requirement imposes only that an honest agent can't generate data generated by the intruder.

We extend the notation "$[\cdot\rangle$" to events by letting $s[e\rangle s'$ whenever $s[act(e)\rangle s'$, and we call $s[e\rangle s'$ a *computation step*.

An event e of $\mathcal{P}$ is *enabled at a state* s, where $s_0[e_1\rangle s_1[\cdots[e_n\rangle s$ and $e_1, \cdots, e_n$ are events of $\mathcal{P}$, if the following properties hold:

1. $s[e\rangle s'$;

2. $LP(e) \subseteq \{e_1, \ldots, e_n\}$.

When an event e is enabled at a state s, where $s_0[e_1\rangle s_1[\cdots[e_n\rangle s$, then we will also say that e is *enabled at the sequence of events* $e_1 \cdots e_n$. It follows from (2) above that a necessary condition for an event e to be enabled at a sequence of events is that all its local predecessors be already in the sequence.

A *computation* or *run* of the protocol is a sequence of computation steps,

$$s_0[e_1\rangle s_1[\cdots[e_n\rangle s_n,$$

also written as $s_0[e_1 \cdots e_n\rangle s_n$ or even $e_1 \cdots e_n$, such that e_i is enabled at sequence $e_1 \cdots e_{i-1}$, for all $1 \leq i \leq n$.

If ξ is a run, $s_0[\xi\rangle s$, and e is an event that is enabled at s, then we will use the notation $(s, \xi)[e\rangle(s', \xi e)$, where $s[e\rangle s'$.

A run of $\mathcal{P}$ is called a *well-typed run* if its events are all well-typed. It is clear that all messages communicated in well-typed runs are bounded-length.

Let $T \subseteq \mathcal{T}_0$ a finite set. A run of $\mathcal{P}$ is called a *T-run* if the terms of all its events are built up upon T.

Example 2.2.5 A well-typed run of the protocol $\mathcal{P}_{NS}$ presented in Example 2.2.1, is given by the following sequence ξ:

$$\begin{array}{llcl}
(r_1, \sigma_1, 1) & A!I & : & (\{m\})\{m, A\}_{K_I} \\
(r_2, \sigma_2, 1) & B?A & : & \{m, A\}_{K_B} \\
(r_2, \sigma_2, 2) & B!A & : & (\{n\})\{m, n\}_{K_A} \\
(r_1, \sigma_1, 2) & A?I & : & \{m, n\}_{K_A} \\
(r_1, \sigma_1, 3) & A!I & : & \{n\}_{K_I} \\
(r_2, \sigma_2, 3) & B?A & : & \{n\}_{K_B}
\end{array}$$

, where σ_1 and σ_2 are well-typed suitable substitutions defined in Example 2.2.2.

The secrecy problem

Definition 2.2.4 (1) A term $t \in \mathcal{T}_0$ is *secret at a state* s if $t \in analz(s_A) - analz(s_I)$, for some honest agent A. A term t is secret at a state s if it can be deduced by some honest agent, but can't be deduced by the intruder;

(2) A term $t \in \mathcal{T}_0$ is *secret along a run* ξ if it is secret at s, where $s_0[\xi\rangle s$. We remark that no agent or public key is secret along any run;

(3) A run ξ is called *leaky* if there exists $t \in \mathcal{T}_0$ and a proper prefix ξ' of ξ such that t is secret along ξ' but not along ξ;

(4) The *secrecy problem* for security protocols is the problem to decide whether a security protocol has leaky runs.

Let $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$ be a protocol, and $T \subseteq \mathcal{T}_0$ a finite set. When for the protocol $\mathcal{P}$ only T-runs are considered we will say that it is a *protocol under T-runs*. The *secrecy problem for security protocols under T-runs* is the problem to decide whether a security protocol under T-runs has leaky T-runs.

If we replace the set $\mathcal{T}_0$ by $\bigcup_{A \in Ho} Secret_A$ in all the definitions above, we obtain a particular case of the secrecy problem, called the *initial secrecy problem*. That is, the initial secrecy problem is the problem to decide whether a given security protocol has runs ξ such that $t \in analz(s_I)$ for some initial secret t, where $s_0[\xi\rangle s$.

Example 2.2.6 The run ξ from Example 2.2.5 illustrates the Lowe's attack on $\mathcal{P}_{NS}$ [38]. We remark that the run ξ is leaky because n is secret at the prefix

$$\xi' = (r_1, \sigma_1, 1)(r_2, \sigma_2, 1)(r_2, \sigma_2, 2)(r_1, \sigma_1, 2)$$

of ξ, but n is not secret at ξ.

Chapter 3

Undecidability Results

The secrecy problem for security protocols is in general undecidable. The main sources of undecidability are *unbounded number of nonces* and *unbounded message length*. The unbounded number of sessions that can appear in a protocol run and the freshness requirement lead to unbounded number of nonces. The freshness requirement means that a nonce generated in a session of the protocol is new. Therefore, without a bound on the number of sessions, the number of nonces in a protocol run is unbounded. The unbounded computation power of the intruder leads to unbounded message length. The intruder can generate arbitrary-length messages and can force the agents to respond them. In [12, 24, 54, 25] was formally proved that the unbounded number of nonces leads to undecidability even if the message length is bounded. If the number of nonces is bounded, the arbitrary-length messages leads also to undecidability [26, 35, 3, 54].

The techniques used vary from model to model, and they are based on the halting problem for Turing machines [18], the halting problem for 2-counter machines (2-CM) [3, 54], Post's correspondence problem (PCP) [26, 35, 25], the implication problem for existential Horn clauses [12, 24], the reachability problem for reset Petri nets [16] etc.

In this chapter we will show that the secrecy problem is undecidable when the number of nonces is unbounded as well as when the message length is unbounded, in both cases using reductions from 2-CM [54] and PCP [25]. Undecidability is based on the intruder's capabilities to put the agents to work in order to simulate the behaviour of a 2-CM (Section 3.1), respectively the construction of a PCP solution (Section 3.2).

3.1 Reducing the halting problem for 2-CM

The halting problem for 2-counter machines was first used in [3] to prove undecidability of secrecy for unbounded message length. Later, Ramanujam and Suresh [54] proved undecidability of the secrecy problem for security protocols with freshness check under the restriction of bounded message length and for security protocols with or without freshness check under the restriction of bounded number of nonces.

A *2-counter machine* is a tuple $M = (Q, \delta, q_0, F)$, where:

- Q is a finite set of *states*;
- $\delta \subseteq Q \times \{0,1\}^2 \times Q \times \{-1,0,1\}^2$ is the *transition relation* which satisfies:

  $$(q, i_1, i_2, q', j_1, j_2) \in \delta \ \wedge \ j_k = -1 \quad \Rightarrow \quad i_k = 1,$$

  for all $k = 1, 2$.

  This condition says that the machine cannot decrement a counter unless it is strictly positive;
- q_0 is the *initial state*;
- $F \subseteq Q$ is the set of *final states*.

A transition $t = (q, i_1, i_2, q', j_1, j_2) \in \delta$ has the following meaning:

- q is the current state, and q' will be the new state after the transition t is applied;
- i_1 gives information about the first counter: $i_1 = 0$ iff the first counter's content is 0. i_2 has a similar meaning with respect to the second counter;
- j_1 specifies the operation on the first counter:
  - an incrementation operation if $j_1 = 1$;
  - an decrementation operation if $j_1 = -1$;
  - no operation if $j_1 = 0$.

  j_2 has a similar meaning with respect to the second counter.

A *configuration* of a 2-counter machine M is a triple (q, n_1, n_2), where $q \in Q$ and $n_1, n_2 \in \mathbf{N}$ are M's counters. The *transition relation between configurations* of a 2-counter machine M is defined by

$$(q, n_1, n_2) \rightarrow_M (q', n_1 + j_1, n_2 + j_2)$$

if and only if $(q, i_1, i_2, q', j_1, j_2) \in \delta$ and $i_k = 0$ iff $n_k = 0$ for all $k = 1, 2$. $\xrightarrow{*}_M$ is the reflexive and transitive closure of $\rightarrow_M$. A configuration (q, n_1, n_2) is *reachable* in M if $(q_0, 0, 0) \xrightarrow{*}_M (q, n_1, n_2)$, and it is *final* if $q \in F$.

The *halting problem for 2-counter machines* is the problem to decide whether or not a final configuration of a given 2-counter machine is reachable. It is well-known that this problem is undecidable [36].

In reducing the halting problem for 2-counter machine to the secrecy problem, the main issue we encounter is encoding natural numbers. Given a natural number n, there are mainly two ways of encoding it:

- as a nonce u using another n-linking pairwise distinct nonces:

$$\{u_0, u_1\}_K, \ldots, \{u_{n-1}, u_n = u\}_K,$$

  where u_i are nonces for each $0 \leq i \leq n$ and K is a key;

- as a tuple using only one nonce z, as follows:

$$\underbrace{(\ldots(((z, z), z), z), \ldots)}_{n \text{ times}}$$

In what follows we will analyse the 2-counter machine's simulation for both ways of encoding natural numbers stated above.

3.1.1 Infinitely many nonces and bounded-length messages

Let $M = (Q, \delta, q_0, F)$ be a 2-counter machine. A security protocol $\mathcal{P}_M$ with freshness check is defined such that a final configuration of M is reachable iff the protocol $\mathcal{P}_M$ has a well-typed leaky run.

We define the protocol $\mathcal{P}_M = (\mathcal{S}, \mathcal{C}, w)$ as follows:

1. The protocol's signature $\mathcal{S}$ consists of:

   - the agents are associated as follows:
     - two agents A and B who are going to initiate the protocol;
     - two agents A_t and B_t to each transition $t \in \delta$;
     - two agents A_q and B_q, for any final state q, who are going to reveal the secret.

     All the agents above are pairwise distinct;
   - let K be a long-term key shared by all honest agents;

- let $\mathcal{N}$ be an infinite set of nonces. Without loss of generality we may assume that $Q \subseteq \mathcal{N}$. Fix a nonce $z \in \mathcal{N}$ whose role is to define natural numbers as follows:
  - the number 0 is represented by z;
  - a number $n > 0$ will be represented in a state s by a nonce u, provided that there exist the distinct nonces $u_0 = z, \ldots, u_n = u$ such that $\{u_i, u_{i+1}\}_K \in \overline{s_I}$, for all $0 \leq i < n$.

  The operations on the counters (which are natural numbers) are simulated in the protocol by:
  - The incrementation of n by 1 is simulated by generating a new nonce u_{n+1} and sending $\{u_n, u_{n+1}\}_K$;
  - The decrementation of n by 1 is simulated by receiving the message $\{u_{n-1}, u_n\}_K$.

  A configuration (q, n_1, n_2) of M is represented in a state s of the protocol by the term $\{q, u_1, u_2\}_K \in \overline{s_I}$, where n_1 is represented by u_1 and n_2 by u_2. The initial configuration $(q_0, 0, 0)$ of M is represented in a state s by the term $\{q_0, z, z\}_K \in \overline{s_I}$.
  We consider five pairwise distinct nonces u_1, u_1', u_2, u_2', $x \in \mathcal{N}$, that are not constants;

2. The set of constants of the protocol is $\mathcal{C} = Q \cup \{z\}$;

3. The actions of the protocol are:

   - (initialization)

     A initiates the protocol by sending the initial configuration of M to start off its simulation:

     $$A!B \ : \ \{z, z\}_K, \{q_0, z, z\}_K, \{z, z\}_K$$

   - (apply a transition t)

     A transition $t = (q, i_1, i_2, q', j_1, j_2)$ of M is simulated in $\mathcal{P}_M$ by two actions performed by A_t and B_t:

     $$\begin{array}{lcl} A_t?B_t & : & \{x_1, y_1\}_K, \{q, u_1, u_2\}_K, \{x_2, y_2\}_K \\ A_t!B_t & : & (M)\, \{x_1', y_1'\}_K, \{q', u_1', u_2'\}_K, \{x_2', y_2'\}_K \end{array}$$

     where $M = \{y_k' | k \in \{1, 2\} \text{ and } j_k = 1\}$ and the following conditions hold for $k \in \{1, 2\}$:
     - if $i_k = 0$ and $j_k = 0$ then $u_k = u_k' = z$ and $x_k = y_k = x_k' = y_k' = z$;

– if $i_k = 0$ and $j_k = 1$ then $x'_k = u_k = z$, $y'_k = u'_k$ and $x_k = y_k = z$;
– if $i_k = 1$ and $j_k = -1$ then $y_k = u_k$, $x_k = u'_k$ and $x'_k = y'_k = z$;
– if $i_k = 1$ and $j_k = 0$ then $u'_k = u_k$ and $x_k = y_k = x'_k = y'_k = z$;
– if $i_k = 1$ and $j_k = 1$ then $x'_k = u_k$, $y'_k = u'_k$ and $x_k = y_k = z$.

First, A_t receives from B_t a message which specifies the current configuration together with specifications for the counters. Then, A_t does the modifications regarding the counters, updates the global configuration and sends to B_t the new configuration.

For example, the transition $t = (q, 1, 0, q', -1, 1) \in \delta$ is simulated by the following actions:

$$\begin{array}{lcl} A_t?B_t & : & \{u'_1, u_1\}_K, \{q, u_1, z\}_K, \{z, z\}_K \\ A_t!B_t & : & (\{u'_2\})\,\{z, z\}_K, \{q', u'_1, u'_2\}_K, \{z, u'_2\}_K \end{array}$$

A_t receives the current configuration $\{q, u_1, z\}_K$ and the term $\{u'_1, u_1\}_K$ that means that the first counter will be decremented and its new value will be u'_1. The incrementation of the second counter is simulated by A_t generating a new nonce u'_2 and sending $\{z, u'_2\}_K$ to B_t.

- (reveal the secret)
  If (q, n_1, n_2) is a final configuration reachable in M, then a secret will be revealed by an agent A_q:

$$\begin{array}{lcl} A_q?B_q & : & \{q, u_1, u_2\}_K \\ A_q!B_q & : & (\{x\})\,\{x\}_K \\ A_q!B_q & : & x \end{array}$$

  , for each $q \in F$.
  When A_q receives a final configuration, he sends a secret in clear to B_q.
  In order to reduce the halting problem for 2-counter machines to initial secrecy problem for security protocols with freshness check, we will consider a distinguished nonce x_0 that is an initial secret of the agent A_q, for each $q \in F$. In this case we will replace the last two actions of the reveal secret process above by the action $A_q!B_q : x_0$.

We can prove that a final configuration of M is reachable iff the protocol $\mathcal{P}_M$ has a well-typed leaky run. A few words about this proof are in order:

- We remark that any instantiation σ does not necessarily preserve the agents. In fact, we have used distinct agent in order to be able to separate distinct roles for each transition, but a run can use, by instantiations, exactly two agents;

- The fact that $\mathcal{N}$ is infinite is necessary because any contor's incrementation is simulated by generating a new nonce. Otherwise, if we don't generate a new nonce for a contor's incrementation, then the same nonce will represent in $\mathcal{P}_M$ different natural numbers. For example, in the sequence:

  $$\{z, u_1\}_K, \{u_1, u_2\}_K, \{u_2, u_1\}_K$$

  , u_1 will represent the both natural numbers 1 and 3. Therefore, in the protocol $\mathcal{P}_M$ the number 3 could be decremented many times that is allowed in M. This fact is simulated by the intruder repeatedly sending the messages $\{u_2, u_1\}_K$, $\{u_1, u_2\}_K$, $\{u_2, u_1\}_K$, $\{u_1, u_2\}_K$ and so on. So, there can be leaky runs in $\mathcal{P}_M$ that don't have a coresponding computation in M that reachs in a final configuration.

- Any computation of M can be simulated by the protocol, because the intruder's state will contain all the messages communicated between agents. This fact allows the intruder to simulate the machine's behavior: he sends as an input for a role, the messages corresponding to counter's operations together with the configuration of M, received from another roles (possible the same role). Moreover, if a final configuration (q, n_1, n_2) of M is reachable then some agent $\sigma(A_q)$ reveals a secret.

  Therefore, the protocol $\mathcal{P}_M$ has a leaky run if a final configuration of M is reachable.

- If there is a leaky run of $\mathcal{P}_M$ then some agent $\sigma_1(A_q)$ reveals a secret, that means that $\sigma_1(A_q)$ received a final configuration of M. This configuration was in the intruder's state and, therefore, either some agent $\sigma_2(A)$ or some agent $\sigma_3(A_t)$ sent it. If it was a $\sigma_3(A_t)$ agent, then the agent got another configuration from the intruder (presumed as being $\sigma_3(B_t)$). It has to be noticed that a $\sigma_3(A_t)$ agent can not perform a send action without performing first the receive action in the corresponding role (see the definition of a run). The reasoning above continues until an agent $\sigma(A)$ is reached.

  Therefore, any computation in the protocol $\mathcal{P}_M$, which leads to secret revealing, can be simulated by a computation in M, which leads to a final configuration.

- Only well-typed runs can be used in order to prove the equivalence above, because the intruder can't build up messages encrypted with the key K. Hence, all messages send by the intruder are previously sent by honests, that assures that all messages are build up using well-typed substitutions.

The main characteristics of this simulation are:

- freshness check;
- infinite number of nonces;
- no short-term keys;
- bounded-length messages (the simulation uses only well-typed runs).

Therefore, we obtain the following result:

Theorem 3.1.1 The (initial) secrecy problem for security protocols with freshness check, infinitely many nonces and bounded-length messages is undecidable.

3.1.2 Finite many nonces and arbitrary-length messages

When natural numbers are represented using only one nonce z, the protocol that will simulate the 2-counter machine will use only finitely many nonces. However, in this case we will lose the well-typeness of runs.

We will show that the secrecy problem for security protocols under T-runs is undecidable too . The proof is a slight variation of the one used in case of bounded message length.

Let $M = (Q, \delta, q_0, F)$ be a 2-counter machine. Define the security protocol $\mathcal{P}_M = (\mathcal{S}, \mathcal{C}, w)$ as follows:

1. The protocol's signature $\mathcal{S}$ consists of:
   - the agents are associated as follows:
     - two agents A and B who are going to initiate the protocol;
     - two agents A_t and B_t to each transition $t \in \delta$;
     - two agents A_q and B_q, for any final state q, who are going to reveal the secret.

     All the agents above are pairwise distinct.
   - let K be a long-term key shared by all honest agents;
   - assume that each element of Q is a nonce. Fix a nonce z whose role is to define natural numbers. A natural number n is represented in the protocol by $\underline{n}$, where:
     - $\underline{0} = z$, and
     - $\underline{n} = (\underline{n-1}, z)$ if $n > 0$.

     The operations on the counters are simulated in the protocol by:
     - The incrementation of n by pairing $\underline{n}$ and z;

– The decrementation of n by receiving the message $(\underline{n-1}, z)$.

A configuration (q, n_1, n_2) of M is represented in a state s of the protocol by the term $\{q, \underline{n_1}, \underline{n_2}\}_K$.

We consider three pairwise distinct nonces u_1, u_2, and x, that are not constants.

2. The set of constants of the protocol is $\mathcal{C} = Q \cup \{z\}$;

3. The actions of the protocol are:

- (initialization)

  A initiates the protocol by sending the initial configuration of M to start off its simulation:

$$A!B \ : \{q_0, z, z\}_K$$

- (apply a transition t)

  A transition $t = (q, i_1, i_2, q', j_1, j_2)$ of M is simulated in $\mathcal{P}_M$ by two actions performed by A_t and B_t:

$$\begin{array}{lcl} A_t?B_t & : & \{q, x_1, x_2\}_K \\ A_t!B_t & : & \{q', x'_1, x'_2\}_K \end{array}$$

  where for $k \in \{1, 2\}$ the following conditions hold :

  – if $i_k = 0$ and $j_k = 0$ then $x_k = x'_k = z$;
  – if $i_k = 0$ and $j_k = 1$ then $x_k = z$ and $x'_k = (z, z)$;
  – if $i_k = 1$ and $j_k = -1$ then $x_k = (u_k, z)$ and $x'_k = u_k$;
  – if $i_k = 1$ and $j_k = 0$ then $x_k = x'_k = u_k$;
  – if $i_k = 1$ and $j_k = 1$ then $x_k = u_k$ and $x'_k = (u_k, z)$.

  First, A_t receives the current configuration from B_t. A_t does the modifications regarding the counters, updates the global configuration and sends to B_t the new configuration.

  For example, the transition $t = (q, 1, 0, q', -1, 1) \in \delta$ is simulated by the following actions:

$$\begin{array}{lcl} A_t?B_t & : & \{q, (u_1, z), z\}_K \\ A_t!B_t & : & (\{q', u_1, (z, z)\}_K \end{array}$$

- (reveal the secret)

If (q, n_1, n_2) is a final configuration reachable in M, then a secret will be revealed by an agent A_q:

$$\begin{array}{lcl} A_q?B_q & : & \{q, u_1, u_2\}_K \\ A_q!B_q & : & (\{x\})\,\{x\}_K \\ A_q!B_q & : & x \end{array}$$

, for each $q \in F$.

When A_q receives a final configuration, he sends a secret in clear to B_q.

We consider the protocol $\mathcal{P}_M$ under T-runs, where T is the finite set of all agents together with the nonces and the key K specified as above. For the case of the initial secrecy problem, the set T will contain a supplimentary nonce x_0 with the same meanings as in previous protocol. By similar arguments to those in the previous section it can be shown that M reaches a final configuration iff there is a leaky T-run of the protocol $\mathcal{P}_M$.

In order to simulate the machine's behaviour, the intruder sends a configuration received from a role, as an input for another role (possible the same role). This means that the T-runs of the protocol $\mathcal{P}_M$ are not necessarily well-typed: the nonce u_1 from example above can be substituted with the representation of any natural number. Therefore, representing natural numbers using only one nonce leads to arbitrary length messages.

The main characteristics of this simulation are:

- finite number of nonces;
- no short-term keys;
- bounded-depth encryptions;
- unbounded-length messages
  (the simulation uses substitutions that are not necessarily well-typed).

Therefore, we obtain the following result:

Theorem 3.1.2 The (initial) secrecy problem for security protocols with or without freshness check, finitely many nonces and arbitrary-length messages is undecidable.

3.2 Reducing Post's Corespondence Problem

In what follows we will discuss the approach based on *Post's Correspondence Problem*. This approach was first used in [26] for half-word ping-pong protocols. Later, the same approach was used in [25] accompanied by the multiset rewriting formalism for security protocols, to prove undecidability of the secrecy problem under restriction of bounded message length. However, the proof in [25] does not work. We will first explain why the proof in [25] does not work and then we will provide a correct proof. Also, we will prove undecidability of the secrecy problem under restriction of bounded number of nonces by reduction from PCP.

A PCP instance is a finite set W of pairs of words over a finite alphabet Σ:

$$W = \{(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)\}.$$

A solution for W is any sequence $i_1, \ldots, i_l$, where $1 \leq i_j \leq k$ for all $1 \leq j \leq l$, such that

$$x_{i_1} x_{i_2} \cdots x_{i_l} = y_{i_1} y_{i_2} \cdots y_{i_l}.$$

The problem consists in deciding whether or not W has solutions. It is well-known that this problem is undecidable [51].

In reducing PCP to secrecy, the main problem we encounter is encoding words. Given a word $x = a_1 \cdots a_n$, there are mainly two ways of encoding it:

- separate letters by distinct nonces and encrypt each triple *(nonce,letter,nonce)*. Thus, x can be encoded by the sequence

  $$\{u_0, a_1, u_1\}_K, \ldots, \{u_{n-1}, a_n, u_n\}_K,$$

  where u_i are nonces for each $0 \leq i \leq n$ and K is a key;
- as a tuple, $x = ((\ldots(a_1, a_2), \ldots, a_{n-1}), a_n)$.

3.2.1 Infinitely many nonces and bounded-length messages

In this section we will present the reduction of PCP to the secrecy problem for security protocols with freshness check proposed in [25], adapted to our model. We will point out the flaw and we propose the solution.

Given a PCP instance W over a finite alphabet Σ, we define a security protocol $\mathcal{P}_W$ with freshness check such that W has a solution iff the protocol $\mathcal{P}_W$ has a well-typed leaky run. The protocol $\mathcal{P}_W = (\mathcal{S}, \mathcal{C}, w)$ is defined as follows:

1. The protocol's signature $\mathcal{S}$ consists of:
   - the agents are associated as follows:

  - two agents A and B who are going to initiate the protocol;
  - two agents A_i and B_i to each pair $(x_i, y_i) \in W$;
  - two agents C_a and D_a to each symbol $a \in \Sigma$, for the verification process;
  - two agents E_a and F_a for each symbol $a \in \Sigma$, to reveal the secret.

  All the agents above are pairwise distinct;

- three pairwise distinct long-term keys K, L, and R shared by all the honest agents. The key L is used for encryption of the left sequence, R for the right sequence;
- let $\mathcal{N}$ be an infinite set of nonces that contains:
  - e_1 and e_2 two distinct fixed nonces;
  - all the elements of Σ;
  - five pairwise distinct nonces u, u', v, v' and x;
  - for each pair $(x, y) \in W$, a number of pairwise distinct nonces equal to the number of letters from x and y;

2. The set of constants of the protocol is $\mathcal{C} = \Sigma \cup \{e_1, e_2\}$;

3. The actions of the protocol are:

   - (initialization)

     A initiates the protocol by sending the pair of nonces $\{e_1, e_2\}$:

     $$A!B \quad : \quad \{e_1, e_2\}_K$$

     The idea is that this pair of nonces will mark the left end of any sequence of words over W. For instance, if $x_1 = ab$, $y_1 = bac$, $x_2 = bba$, and $y_2 = aa$, then the following two sequences can be generated:

     $$\{e_1, b, u_1\}_L, \{u_1, b, u_2\}_L, \{u_2, a, u_3\}_L, \{u_3, a, u_4\}_L, \{u_4, b, u_5\}_L$$

     and

     $$\{e_2, a, v_1\}_R, \{v_1, a, v_2\}_R, \{v_2, b, v_3\}_R, \{v_3, a, v_4\}_R, \{v_4, c, v_5\}_R$$

     These two sequences simulate x_2x_1 and y_2y_1;

   - (append a pair (x_i, y_i))

The addition of a new pair (x_i, y_i) is simulated by two actions performed by the agents A_i and B_i:

$$\begin{array}{lcl} A_i?B_i & : & \{u, v\}_K \\ A_i!B_i & : & (\{u_1, \ldots, u_{s_i}, v_1, \ldots, v_{p_i}\}) \\ & & \{u, x_i^1, u_1\}_L, \{u_1, x_i^2, u_2\}_L, \ldots, \{u_{s_i-1}, x_i^{s_i}, u_{s_i}\}_L, \\ & & \{v, y_i^1, v_1\}_R, \{v_1, y_i^2, v_2\}_R, \ldots, \{v_{p_i-1}, y_i^{p_i}, v_{p_i}\}_R, \\ & & \{u_{s_i}, v_{p_i}\}_K \end{array}$$

where $x_i = x_i^1 \cdots x_i^{s_i}$ and $y_i = y_i^1 \cdots y_i^{p_i}$.

A_i receives (u, v) "the last pair of nonces" of the current sequences of words, appends x_i and y_i, correspondingly and sends (u_{s_i}, v_{p_i}) the "last pair of nonces" of newly constructed sequences in order to be used to link other pairs to the current sequences;

- (verification)

  Two agents C_a and D_a check whether or not any two letters on corresponding positions are the same:

$$\begin{array}{lcl} C_a?D_a & : & \{u, v\}_K, \{u', a, u\}_L, \{v', a, v\}_R \\ C_a!D_a & : & \{u', v'\}_K \end{array}$$

  where $a \in \Sigma$;

- (reveal the secret)

  When a solution to W is found, the agent E_a reveals a secret:

$$\begin{array}{lcl} E_a?F_a & : & \{u, v\}_K, \{e_1, a, u\}_L, \{e_2, a, v\}_R \\ E_a!F_a & : & (\{x\})\{x\}_K \\ E_a!F_a & : & x \end{array}$$

  where $a \in \Sigma$.

This solution is wrong because it uses the same pair $\{u, v\}_K$ of nonces both for the appending process and for the verification one. In this way, new pairs (x_i, y_i) can be appended while the verification process of some pair from the sequences is not done.

For example, if we consider the PCP instance $W = \{(ca, a), (b, cb)\}$, then the following well-typed run of $\mathcal{P}_W$ is leaky:

- (initialization)

$$A!B \quad : \quad \{e_1, e_2\}_K$$

- (append the pair (ca, a))

$$\begin{array}{lcl} A?B & : & \{e_1, e_2\}_K \\ A!B & : & (\{u_1, u_2, v_1\}) \\ & & \{e_1, c, u_1\}_L, \{u_1, a, u_2\}_L \\ & & \{e_2, a, v_1\}_R \\ & & \{u_2, v_1\}_K \end{array}$$

- (verification process)

$$\begin{array}{lcl} A?B & : & \{u_2, v_1\}_K, \{u_1, a, u_2\}_L, \{e_2, a, v_1\}_R \\ A!B & : & \{u_1, e_2\}_K \end{array}$$

- (append the pair (b, cb))

$$\begin{array}{lcl} A?B & : & \{u_1, e_2\}_K \\ A!B & : & (\{u_3, v_2, v_3\}) \\ & & \{u_1, b, u_3\}_L \\ & & \{e_2, c, v_2\}_R, \{v_2, b, v_3\}_R \\ & & \{u_3, v_3\}_K \end{array}$$

- (verification process)

$$\begin{array}{lcl} A?B & : & \{u_3, v_3\}_K, \{u_1, b, u_3\}_L, \{v_2, b, v_3\}_R \\ A!B & : & \{u_1, v_2\}_K \end{array}$$

- (reveal the secret)

$$\begin{array}{lcl} A?B & : & \{u_1, v_2\}_K, \{e_1, c, u_1\}_L, \{e_2, c, v_2\}_R \\ A!B & : & (\{x\})\{x\}_K \\ A!B & : & x \end{array}$$

Therefore, $\mathcal{P}_W$ reveals the secret, but W has no solution. This problem occurs because the same pairs of nonces are used both for the appending process and for the verification one.

Our solution makes distinction between these two processes. We use pairs $\{u, v\}_K$ of nonces for appending and pairs $\{(\$, u), (\$, v)\}_K$ for verification, where $\$$ is a constant nonce. Thus, the protocol $\mathcal{P}_W$ is updated as follows:

- the actions for initialization and appending stay the same;

- two agents C and D are considered to start the verification process by changing any pair (u, v) of nonces they receive into a pair $((\$, u), (\$, v))$:

$$\begin{array}{lcl} C?D & : & \{u, v\}_K \\ C!D & : & \{(\$, u), (\$, v)\}_K \end{array}$$

- the verification process is updated as follows:

$$\begin{array}{lcl} C_a?D_a & : & \{(\$,u),(\$,v)\}_K, \{u',a,u\}_L, \{v',a,v\}_R \\ C_a!D_a & : & \{(\$,u'),(\$,v')\}_K \end{array}$$

  where $a \in \Sigma$;

- revealing the secret process is correspondly updated:

$$\begin{array}{lcl} E_a?F_a & : & \{(\$,u),(\$,v)\}_K, \{e_1,a,u\}_L, \{e_2,a,v\}_R \\ E_a!F_a & : & (\{x\})\{x\}_K \\ E_a!F_a & : & x \end{array}$$

  where $a \in \Sigma$.

  One more remark is in order: one may say that the verification process performed by E_a and F_a can be simplified to

$$\begin{array}{lcl} E_a?F_a & : & \{(\$,e_1),(\$,e_2)\}_K \\ E_a!F_a & : & (\{x\})\{x\}_K \\ E_a!F_a & : & x \end{array}$$

  However, this is wrong because C may receive $\{e_1,e_2\}_K$. In such a case, C returns $\{(\$,e_1),(\$,e_2)\}_K$ which can close the verification process although no solution exists.

With this updates for the protocol $\mathcal{P}_W$, it is not difficult to prove that W has a solution iff $\mathcal{P}_W$ has a well-typed leaky run.

As we can see, if $\mathcal{P}_W$ updated has a well-typed leaky run then W has certainly a solution and a situation like that in example above is not possible. The intruder can append a new pair only after the verification process is done for a hole sequence of pairs. This is because the intruder's state has only pairs of nonces $\{u,v\}_K$ that delimitates pairs of words in W. The nonces that links letters from words are in the intruder's state in the form $\{(\$,u),(\$,v)\}_K$ and can't be send in appending process by the intruder in a well-typed run.

In the case in which the nonces that links the letters aren't fresh generated, in the verification process the intruder can force the agents to skip words or pieces of words. In this situation there can be well-typed leaky runs for $\mathcal{P}_W$ but W has not necessarily a solution.

The main characteristics of this simulation are:

- freshness check;
- infinite number of nonces;

- no short-term key;
- bounded-length messages.

Therefore, with the same discussion for the case of the intial secrecy problem as in Section 3.1.1, we obtain the following result:

Theorem 3.2.1 The (initial) secrecy problem for security protocols with freshness check, infinitely many nonces and bounded-length messages is undecidable.

3.2.2 Finite many nonces and arbitrary-length messages

If we model words as tuples of letters, then we will show that the secrecy problem for security protocols under T-runs is undecidable. The protocol is pretty much the same as the one presented above.

Given a PCP instance W over a finite alphabet Σ, a security protocol $\mathcal{P}_W = (\mathcal{S}, \mathcal{C}, w)$ is defined as follows:

1. The protocol's signature $\mathcal{S}$ consists of:
   - the agents are associated as in previous case:
     - two agents A and B who are going to initiate the protocol;
     - two agents A_i and B_i to each pair $(x_i, y_i) \in W$;
     - two agents C and D to start the verification process;
     - two agents C_a and D_a to each symbol $a \in \Sigma$, for the verification process;
     - two agents E_a and F_a for each symbol $a \in \Sigma$, to reveal the secret.

     All the agents above are pairwise distinct;
   - let K be a long-term key shared by all honest agents;
   - assume that the elements of Σ are nonces and fix three distinct nonces e_1, e_2, and \$ with the same meanings as in previous case. We also consider three pairwise distinct nonces u, v, and x, that are not constants.
2. The set of constants of the protocol is $\mathcal{C} = \Sigma \cup \{e_1, e_2, \$\}$;
3. The actions of the protocol are:
   - (initialization)
   $$A!B \quad : \quad \{e_1, e_2\}_K$$

- (append a pair (x_i, y_i))
  The addition of a new pair (x_i, y_i) is simulated by two actions performed by the agents A_i and B_i:

$$\begin{array}{lcl} A_i?B_i & : & \{u, v\}_K \\ A_i!B_i & : & \{(\ldots(u, x_i^1), \ldots, x_i^{s_i}), (\ldots(v, y_i^1), \ldots, y_i^{p_i})\}_K \end{array}$$

  where $x_i = x_i^1 \cdots x_i^{s_i}$ and $y_i = y_i^1 \cdots y_i^{p_i}$.
  A_i receives the current sequences of words, appends x_i and y_i, correspondingly and sends the newly constructed sequences in order to be used to link other pairs to the current sequences;

- (start of verification)

$$\begin{array}{lcl} C?D & : & \{u, v\}_K \\ C!D & : & \{(\$, u), (\$, v)\}_K \end{array}$$

- (verification)

$$\begin{array}{lcl} C_a?D_a & : & \{(\$, (u, a)), (\$, (v, a))\}_K \\ C_a!D_a & : & \{(\$, u), (\$, v)\}_K \end{array}$$

  where $a \in \Sigma$;

- (reveal the secret)
  When a solution for W is found, the agent E_a reveals a secret:

$$\begin{array}{lcl} E_a?F_a & : & \{(\$, (e_1, a)), (\$, (e_2, a))\}_K \\ E_a!F_a & : & (\{x\})\{x\}_K \\ E_a!F_a & : & x \end{array}$$

  where $a \in \Sigma$.

We consider the protocol $\mathcal{P}_W$ under T-runs, where T is the finite set of all agents together with the nonces and the key K specified as above and, it is not difficult to prove that W has a solution iff there is a leaky T-run of the protocol $\mathcal{P}_W$.

As we can see, using the constant nonce $\$$ in the verification process avoid the situations to those in previous section. Because the simulation uses non well-typed runs, new pairs (x_i, y_i) can be appended while the verification process is not done. But, in this case the current sequences of words obtained after the appending process can't be send in the verification process.

The main characteristics of this simulation are:

- finite number of nonces;
- no short-term key;
- bounded-depth encryptions;
- unbounded-length messages.

Therefore, with the same discussion for the case of the intial secrecy problem as in Section 3.1.2, we obtain the following result:

Theorem 3.2.2 The (initial) secrecy problem for security protocols with or without freshness check, finitely many nonces and arbitrary-length messages is undecidable.

Chapter 4

Decidability and Complexity Results

As we have seen in the previous chapter, the prominent sources of undecidability are *unbounded number of nonces* and/or *unbounded message length* . By imposing various (reasonable) restrictions on security protocols, decidability of secrecy can be gained. For instance, when both message length and the number of nonces is bounded, the secrecy is decidable [24]. But this is not the only case. In [57, 43, 3], bounds on the number of sessions that can occur in any run of the protocol, lead again to decidability of secrecy. Some semantic criteria which give decidability in the presence of terms of arbitrary length and with finitely many nonces were first introduced in [53], but the results obtained there are wrong. Another approach based on the same semantic criteria was proposed later in [62]. Imposing syntactic restrictions to security protocols, decidability results can be also obtained. In [40], decidability of a syntactic subclass of protocols is proved for unbounded number of nonces and bounded message length. Decidability for a subclass of tagged protocols without any external bound is obtain in [55].

In this chapter we will provide a quite complete picture of the complexity of (initial) secrecy problem for bounded protocols, all the results being developed under the same formalism. We will prove that the initial secrecy problem for bounded protocols whithout freshness check is DEXPTIME-complete and the (initial) secrecy problem for bounded protocols with freshness check is NEXPTIME-complete. We also prove NP-completeness of the secrecy problem for bounded protocols with or without freshness check under bounded number of sessions. Moreover, we will corect some complexity results for bounded protocols in the restricted form, stated in [25] under the multiset rewriting formalism and provide an undecidability result closely related to a problem left open in [25] (Table 9, page 282). In Section 4.2 we correct some flawed statements from [53] and we show decidability of secrecy using the same frame as in that paper. We close the chapter presenting a syntactic restriction (tagging) and how it can be used to obtain decidability of secrecy.

4.1 Bounded Protocols

Definition 4.1.1 Let $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w = a_1 \dots a_l)$ be a protocol, $T \subseteq \mathcal{T}_0$ a finite set, and $k \geq max\{|t(a_i)| \,/\, 1 \leq i \leq l\}$.

- A run of $\mathcal{P}$ is called a (T,k)-*run* if all messages communicated in the course of the run are built up upon T and have length at most k;
- A *1-session* (T,k)-*run* of $\mathcal{P}$ is any (T,k)-run of $\mathcal{P}$ obtained by applying each role at most once (not necessarily in its entirety), under the same substitution (i.e., all its events are defined by using the same substitution).

Therefore, any 1-session (T,k)-run has length at most $|w|$. When for the protocol $\mathcal{P}$ only (T,k)-runs (1-session (T,k)-runs) are considered we will say that it is a (T,k)-*bounded protocol* (*1-session* (T,k)-*bounded protocol*). The (*initial*) *secrecy problem* for (T,k)-*bounded protocols* (*1-session* (T,k)-*bounded protocols*) is formulated with respect to (T,k)-runs (1-session (T,k)-runs) only, by taking into consideration the set T instead of $\mathcal{T}_0$. A (*1-session*) *bounded protocol* is a (1-session) (T,k)-bounded protocol, for some T and k.

Let $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$ be a (T,k)-bounded protocol. Then:

1. the number of messages communicated in the course of any (T,k)-run of $\mathcal{P}$ is bounded by
$$k^3|T|^{\frac{k+1}{2}} = 2^{3\log k + \frac{k+1}{2}\log|T|};$$
2. the number of instantiations (substitutions) of a given role r of $\mathcal{P}$ with messages of length at most k over T is bounded by
$$\left(2^{3\log k + \frac{k+1}{2}\log|T|}\right)^{|r|(\frac{k+1}{2}+2)}$$
(r has exactly $|r|$ actions, and each action has at most $\frac{k+1}{2}+2$ elements that can be substituted);
3. the number of (T,k)-*events of* $\mathcal{P}$ (i.e., events that can occur in all (T,k)-runs of $\mathcal{P}$) is bounded by
$$\begin{array}{rcl} \text{number of } (T,k)\text{-events} & \leq & \sum_{r\in Roles(\mathcal{P})} |r| \cdot 2^{(3\log k + \frac{k+1}{2}\log|T|)|r|(\frac{k+1}{2}+2)} \\ & \leq & \sum_{r\in Roles(\mathcal{P})} |r| \cdot 2^{(3\log k + \frac{k+1}{2}\log|T|)|w|(\frac{k+1}{2}+2)} \\ & = & |w| \cdot 2^{(3\log k + \frac{k+1}{2}\log|T|)|w|(\frac{k+1}{2}+2)} \\ & = & 2^{\log|w| + (3\log k + \frac{k+1}{2}\log|T|)|w|(\frac{k+1}{2}+2)}. \end{array}$$

Define the *size* of a (T, k)-bounded protocol $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$ as being

$$size(\mathcal{P}) = |w| + k \log |T|.$$

This is a realistic measure. It takes into consideration the number of actions and the maximum number of bits necessary to represent messages of length at most k over T (we remark that the size of the representation of $\mathcal{P}$ is polynomial in $size(\mathcal{P})$). As we can see, the number of events that can occur in all (T, k)-runs of a (T, k)-bounded protocol $\mathcal{P}$ is exponential in $poly(size(\mathcal{P}))$, for some polynomial $poly$.

In a (T, k)-bounded protocol the set of messages that can appear in all (T, k)-runs is bounded and, as a result, the (initial) secrecy problem is decidable.

In [25], several DEXPTIME-completeness results for the initial secrecy problem for bounded protocols in the restricted form and with bounded existentials have been proposed (the concept of "existential" in [25] corresponds to the concept of "freshness check" in thesis. In Section 4.1.4 we will provide the reader with full details about the relationship between the formalism in [25] and the one used in thesis). Unfortunately, all these results are wrong just because they are based on an algorithm which does not work for the case of protocols with existentials (see Section 4.1.4). In fact, it turns out that the presence of existentials requires much more effort than their absence, making the secrecy problem complete for NEXPTIME. We will prove this result as follows. First, by means of a slight variation of an algorithm in [25] adapted to the formalism used in thesis we show that the initial secrecy problem for bounded protocols without freshness check is DEXPTIME-complete (Section 4.1.1). Then, we point out why this algorithm does not work for the case of freshness check and we show that in this case the (initial) secrecy problem is NEXPTIME-complete (Section 4.1.2). In Section 4.1.3 we will show that the (initial) secrecy problem is NP-complete for 1-session bounded protocols with or without freshness check. Section 4.1.4 recalls the MSR formalism [25], points out why the DEXPTIME-completeness results in [25] are wrong, and then corrects them. We also provide an undecidability result closely related to a problem left open in [25] (Table 9, page 282).

4.1.1 The Initial Secrecy Problem for Bounded Protocols without Freshness Check

Let $\mathcal{P}$ be a bounded protocol. If $s_0[\xi\rangle s$ is a run of $\mathcal{P}$ and $e \in Events(\xi)$ is an event enabled at the state s such that $s[e\rangle s'$, then $s = s'$. That is, if we extend a run ξ by an event already in the run, then the state generated by ξ does not change.

Definition 4.1.2 Let $\mathcal{P}$ be a bounded protocol and ξ be a run of $\mathcal{P}$.

- An event e of $\mathcal{P}$ *extends* ξ *w.r.t. the intruder state* (*extends* ξ, for short) if e is an event enabled at ξ and $e \notin Events(\xi)$;
- A run ξ of $\mathcal{P}$ is called *maximal w.r.t. the intruder state* (*maximal*, for short) if no event of $\mathcal{P}$ can extend ξ.

We remark that, when an event e extends ξ, the state generated by ξ may not change by applying the event e. The *extends* ξ notion does not avoid such a case because the event e may be necessary in order to apply another event e' which succeeds e in some role (according to the computation rule). In a maximal run each event appears at most once, because a second appearance of the same event does not change the current state. The terminology "maximal" is justified by the fact that in a maximal run the intruder get a maximum set of knowledge.

Maximal runs of bounded protocols without freshness check have special properties and in what follows we will discuss them. Recall that the protocols without freshness check are obtained by replacing the freshness check requirement by "$M \cap Sub(s_{0I})$" (see Section 2.2).

Lemma 4.1.1 Let $\mathcal{P}$ be a bounded protocol without freshness check and ξ and ξ' two maximal runs of $\mathcal{P}$. Then, $Events(\xi) = Events(\xi')$.

Proof Assume that there exists an event $e \in Events(\xi) - Events(\xi')$ and e is the first event in ξ that satisfies this condition. Let $\xi = \xi_1 e \xi_2$. Since the protocol $\mathcal{P}$ is without freshness check and $Events(\xi_1) \subseteq Events(\xi')$, we have that e is enabled at ξ'. This shows that the event e can extends ξ', contradicting its maximality. Therefore, $Events(\xi) \subseteq Events(\xi')$. Similarly, we can prove that $Events(\xi') \subseteq Events(\xi)$. Hence $Events(\xi) = Events(\xi')$. □

Corollary 4.1.1 Let $\mathcal{P}$ be a bounded protocol without freshness check and $\xi = e_1 \cdots e_n$ a maximal run of $\mathcal{P}$. Then, any other maximal run of $\mathcal{P}$ is of the form $\pi(\xi) = e_{\pi(1)} \cdots e_{\pi(n)}$, where π is a permutation of the set $\{1 \cdots n\}$.

Corollary 4.1.2 Let $\mathcal{P}$ be a bounded protocol without freshness check and $s_0[\xi\rangle s$ and $s_0[\xi'\rangle s'$ maximal runs of $\mathcal{P}$. Then, $s = s'$.

Theorem 4.1.1 The initial secrecy problem for bounded protocols without freshness check is in DEXPTIME.

Proof The following algorithm, which is a slight variation of the algorithm in [25] adapted to the formalism used in this thesis, decides whether or not a bounded protocol without freshness check has leaky runs with respect to initial secrets.

Algorithm A1

input: $\mathcal{P}$ a (T,k)-bounded protocol without freshness check;
output: "leaky protocol" if $\mathcal{P}$ has some leaky (T,k)-run w.r.t. initial secrets, and "non-leaky protocol", otherwise;

```
begin
  let E' be the set of all (T,k)-events of P;
  ξ := λ;                          % λ denotes the empty sequence
  s := s_0;
  repeat
    E := E';
    E' := ∅;
    bool := 0;
    while E ≠ ∅ do
     begin
       choose e ∈ E;
       E := E − {e};
       if (s, ξ)[e⟩(s', ξe) then
          begin
            s := s';
            ξ := ξe;
            bool := 1;
          end
        else E' := E' ∪ {e};
     end
  until bool = 0;
  if (⋃_{A∈Ho} Secret_A) ∩ analz(s_I) ≠ ∅
     then "leaky protocol" else "non-leaky protocol"
end.
```

In the algorithm A1, E' is the set of all events that could not be applied in the previous cycle. Initially, E' is the set of all events of the protocol. The boolean variable $bool$ takes the value 0 when no event in E can be applied.

To show the correctness of the algorithm, we will prove that the protocol is leaky with respect to initial secrets if and only if the run $s_0[\xi\rangle s$ generated by algorithm is leaky with respect to initial secrets (i.e., $(\bigcup_{A\in Ho} Secret_A) \cap analz(s_I) \neq \emptyset$). The algorithm generates a maximal run ξ with respect to the intruder's state (that is, the state s has the property that $analz(s_I)$ is the maximum set of knowledge the intruder can get). Indeed, if $\mathcal{P}$ is leaky with respect to initial secrets then there exists $s_0[\xi'\rangle s'$ a leaky (T,k)-run of $\mathcal{P}$. ξ' is leaky implies that there exists $x \in (\bigcup_{A\in Ho} Secret_A) \cap analz(s'_I)$. If ξ' is not a maximal run, we extend ξ' to a maximal run ξ'' of $\mathcal{P}$ ($\xi'' = \xi'$ if ξ' is a maximal run), where $s_0[\xi''\rangle s''$. Therefore,

$x \in analz(s_I'')$. Corollary 4.1.2 leads to s=s" and therefore $x \in analz(s_I)$ that shows that the maximal run ξ is leaky with respect to initial secrets. Conversely, is clear that if ξ is leaky with respect to initial secrets then $\mathcal{P}$ is leaky with respect to initial secrets.

The algorithm terminates in exponential time with respect to $poly(size(\mathcal{P}))$, for some polynomial $poly$. Indeed, if no event in E can extend the current run ξ, then the algorithm terminates. Otherwise, all events in E that can extend the current run, taken in an arbitrary but fixed order, are applied in one cycle of `repeat`. The next cycle will process, in the same way, the elements of E that could not be applied at the previous cycle. Therefore, the number of cycles of `repeat` is bounded by $|E|$ (each cycle applies at least one event, except for the last one). At the first cycle of `repeat` the number of cycles in `while` is $|E|$, at the second cycle of `repeat` the number of cycles in `while` is at most $|E| - 1$, and so on. Therefore, the number of cycles in `while`, in all `repeat` cycles, is bounded by $|E| + (|E| - 1) + \cdots + 1 = \mathcal{O}(|E|^2)$. Therefore, the complexity of the algorithm is exponential in $poly(size(\mathcal{P}))$, for some polynomial $poly$, showing that the initial secrecy problem for bounded protocols without freshness check is in DEXPTIME. □

DEXPTIME-hardness of the initial secrecy problem for protocols in the restricted form, with bounded length messages, no existentials, and an intruder with no existentials, has been proved in [24] by exhibiting a reduction from the implication problem for Horn clauses without existential quantifiers and function symbols. We will show that the initial secrecy problem for bounded protocols without freshness check is DEXPTIME-hard. To this we will exhibit a simpler reduction than the one in [24] based on the membership problem for unary logic programs [15].

Recall first the concept of a unary logic program. Let Σ be a set consisting of one constant symbol $\perp$ and finitely many unary function symbols, let $Pred$ be a finite set of unary predicate symbols, and x be a variable. A *unary logic program* over Σ, $Pred$, and x is a finite set of clauses of the form

$$p_0(t_0) \leftarrow p_1(t_1), \ldots, p_n(t_n)$$

or

$$p_0(t_0) \leftarrow true,$$

where $p_0, \ldots, p_n \in Pred$, and $t_0, \ldots, t_n$ are terms over $\Sigma \cup \{x\}$ with t_0 being flat, that is, $t_0 \in \{\perp, x, f(x) | f \in \Sigma - \{\perp\}\}$. Moreover, all clauses with $p_0(\perp)$ in the head have only $true$ in the body.

An *atom* is a construct of the form $p(t)$, where $p \in Pred$ and t is a term. If t is a *ground term*, that is, it does not contain x, then $p(t)$ is called a *ground atom*.

A *proof tree* for a ground atom $p(t)$ under a unary logic program LP is any tree that satisfies:

- its nodes are labeled by ground atoms;
- the root is labeled by $p(t)$;
- each intermediate node which is labeled by some B has children labeled by $B_1, \ldots, B_n$, where $B \leftarrow B_1, \ldots, B_n$ is a ground instance of a clause in LP (i.e., the variable x is substituted by ground terms over Σ);
- all the leaves are labeled by $true$.

The *membership problem for unary logic programs* is the problem to decide, given a unary logic program LP and a ground atom $p(t)$, whether there exists a proof tree for $p(t)$ under LP. In [15] it has been proved that this problem is DEXPTIME-complete (being equivalent to the type-checking problem for path-based approximation for unary logic programs).

Theorem 4.1.2 The initial secrecy problem for bounded protocols without freshness check is DEXPTIME-hard.

Proof Let LP be a unary logic program over some Σ, $Pred$, and x, and let $p(t)$ be a ground atom over Σ and $Pred$. Define a security protocol without freshness check $\mathcal{P}_{LP}$ as follows:

- to each element $e \in \Sigma \cup Pred \cup \{x\}$ associate a nonce u_e. Except for u_x, all these nonces are constants of the protocol;
- encode terms and atoms as follows:
  - $\langle e \rangle = u_e$, for all $e \in \{\bot, x\}$;
  - $\langle f(t) \rangle = (u_f, \langle t \rangle)$, for any unary function symbol f and term t;
  - $\langle p(t) \rangle = (u_p, \langle t \rangle)$, for any predicate symbol p and term t.
- consider the agents A_C and B_C, for any clause C from LP and two agents A and B to reveal the secret. It is assumed that they are pairwise distinct;
- consider a key K known only by the honest agents;
- $Secret_A = \{y\}$, where y is a distinct nonce, and $Secret_X = \emptyset$, for all $X \neq A$;

- to each clause $C:\ p_0(t_0) \leftarrow p_1(t_1), \ldots, p_n(t_n)$ we associate the role

$$\begin{array}{lcl} A_C?B_C & : & \{\langle p_1(t_1)\rangle\}_K, \ldots, \{\langle p_n(t_n)\rangle\}_K \\ A_C!B_C & : & \{\langle p_0(t_0)\rangle\}_K \end{array}$$

- to each clause $C:\ p_0(t_0) \leftarrow true$ we associate the role

$$A_C!B_C \ : \ \{\langle p_0(t_0)\rangle\}_K$$

- the following role reveals an initial secret if $p(t)$ has a tree proof under LP:

$$\begin{array}{lcl} A?B & : & \{\langle p(t)\rangle\}_K \\ A!B & : & y \end{array}$$

We consider the protocol $\mathcal{P}_{LP}$ under (T, k)-runs, where T is the set of all elements mentioned above (nonces, agents, the key K, and the nonce y) and k is a suitable chosen constant (linear in the maximum length of some clause under some instantiation used to decide the membership of $p(t)$). Then, $p(t)$ has a tree proof in LP if and only if the protocol $\mathcal{P}_{LP}$ under (T, k)-runs reveals the secret. □

Corollary 4.1.3 The initial secrecy problem for bounded protocols without freshness check is DEXPTIME-complete.

Summing up, bounded protocols without freshness check are characterized by:

- bounded number of nonces and short-term keys;
- bounded-length messages (but the protocol runs may be non-well-typed);
- unbounded number of sessions.

Discussion

One may ask whether the algorithm A1 in proof of Theorem 4.1.1 can be used to decide the secrecy problem for bounded protocols without freshness check. In what follows we will show that the algorithm A1 can't be used, because, in this case, the algorithm can generate a non-leaky maximal run, but the protocol can have a leaky run. In order to show that, we consider the following protocol under

$(T, 3)$-runs, where T is a finite subset of $\mathcal{T}_0$ which includes the agents A, B, C and D, the nonces x and y, and the long-term key K:

$$\begin{array}{lcl} A!B & : & (\{x\})\{x\}_K \\ B?A & : & \{x\}_K \\ C!D & : & (\{y\})y \\ D?C & : & y \end{array}$$

If the algorithm applies all the events based on the third action and then applies all the events based on the first two actions, a non-leaky maximal run is generated; therefore, the algorithm outputs "non-leaky protocol". However, the protocol is leaky.

As a conclusion, the order in which the events are applied when freshness check is not required but "new secrets" are generated in the course of runs, is crucial.

4.1.2 The (Initial) Secrecy Problem for Bounded Protocols with Freshness Check

The algorithm A1 given in Theorem 4.1.1 from previous section cannot be applied to the (initial) secrecy problem for bounded protocols with freshness check. We will illustrate this by considering two examples, one for the initial secrecy problem and one for the secrecy problem.

Example 4.1.1 Let $\mathcal{P}_1$ be the protocol given below:

$$\begin{array}{lcl} A!B & : & (\{K\})x_0, K \\ B?A & : & x_0, K \\ C!D & : & (\{K'\})K' \\ D?C & : & K' \end{array}$$

In this protocol, A, B, C, and D are constants, x_0 is an initial secret of A, and K and K' are keys in a finite non-empty set of short-term keys.

Clearly, the protocol is leaky. However, if the algorithm A1 applies first all the events based on the third action, then no event based on the first action will be enabled. Therefore, the algorithm generates a non-leaky maximal run, and concludes that the protocol is not leaky.

It should be clear that the algorithm A1 does not work for the secrecy problem either. To be more convincing we will provide an example.

Example 4.1.2 Let $\mathcal{P}_2$ be the protocol given below:

$$\begin{array}{lcl} A!B & : & (\{x\})\{x\}_K \\ B?A & : & \{x\}_K \\ A!B & : & K \\ B?A & : & K \\ C!D & : & (\{y\})y \\ D?C & : & y \end{array}$$

where A, B, C, and D are constants in the protocol, x and y are nonces, and K is a key.

The protocol is leaky. However, if all events based on the fifth action are applied first, then the algorithm A1 generates a non-leaky maximal run and concludes that the protocol is not leaky.

All these examples show that the order in which events are applied when freshness check is required, is crucial. That makes the secrecy problem for the case "with freshness check" considerable harder than the case "without freshness check".

Theorem 4.1.3 The (initial) secrecy problem for bounded protocols with freshness check is NEXPTIME-complete.

Proof The following non-deterministic algorithm decides the secrecy problem for bounded protocols with freshness check.

Algorithm A2
`input:` $\mathcal{P}$ a (T,k)-bounded protocol with freshness check;
`output:` "leaky protocol" if $\mathcal{P}$ has leaky (T,k)-runs;
`begin`
  `let` E be the set of all (T,k)-events of $\mathcal{P}$;
  `guess` a sequence $\xi := e_1 \cdots e_m$ of pairwise distinct events from E;
  `if` ξ is a run `then`
    `begin`
      `let` $s_0[e_1\rangle s_1[\cdots[e_m\rangle s_m$;
      $Secret := \bigcup_{A \in Ho} Secret_A$;
      `for` $i := 1$ `to` $m-1$ `do`
          $Secret := Secret \cup ((\bigcup_{A \in Ho}(analz(s_{iA}) \cap T)) - analz(s_{iI}))$;
      `if` $Secret \cap analz(s_{mI}) \neq \emptyset$ `then` "leaky protocol";
    `end`
`end.`

If the "`for`" statement is dropped in the algorithm A2, the new algorithm will decide the initial secrecy problem for bounded protocols with freshness check.

It is easy to see that the algorithm is correct and terminates in non-deterministic exponential time with respect to $poly(size(\mathcal{P}))$, for some polynomial $poly$. Therefore, the (initial) secrecy problem for bounded protocols with freshness check is in NEXPTIME.

To prove completeness, we shall reduce any language in NEXPTIME to the initial secrecy problem for bounded protocols with freshness check (this is also sufficient for the secrecy problem). So, suppose that L is a language decided by a non-deterministic Turing machine $TM = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ in time 2^n, where Q is the set of states, Σ is the input alphabet, Γ is the tape alphabet, q_0 is the initial state, $\Box$ is the blank symbol, F is the set of final states, and $\delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{-1, 1\}$ is the transition relation (-1 specifies a move to the left, while 1 specifies a move to the right). A configuration of TM will be written in the form (q, α, a, β), where q is the current state, a is the symbol scanned by the tape head, α is the string to the left of the tape head, and β is the string to the right of the tape head.

For each input w of TM we construct a (T, k)-bounded protocol $\mathcal{P}_{TM,w}$ with freshness check whose size is polynomial in $n = |w|$ and such that w is accepted by TM in time 2^n if and only if $\mathcal{P}_{TM,w}$ under (T, k)-runs is leaky. The protocol is as follows:

- we assume that all the elements of $Q \cup \Gamma$ are nonces. We also consider a distinguished nonce $\$$. All these nonces are constants of the protocol. A set of $2^n + ||w||$ distinct nonces, which are not constants of the protocol, is also considered ($||w|| = |w| + 1$, if $w \neq \lambda$, and $||w|| = 2$, otherwise);

- the agents are A, B, $A_{t,b}$, $B_{t,b}$, $A_{t\$,b}$, $B_{t\$,b}$, $A_{b,t}$, $B_{b,t}$, $A_{b,t\$}$, $B_{b,t\$}$, $A_{\lambda,t}$, $B_{\lambda,t}$, $A_{q,b}$, $B_{q,b}$, $A_{q,b\$}$, and $B_{q,b\$}$, for any transition t of TM, $b \in \Gamma$, and final state q;

- all honest agents will share a long-term key K;

- it is assumed that $Secret_{A_{q,a}} = Secret_{A_{q,a\$}} = \{x\}$ for any final state q and $a \in \Gamma$, where x is a distinguished nonce, and $Secret_X = \emptyset$ for all the other agents X.

- a configuration $(q, a_1 \cdots a_n, a, b_1 \cdots b_m)$ of TM is represented in the protocol $\mathcal{P}_{TM,w}$ by a sequence of the form

$$\{u_1, a_1, u_2\}_K, \ldots, \{u_n, a_n, u_{n+1}\}_K, \{u_{n+1}, (q, a), u_{n+2}\}_K,$$
$$\{u_{n+2}, b_1, u_{n+3}\}_K, \ldots, \{u_{n+m+1}, (b_m, \$), u_{n+m+2}\}_K,$$

  where $u_1, \ldots, u_{n+m+2}$ are distinct nonces.

- the protocol actions are:
  - A initiates the protocol and sends $w = a_1 \cdots a_n$ to B:

$$\begin{cases} A!B & : & (\{u_1, u_2\})\{u_1, (q_0, (\Box, \$)), u_2\}_K, & \text{if } n = 0 \\ A!B & : & (\{u_1, u_2\})\{u_1, (q_0, (a_1, \$)), u_2\}_K, & \text{if } n = 1 \\ A!B & : & (\{u_1, \ldots, u_{n+1}\}) & \\ & & \{u_1, (q_0, a_1), u_2\}_K, \{u_2, a_2, u_3\}_K \ldots, & \\ & & \{u_{n-1}, a_{n-1}, u_n\}_K, \{u_n, (a_n, \$), u_{n+1}\}_K, & \text{if } n > 1; \end{cases}$$

  - a transition $t = (q, a, q', a', -1)$ is simulated by:

$$\begin{array}{rcl} A_{t,b}?B_{t,b} & : & \{u, b, v\}_K, \{v, (q, a), z\}_K \\ A_{t,b}!B_{t,b} & : & (\{v'\})\{u, (q', b), v'\}_K, \{v', a', z\}_K \end{array}$$

    or

$$\begin{array}{rcl} A_{t\$,b}?B_{t\$,b} & : & \{u, b, v\}_K, \{v, (q, (a, \$)), z\}_K \\ A_{t\$,b}!B_{t\$,b} & : & (\{v'\})\{u, (q', b), v'\}_K, \{v', (a', \$), z\}_K \end{array}$$

    for any $b \in \Gamma$;

  - a transition $t = (q, a, q', a', 1)$ is simulated by:

$$\begin{array}{rcl} A_{b,t}?B_{b,t} & : & \{u, (q, a), v\}_K, \{v, b, z\}_K \\ A_{b,t}!B_{b,t} & : & (\{v'\})\{u, a', v'\}_K, \{v', (q', b), z\}_K \end{array}$$

    or

$$\begin{array}{rcl} A_{b,t\$}?B_{b,t\$} & : & \{u, (q, a), v\}_K, \{v, (b, \$), z\}_K \\ A_{b,t\$}!B_{b,t\$} & : & (\{v'\})\{u, a', v'\}_K, \{v', (q', (b, \$)), z\}_K \end{array}$$

    or

$$\begin{array}{rcl} A_{\lambda,t}?B_{\lambda,t} & : & \{u, (q, (a, \$)), v\}_K \\ A_{\lambda,t}!B_{\lambda,t} & : & (\{v'\})\{u, a', v'\}_K, \{v', (q', (\Box, \$)), v\}_K \end{array}$$

    for any $b \in \Gamma$;

  - when a final configuration is reached, the secret is revealed:

$$\begin{array}{rcl} A_{q,a}?B_{q,a} & : & \{u, (q, a), v\}_K \\ A_{q,a}!B_{q,a} & : & x \end{array}$$

    or

$$\begin{array}{rcl} A_{q,a\$}?B_{q,a\$} & : & \{u, (q, (a, \$)), v\}_K \\ A_{q,a\$}!B_{q,a\$} & : & x \end{array}$$

    for any $q \in F$ and $a \in \Gamma$.

Let T be the set of all basic terms of the protocol. We will show that w is accepted by TM in time 2^n if and only if $\mathcal{P}_{TM,w}$ under (T,k)-runs is leaky, where k is some suitable chosen constant linear in n. Any computation of TM with at most 2^n steps can be simulated by a (T,k)-run of $\mathcal{P}_{TM,w}$. Moreover, if TM accepts w, then $\mathcal{P}_{TM,w}$ reveals the secret. Conversely, if a (T,k)-run ξ of the protocol reveals the secret, then there exists a computation of TM which accepts w. ξ cannot have more than 2^n+1 send events that generate new nonces because each send event generates at least one new nonce and the set of all non-constant nonces has cardinality $2^n + ||w||$ ($||w||$ distinct nonces are used to represent w). Therefore, the corresponding computation of TM cannot have more than 2^n steps.

The size of the protocol $\mathcal{P}_{TM,w}$ is polynomial in n. The protocol depends by TM, but TM is constant for any instance w of L. The set of nonces can be considered as an initial fragment of the set of natural numbers and, therefore, it can be specified by giving just a natural number (the cardinality of this set). As a conclusion, the protocol can be constructed in polynomial time with respect to $n = |w|$.

This shows that L is reducible to the initial secrecy problem for bounded protocols with freshness check. □

Remark 4.1.1 In the proof of Theorem 4.1.3, all the roles associated to transitions change the linking nonce v into a fresh one v'. This is crucial for a correct simulation of the Turing machine; otherwise, it might happen that the intruder sends two linked terms $\{u,(q,a),v\}_K, \{v,b,z\}_K$, where $\{u,(q,a),v\}_K$ represents the content of a tape cell at a given step, while $\{v,b,z\}_K$ represents the content of a (possible adjacent) cell at a different step. In this way, there can be leaky runs in protocol with no corresponding computation in the Turing machine.

4.1.3 The (Initial) Secrecy Problem for 1-Session Bounded Protocols

Another way to restrict protocols in order to get decidability of secrecy is to bound the number of sessions that can occur in any run of the protocol. The first significant results that obtain the NP-completness of initial secrecy problem with finite number of sessions and unbounded message length were given in [57, 3].

In what follows, we will show that the (initial) secrecy problem for bounded protocols with finite number of sessions is NP-complete.

Even if each run of a 1-session bounded protocol is based on exactly one substitution, this substitution is crucial in order to draw the correct conclusion about the secrecy problem. We illustrate this by considering the following 1-session bounded protocol without freshness check:

$$\begin{array}{lcl} A!B & : & (\{x, K\})\{x\}_K \\ B?A & : & \{x\}_K \\ B!A & : & (\{K'\})K' \\ A?B & : & K' \end{array}$$

In this protocol, A and B are constants, x is a nonce, and K and K' are distinct short-term keys. The 1-session run defined by the identity substitution is non-leaky. However, under the substitution that assigns K to K' and is the identity for all the other elements, the protocol is leaky.

We say that a message m is *derivable* from a state s of a protocol, and denote this by $s \vDash m$, if m can be obtained by applying analysis and synthesis rules to terms in s. The following result has been mentioned in [25] under the MSR formalism. We recall it here under the formalism used in our paper.

Lemma 4.1.2 Let $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$ be a 1-session (T, k)-bounded protocol, s_A be the state of an agent A in some run of $\mathcal{P}$, and m be a message of length at most k over T. Then, it is decidable in deterministic polynomial time with respect to $size(\mathcal{P})$ whether $s_A \vDash m$.

Proof Let $D = analz(s_A)$. Decompose m into pieces using analysis rules until either all pieces are in D, or at least one of them is not in D but it cannot be decomposed any further. In the first case $s_A \vDash m$, and in the second case $s_A \nvDash m$. The complexity of this procedure is polynomial in $size(\mathcal{P})$:

- D can be computed in $\mathcal{O}(size(\mathcal{P})^2)$ (s_A has at most $|w|$ messages of length at most k);
- m can be decomposed in $\mathcal{O}(size(\mathcal{P}))$, and
- checking whether all pieces in the decomposition of m are in D can be done in $\mathcal{O}(size(\mathcal{P})^3)$ ($|D| = \mathcal{O}(size(\mathcal{P})^2)$).

As a conclusion, the complexity of this procedure is $\mathcal{O}(size(\mathcal{P})^3)$. □

Theorem 4.1.4 The (initial) secrecy problem for 1-session bounded protocols with or without freshness check is in NP.

Proof Let $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$ be a 1-session (T, k)-bounded protocol. A *1-session sequence of $\mathcal{P}$ under a substitution σ* is a sequence ξ of events such that the following properties hold true:

- $|\xi| \leq |w|$;
- each event $e \in Events(\xi)$ has the form (r, σ, i) for some $r \in Roles(\mathcal{P})$;

- each event in $e \in Events(\xi)$ appears at most once in ξ and the order it has in its role (i.e., if $\xi = \xi' e \xi''$, then $LP(e) \subseteq Events(\xi')$).

The following non-deterministic algorithm decides the (initial) secrecy problem for 1-session bounded protocols with or without freshness check in polynomial time.

Algorithm A3
`input:` $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$ a 1-session (T,k)-bounded protocol;
`output:` "leaky protocol" if $\mathcal{P}$ has leaky 1-session (T,k)-runs;
`begin`
`guess` a 1-session sequence $\xi = e_1 \cdots e_n$ of $\mathcal{P}$ under some substitution σ;
`if` ξ is a 1-session run `then`
`begin`
`let` $s_0[e_1\rangle s_1[\cdots[e_n\rangle s_n$;
$Secret := \bigcup_{A \in Ho} Secret_A$;
`for` $i := 1$ `to` $n-1$ `do`
$Secret := Secret \cup ((\bigcup_{A \in Ho} analz(s_{iA})) - analz(s_{iI}))$;
`guess` $t \in T$;
`if` $t \in Secret \cap analz(s_{nI})$ `then` "leaky protocol";
`end`
`end.`

If the "`for`" statement is dropped, the algorithm will decide the initial secrecy problem for 1-session bounded protocols.

It is easy to see that the algorithm is correct. Its complexity can be determined as follows. Given a state s and an event e, one can check whether e is enabled at s in $\mathcal{O}(size(\mathcal{P})^3)$ (Lemma 4.1.2). Therefore, verifying whether a 1-session sequence is a run can be done in $\mathcal{O}(size(\mathcal{P})^4)$. The complexity of the `for` loop is $\mathcal{O}(size(\mathcal{P})^4)$, which gives the complexity of the entire algorithm. Therefore, the (initial) secrecy problem for 1-session bounded protocols with or without freshness check is in NP. □

To prove that the secrecy problem for 1-session bounded protocols with or without freshness check is NP-hard, a reduction from 3-SAT is exhibited. An instance of 3-SAT consists of:

- a set of boolean variables $\{x_1, \ldots, x_n\}$ and
- a boolean expression $E = C_1 \wedge \cdots \wedge C_m$, where each C_j, called *clause* is of the form
$$C_j = \alpha_{j,1} \vee \alpha_{j,2} \vee \alpha_{j,3}$$
and $\alpha_{j,l}$ is a *literal*, i.e. either x or $\neg x$, for some variable x.

E is *satisfiable* if it can be evaluated to the truth value true under some assignment. The 3-SAT problem consists in deciding whether or not a boolean expression E defined as above is satisfiable. It is well-known that this problem is NP-complete.

A reduction from 3-SAT was shown also in [57]. However, there are major differences between the reduction in [57] and our reduction. First, the "fake values" used in [57] to enforce each role to be applied completely are no longer used in our protocol because we consider 1-session runs that does not necessarely ends completely. Secondly, our reduction works for secrets generated in the course of runs and not only for initial secrets. Thirdly, the reduction in [57] is based on initial "temporary secrets" and initial "big secrets", and the protocols are considered leaky only if the intruder gets the initial big secrets (initial temporary secrets are needed only for technical reasons, they should be considered as secrets, but the protocol is not considered leaky if the intruder gets any of them). Under the formalism we consider, which is as general as possible, all secrets are equally treated (which is a natural approach). Therefore, there is no temporary secrets in our formalism and, as a conclusion, the reduction in [57] cannot be applied in this case. Our reduction considers the most general case of secrets generated in the course of runs.

Theorem 4.1.5 The (initial) secrecy problem for 1-session bounded protocols with or without freshness check is NP-hard.

Proof Let E be a 3-SAT instance as the one above. A 1-session (T, k)-bounded protocol $\mathcal{P}_E$ will be defined such that E is satisfiable if and only if the protocol $\mathcal{P}_E$ reveals secrets generated in the course of some run. The idea of the construction is to let the intruder to generate an assignment as a possible solution to the 3-SAT instance E. After generating such an assignment, the intruder will use the honest agents to check the truth value of each clause of E. If the assignment generated by the intruder is a solution, then a secret will be generated and revealed by some honest agent. The protocol is as follows:

- A, B, A_i, B_i, $A_{j,l}$, and $B_{j,l}$ are pairwise distinct agents, for any $1 \leq i \leq n$, $1 \leq j \leq m$, and $l = 1, 2, 3$. A_i and B_i correspond to the variable x_i, and $A_{j,l}$ and $B_{j,l}$ correspond to the literal $\alpha_{j,l}$;

- V_i is a long-term key shared by A_i, B_i, $A_{j,l}$, and $B_{j,l}$ such that x_i defines the literal $\alpha_{j,l}$ (that is, $\alpha_{j,l}$ is either x_i or $\neg x_i$), for any i, j, and l;

- $K_{j,A}$ is a long-term key shared by $A_{j,l}$ and A, for any j and l;

- K is a long-term key shared by all honest agents;

- $True$ and $False$ are two distinct constant nonces representing the truth values *true* and, respectively, *false*. They will be used by the intruder to generate an assignment. We also consider v and x two distinct nonces.

- the protocol actions are:

  – (variable assignment)

  $$\begin{array}{lcl} A_i?B_i & : & v \\ A_i!B_i & : & \{v\}_{V_i} \end{array}$$

  for each $1 \leq i \leq n$. When A_i receives (from the intruder) a truth value v, he assigns it to x_i by sending $\{v\}_{V_i}$. As there is exactly one value for each i, any 1-session run will ensure that all variables are instantiated in a non-redundant way;

  – (checking the truth value of clause C_j)

  $$\begin{array}{lcl} A_{j,l}?B_{j,l} & : & \{True\}_{V_i} \\ A_{j,l}!B_{j,l} & : & \{True\}_{K_{j,A}} \end{array}$$

  if $\alpha_{j,l} = x_i$, and

  $$\begin{array}{lcl} A_{j,l}?B_{j,l} & : & \{False\}_{V_i} \\ A_{j,l}!B_{j,l} & : & \{True\}_{K_{j,A}} \end{array}$$

  if $\alpha_{j,l} = \neg x_i$, for each j and l.

  A few words regarding these rules are in order. A clause C_j is satisfiable iff at least one of its literals is evaluated to the truth value $true$. Therefore, if an agent $A_{j,l}$ receives an assignment $\{True\}_{V_i}$ for a variable x_i and $\alpha_{j,l} = x_i$, then it returns $\{True\}_{K_{j,A}}$ because, from his point of view, C_j is evaluated to the truth value *true*.

  – (reveal the secret)

  $$\begin{array}{lcl} A?B & : & \{True\}_{K_{1,A}}, \ldots, \{True\}_{K_{m,A}} \\ A!B & : & (\{x\})\{x\}_K \\ A!B & : & x \end{array}$$

  When A receives the confirmation that all clauses were evaluated to the truth value true, then A generates a secret x, sends x encrypted by K, and then reveals x.

Let T be the set of all basic terms of the protocol $\mathcal{P}_E$. Then, it is easy to see that E is satisfiable if and only if there exists a 1-session (T, k)-run of $\mathcal{P}_E$ which reveals a secret generated in the course of the run, where k is some suitable chosen constant linear in m. Moreover, the size of $\mathcal{P}_E$ is polynomial in $max\{n, m\}$. Therefore, 3-SAT can be reduced in polynomial time to the secrecy problem.

In order to reduce 3-SAT to the initial secrecy problem what we have to do is to provide A in the protocol above with some initial secret x_0 and to replace the last two actions by "$A!B : x_0$". □

Corollary 4.1.4 The (initial) secrecy problem for 1-session bounded protocols with or without freshness check is NP-complete.

Table 1 below summarizes the main complexity results regarding the initial secrecy problem and the secrecy problem for bounded protocols obtained in previous sections of this chapter (X-c means that the problem is complete for the complexity class X, and "?" indicates an open problem). The NP-completeness of the initial secrecy problem for 1-session bounded protocols with freshness check follows from [57], and the NP-completeness of the initial secrecy problem for 1-session bounded protocols without freshness check was established in [25]. These results also follow from Corollary 4.1.4. Moreover, Corollary 4.1.4 proves the NP-completeness of the secrecy problem for 1-session bounded protocols with or without freshness check. The DEXP-complete result was proposed in [24] under the MSR formalism (Corollary 4.1.3 provides a simple proof under the formalism used in thesis) and the NEXP-complete results follow from Theorem 4.1.3.

Bounded Protocols	1-session		Multi-session	
	initial secrecy	secrecy	initial secrecy	secrecy
Freshness check	NP-c [57, 68]	NP-c [68]	NEXP-c [67, 68]	NEXP-c [67, 68]
No freshness check	NP-c [25, 68]	NP-c [68]	DEXP-c [24, 69]	?

Table 1: Complexity results for the initial secrecy problem and the secrecy problem for bounded protocols

4.1.4 Relationship with the MSR Formalism

The algorithm A1 in Section 4.1.1, proposed initialy in [25] under MSR formalism for modelling security protocols, was used to develop several complexity results for the initial secrecy problem [25]. Unfortunately, all the results based on this algorithm are wrong. In order to show that we recall first the MSR formalism, introduced in [23]. This formalism is based on:

- a *signature*, which specifies a set of *sorts* (for keys, messages, nonces etc.) together with function and predicate symbols (each symbol having associated a specific sort). Function symbols with no arguments are also called *constant symbols*;
- a set of *variables*, each of which having associated a sort;
- *terms*, which are defined as usual;
- *atomic formulas*, which are constructs of the form $P(t_1, \ldots, t_n)$, where P is a predicate symbol of sort $s_1 \cdots s_n$ and t_i is a term of sort s_i, for any i;
- *facts*, which are atomic formulas $P(t_1, \ldots, t_n)$, where all terms t_i are *ground terms* (i.e., variable-free terms);
- *states*, which are multisets of facts.
- *rules*, which are constructs of the form

$$F_1, \ldots, F_k \ \rightarrow \ \exists x_1, \ldots, \exists x_l.G_1, \ldots, G_p,$$

  where the F's and G's are atomic formulas and the x's are variables;

If S is a state,

$$r: \quad F_1, \ldots, F_k \ \rightarrow \ \exists x_1, \ldots, \exists x_l.G_1, \ldots, G_p$$

is a rule, and σ is a ground substitution such that $\sigma(x_i)$ is a new constant symbol (that is, not previously generated) and $\sigma(F_j) \in S$, for all i and j, then the rule r under substitution σ can be applied to S yielding a new state S'. This state is obtained from S by

$$S' = (S - \langle\langle \sigma(F_j) | 1 \leq j \leq k \rangle\rangle) \cup \langle\langle \sigma(G_j) | 1 \leq j \leq p \rangle\rangle,$$

where "$\langle\langle \cdots \rangle\rangle$" denotes multisets and the difference and union with multisets are defined as usual. We denote this by $S \stackrel{r,\sigma}{\Longrightarrow} S'$.

Existential quantifiers in the right hand side of rules, simply called *existentials*, capture the idea of "generation of new elements" (short-term keys, nonces etc.). For example, the new elements generated in the computation step $S \stackrel{r,\sigma}{\Longrightarrow} S'$ are $\sigma(x_1), \ldots, \sigma(x_l)$.

The MSR formalism has been extended in [25] with *tests for disequality*, which are constructs of the form "$t_1 \neq t_2$", where t_1 and t_2 are terms. These tests may be added only to the left hand side of rules. For example,

$$P(t_1), Q(t_2), t_1 \neq t_2 \ \rightarrow \ \exists x.R(x)$$

is such a rule. It can be applied to a state S under a substitution σ if the state S contains two facts $\sigma(P(t_1))$ and $\sigma(Q(t_2))$ such that $\sigma(t_1) \neq \sigma(t_2)$.

The mechanism based on existentials in the MSR formalism is equivalent to the freshness check mechanism used with the formalism in this thesis. Disequality tests add more power to the MSR formalism: a disequality test on a nonce means that the protocol compares a supposedly fresh nonce it receives against all other nonces that have been received, to be sure that it is actually fresh. Disequality tests do not have any equivalent in the formalism in this thesis.

In the MSR formalism the intruder is modeled by the following set of rules:

- rules for intercepting a message from the network and sending one to the network:

$$\begin{array}{lcl} N_S(x) & \rightarrow & D(x) \\ C(x) & \rightarrow & N_R(x) \end{array}$$

- rules for decomposition, decryption and memoring a message:

$$\begin{array}{lcl} D(\langle x, y\rangle) & \rightarrow & D(x), D(y) \\ D(enc(K,x)), KP(K,K'), M(K') & \rightarrow & D(x), M(enc(K,x)), \\ & & KP(K,K'), M(K') \\ D(x) & \rightarrow & M(x) \end{array}$$

- rules for transforming a fact from intruder's memory into a composable one, pairing, encryption and generating new data (nonces or short-term keys):

$$\begin{array}{lcl} M(x) & \rightarrow & C(x), M(x) \\ C(x), C(y) & \rightarrow & C(\langle x, y\rangle) \\ M(K), C(x) & \rightarrow & C(enc(K,x)), M(K) \\ & \rightarrow & \exists x.M(x) \end{array}$$

In the rules above $\langle\ ,\ \rangle$ and $enc(\ )$ are functions for pairing, respectively, encryption. The meaning of the predicates above is as follows. N_S is a predicate for a message sent on the network, and N_R is a predicate for a message received from the network. D means that a message is decomposable, M that a message is stored in the intruder's memory, C that a message is composable. $KP(K, K')$ is used to point out that (K, K') is a (public, private) key pair (to model the intruder we discuss only public-key encryption).

An "intruder with no existentials" in the MSR formalism means that the intruder does not generate new elements and, therefore, it is equivalent to an *intruder without generation* (i.e., $\mathcal{N}_I \cup \mathcal{K}_{0,I} = \emptyset$) in this thesis. But an "intruder

with existentials" in the MSR formalism, which means that the intruder can generate new elements from an arbitrary (possible infinite) set of elements does not have any equivalent in the formalism used in thesis because there is no freshness check mechanism for the elements generated by the intruder in the formalism in this thesis. An "intruder with bounded existentials" in the MSR formalism means that the intruder can generate new elements from a finite set of elements; this concept still does not have any equivalent in the formalism used in thesis for the same reason as that above. A "protocol with bounded existentials" in [25] means that the honest agents can generate new elements from a given finite set.

As a conclusion of our discussion above we can say that any protocol in the formalism in this thesis for which the intruder is without generation can be easily translated into the MSR formalism:

- if the protocol is without freshness check then the corresponding protocol in the MSR formalism does not have existentials and disequality tests, and the intruder is with no existentials;
- if the protocol is with freshness check then the corresponding protocol in the MSR formalism may have existentials but it does not have disequality tests, and the intruder is with no existentials.

Example 4.1.3 The Needham-Schroeder public-key protocol presented in the Example 2.2.1 under the formalism used in thesis is translated in the MSR formalism as follows:

- (Initialization Theory)

$$\begin{array}{rcl} & \rightarrow & \exists K, K'.Ho(K, K'), KP(K, K') \\ Ho(K, K') & \rightarrow & Pub(K), Ho(K, K') \end{array}$$

  The $Ho(K, K')$ predicate indicates an honest agent with his (public, private) key pair and $Pub(K)$ predicate means that an agent makes available his public key K. The initialization phase allows to create an unbounded number of honest agents.

- (Role Generation Theory)

$$\begin{array}{rcl} Ho(K, K') & \rightarrow & Ho(K, K'), A_0(K) \\ Ho(K, K') & \rightarrow & Ho(K, K'), B_0(K) \end{array}$$

  A_0 and B_0 are the initial role states for the initiator role, respectively, the responder role, parametrized by the public key of the agent that plays that role. This rules allow an unbounded number of sessions to be started.

- (Protocol Role Theory)

$$
\begin{array}{lcl}
A_0(K), Pub(K') & \rightarrow & \exists x.A_1(K,K',x), \\
 & & N_{S1}(enc(K',\langle x,K\rangle)), \\
 & & Pub(K') \\
B_0(K), N_{R1}(enc(K,\langle x,K'\rangle)), Pub(K') & \rightarrow & \exists y.B_1(K,K',x,y), \\
 & & N_{S2}(enc(K',\langle x,y\rangle)), \\
 & & Pub(K') \\
A_1(K,K',x), N_{R2}(enc(K,\langle x,y\rangle)) & \rightarrow & A_2(K,K',x,y), \\
 & & N_{S3}(enc(K',y)) \\
B_1(K,K',x,y), N_{R3}(enc(K,y)) & \rightarrow & B_2(K,K',x,y)
\end{array}
$$

  A_i and B_i with $i = 1, 2$ are role states for initiator role, respectively, for responder role. N_{Ri} means that the i-th message is received from the network, and N_{Si} that the i-th message is sent to the network

For this protocol, we add a rule to the intruder that allows him to learn the public key of an honest agent:

$$Pub(K) \rightarrow M(K), Pub(K).$$

Now, we obtain the following results (in the MSR formalism).

Corollary 4.1.5 The following four problems, in the MSR formalism, are NEXPTIME-complete:

1. the initial secrecy problem for protocols with bounded length messages, bounded existentials, without disequality tests, and an intruder with no existentials;

2. the initial secrecy problem for protocols with bounded length messages, bounded existentials, with disequality tests, and an intruder with no existentials;

3. the initial secrecy problem for protocols with bounded length messages, bounded existentials, without disequality tests, and an intruder with existentials.

4. the initial secrecy problem for protocols with bounded length messages, bounded existentials, with disequality tests, and an intruder with bounded existentials.

Proof First, we transform the third problem into an equivalent problem by replacing the "intruder with existentials" by an intruder with no existentials and which has a distinguished nonce n and a distinguished short-term key K in the initial state (as in [25]). This transformation is simply performed by removing intruder's rule for generating new data and adding the ground facts $M(n)$ and $M(K)$ to the initial state. These ground facts say that the intruder knows initially the nonce n and the short-term key K, and can use them during any computation. The protocol such obtained is equivalent to the original one (which is assumed to be a protocol with bounded length messages, bounded existentials, without disequality tests, and an intruder with existentials) in the sense that it does not change the status of the initial secrecy problem of the original protocol. This is simply obtained just because of the absence of disequality tests.

With the transformation above, every instance of any problem in the corollary defines a finite set of ground facts and ground instances of the rules. Now, to show that all these problems are in NEXPTIME, one can easily develop a non-deterministic version of the algorithm in [25] (page 284). This algorithm works similarly to our algorithm A2: it guesses a sequence of ground instances of the protocol's and the intruder's rules, and then verifies that the sequence is a run. If it is, the algorithm checks whether the intruder gets any initial secrets.

In order to show that these problems are NEXPTIME-complete it is sufficient to show that they include a NEXPTIME-complete subproblem. In the proof of Theorem 4.1.3 we may consider that the protocol satisfies $\mathcal{N}_I \cup \mathcal{K}_{0,I} = \emptyset$ (i.e., the intruder is without generation). As a conclusion, the initial secrecy problem for bounded protocols with freshness check and an intruder without generation is NEXPTIME-complete as well. Every instance of this problem can be encoded in polynomial time into an instance of the initial secrecy problem for protocols with bounded length messages, bounded existentials, without disequality tests, and an intruder with no existentials (in the MSR formalism). Therefore, the initial secrecy problem for protocols with bounded length messages, bounded existentials, without disequality tests, and an intruder with no existentials includes a NEXPTIME-complete problem, which shows that it is NEXPTIME-complete too.

As the second, the third, and the fourth problem includes the first problem as a subproblem, it follows that these problems are NEXPTIME-complete too. □

In [25], several DEXPTIME-completeness results have been obtained for classes of bounded protocols in the *restricted form*. This form avoids (direct or indirect) recursion. More precisely, in such a protocol the role of any agent A contains only rules of the form

$$A_i(\ldots), N_{R_j}(\ldots) \rightarrow \exists \ldots A_k(\ldots) N_{S_l}(\ldots)$$

where $A_1, \ldots, A_m$ is a finite list of predicates defining the role states, $N_{R_1}, \ldots, N_{R_n}$ and $N_{S_1}, \ldots, N_{S_n}$ are finite lists of network predicates, $i < k \leq m$ and $j < l \leq n$. When we do not use the terminology "restricted form" we implicitly understand that the protocol is not in the restricted form. Moreover, sometimes we will say "protocol in the unrestricted form" in order to emphasize that the protocol is not in the restricted form.

Corollary 4.1.6 The following four problems, in the MSR formalism, are NEXPTIME-complete:

1. the initial secrecy problem for protocols in the restricted form, with bounded length messages, bounded existentials, without disequality tests, and an intruder with no existentials;

2. the initial secrecy problem for protocols in the restricted form, with bounded length messages, bounded existentials, with disequality tests, and an intruder with no existentials;

3. the initial secrecy problem for protocols in the restricted form, with bounded length messages, bounded existentials, without disequality tests, and an intruder with existentials.

4. the initial secrecy problem for protocols in the restricted form, with bounded length messages, bounded existentials, with disequality tests, and an intruder with bounded existentials.

Proof It follows the same line as the proof of Corollary 4.1.5 but with the remark that one can easily translate the protocol in the proof of Theorem 4.1.3 into a protocol in the MSR formalism which is in the restricted form, with bounded length messages, bounded existentials, without disequality tests, and an intruder with no existentials. □

Corollary 4.1.6 corrects three false statements in [25] according to which the initial secrecy problem for protocols in the restricted form, with bounded length messages, bounded existentials, with or without disequality tests, and an intruder with no existentials, and the initial secrecy problem for protocols in the restricted form, with bounded length messages, bounded existentials, without disequality tests, and an intruder with existentials, are DEXPTIME-complete. The source of these mistakes was that the authors in [25] claimed that the algorithm A1 works in the case of existentials because, as the authors said, "... bounding the number of protocol existentials means that we can generate all the existentials used by all runs of the protocol during the initialization phase". However, this is false and in order to see that one can easily translate the protocol in Example 4.1.1 into a protocol in the restricted form in the MSR formalism:

$$
\begin{array}{lcl}
 & \rightarrow & A_0(x_0) \\
 & \rightarrow & B_0() \\
 & \rightarrow & C_0() \\
 & \rightarrow & D_0() \\
A_0(x_0) & \rightarrow & \exists K.A_1(x_0,K), N_{S1}(x_0,K) \\
B_0(), N_{R1}(x_0,K) & \rightarrow & B_1(x_0,K) \\
C_0() & \rightarrow & \exists K.C_1(K), N_{S2}(K) \\
D_0(), N_{R2}(K) & \rightarrow & D_1(K)
\end{array}
$$

By the role generation theory (the first four rules) the ground facts $A_0(x_0)$, $B_0()$, $C_0()$ and $D_0()$ can be "pumped" unboundedly. Therefore, if all ground instances of the seventh rule are applied first, the algorithm A1 generates a non-leaky maximal run and concludes that the protocol is not leaky, while the protocol is leaky.

We would like to point out that the status of the initial secrecy problem for protocols in the restricted form, with bounded length messages, no existentials, and an intruder with no existentials was correctly identified as being DEXPTIME-complete in [24].

Next, we establish an undecidability result.

Theorem 4.1.6 The initial secrecy problem for protocols in the unrestricted form, with bounded length messages, no existentials, disequality tests and an intruder with existentials is undecidable.

Proof We will reduce the halting problem for deterministic Turing machines to this problem. Given a deterministic Turing machine and a input for this machine we will construct a protocol with bounded length messages, no existentials, disequality tests and an intruder with existentials such that the Turing machine halts on the input iff the protocol is leaky. The construction follows the same idea as in the proof of Theorem 4.1.3 adapted to the MSR formalism (and except for the fact that in this case the Turing machines are deterministic).

Let $TM = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ be a deterministic Turing machine (Q, Σ, Γ, q_0, $\Box$, and F are as in the proof of Theorem 4.1.3, and δ is a partial function from $Q \times \Gamma$ into $Q \times \Gamma \times \{-1, 1\}$) and an input w for TM. We construct a protocol $\mathcal{P}_{TM,w}$ with bounded length messages, no existentials, disequality tests and an intruder with existentials such that TM halts on w if and only if $\mathcal{P}_{TM,w}$ is leaky. First, we may assume that all the elements in $Q \cup \Gamma$ are constants of sort nonce. We consider three distinguished constants of sort nonce, \$, $init$, and $\perp$, a long-term key K, a secret information x_0 known only to the honest agents, the variables x, y, z, u, v, v', s of sort nonce and the predicates A_q, $T_{q,a}$, $T_{q,a\$}$, L, N_{Ri}, and N_{Si}, for any $i = 1, 2, 3$, state q and tape symbol a. The meaning of these predicates is as follows. The predicate $A_q(a)$ initializes the rules which simulate the transition

$\delta(q,a)$. N_{Ri} and N_{Si} with $i = 1,2,3$ are network predicates for receive actions, respectively, send actions. L is used to define a list of distinct nonces. The first element of the list is $init$ and the last element is $\perp$. $L(x,y)$ means that x and y are linked in the list. $T_{q,a}(x,y,u,s,z)$ and $T_{q,a\$}(x,y,u,s,z)$ means that the nonce x, which is an element of the list, is to be compared against the nonce y and the nonces u,s,z will be used to send the new configuration after the transition $\delta(q,a)$. These predicates will be used in conjunction with the predicate L and disequality tests: when a nonce is received, it is compared with all nonces in the list; if it is different than all nonces in the list then it is appended to the list. Only nonces in this list will be used to simulate the computation of the Turing machine starting with w.

To simplify the notation we will use $\{\alpha\}_K$ instead of $enc(K,\alpha)$ and α,β instead of $\langle\alpha,\beta\rangle$ as it is used in [25].

A configuration $(q, a_1\cdots a_n, a, b_1\cdots b_m)$ of TM is represented in the protocol $\mathcal{P}_{TM,w}$ by a sequence of the form

$$\{u_1,a_1,u_2\}_K,\ldots,\{u_n,a_n,u_{n+1}\}_K,\{u_{n+1},(q,a),u_{n+2}\}_K,$$
$$\{u_{n+2},b_1,u_{n+3}\}_K,\ldots,\{u_{n+m+1},(b_m,\$),u_{n+m+2}\}_K,$$

where $u_1,\ldots,u_{n+m+2}$ are distinct nonces.

For $w = a_1\cdots a_n$, the initial state of the protocol consists of the following ground facts:

- $N_{S1}(\{u_1,(q_0,(\Box,\$)),u_2\}_K)$, $L(init,u_1)$, $L(u_1,u_2)$, $L(u_2,\perp)$, and $A_{q_0}(\Box)$, if $n=0$;
- $N_{S1}(\{u_1,(q_0,(a_1,\$)),u_2\}_K)$, $L(init,u_1)$, $L(u_1,u_2)$, $L(u_2,\perp)$, and $A_{q_0}(a_1)$, if $n=1$;
- $N_{S1}(\{u_1,(q_0,a_1),u_2\}_K)$, $N_{S1}(\{u_2,a_2,u_3\}_K),\ldots,N_{S1}(\{u_{n-1},a_{n-1},u_n\}_K)$, $N_{S1}(\{u_n,(a_n,\$),u_{n+1}\}_K)$, $L(init,u_1)$, $\ldots$, $L(u_n,u_{n+1})$, $L(u_{n+1},\perp)$, and $A_{q_0}(a_1)$, if $n>1$.

The protocol rules are:

- a transition $\delta(q,a)=(q',a',-1)$ is simulated by:

$$\begin{array}{lcl}
A_q(a), N_{R1}(\{u,s,v\}_K,\{v,(q,a),z\}_K,v') & \rightarrow & T_{q,a}(init,v',u,s,z)\\
T_{q,a}(x,v',u,s,z), L(x,y), x\neq v', y\neq\perp & \rightarrow & L(x,y), T_{q,a}(y,v',u,s,z)\\
T_{q,a}(x,v',u,s,z), L(x,\perp), x\neq v', \perp\neq v' & \rightarrow & L(x,v'), L(v',\perp),\\
 & & N_{S2}(\{u,(q',s),v'\}_K,\{v',a',z\}_K),\\
 & & A_{q'}(s)
\end{array}$$

or

$$\begin{array}{lcl}
A_q(a), N_{R1}(\{u,s,v\}_K,\{v,(q,(a,\$)),z\}_K,v') & \rightarrow & T_{q,a\$}(init,v',u,s,z)\\
T_{q,a\$}(x,v',u,s,z), L(x,y), x\neq v', y\neq\perp & \rightarrow & L(x,y), T_{q,a\$}(y,v',u,s,z)\\
T_{q,a\$}(x,v',u,s,z), L(x,\perp), x\neq v', \perp\neq v' & \rightarrow & L(x,v'), L(v',\perp),\\
 & & N_{S2}(\{u,(q',s),v'\}_K,\{v',(a',\$),z\}_K),\\
 & & A_{q'}(s)
\end{array}$$

- a transition $\delta(q,a) = (q', a', 1)$ is simulated by:

$$\begin{array}{lcl} A_q(a), N_{R1}(\{u,(q,a),v\}_K, \{v,s,z\}_K, v') & \rightarrow & T_{q,a}(init, v', u, s, z) \\ T_{q,a}(x, v', u, s, z), L(x,y), x \neq v', y \neq \bot & \rightarrow & L(x,y), T_{q,a}(y, v', u, s, z) \\ T_{q,a}(x, v', u, s, z), L(x,\bot), x \neq v', \bot \neq v' & \rightarrow & L(x, v'), L(v', \bot), \\ & & N_{S2}(\{u, a', v'\}_K, \{v', (q', s), z\}_K), \\ & & A_{q'}(s) \end{array}$$

or

$$\begin{array}{lcl} A_q(a), N_{R1}(\{u,(q,a),v\}_K, \{v,(s,\$),z\}_K, v') & \rightarrow & T_{q,a\$}(init, v', u, s, z) \\ T_{q,a\$}(x, v', u, s, z), L(x,y), x \neq v', y \neq \bot & \rightarrow & L(x,y), T_{q,a\$}(y, v', u, s, z) \\ T_{q,a\$}(x, v', u, s, z), L(x,\bot), x \neq v', \bot \neq v' & \rightarrow & L(x, v'), L(v', \bot), \\ & & N_{S2}(\{u, a', v'\}_K, \{v', (q', (s,\$)), z\}_K), \\ & & A_{q'}(s) \end{array}$$

or

$$\begin{array}{lcl} A_q(a), N_{R2}(\{u,(q,(a,\$)),v\}_K, v') & \rightarrow & T_{q,a\$}(init, v', u, \Box, z) \\ T_{q,a\$}(x, v', u, \Box, z), L(x,y), x \neq v', y \neq \bot & \rightarrow & L(x,y), T_{q,a\$}(y, v', u, \Box, z) \\ T_{q,a\$}(x, v', u, \Box, z), L(x,\bot), x \neq v', \bot \neq v' & \rightarrow & L(x, v'), L(v', \bot), \\ & & N_{S2}(\{u, a', v'\}_K, \{v', (q', (\Box,\$)), z\}_K), \\ & & A_{q'}(\Box) \end{array}$$

- when a final configuration is reached, the secret is revealed:

$$A_q(s), N_{R3}(\{u,(q,s),v\}_K) \rightarrow N_{S3}(x_0)$$

or

$$A_q(s), N_{R3}(\{u,(q,(s,\$)),v\}_K) \rightarrow N_{S3}(x_0)$$

for any $q \in F$.

It is clear that any computation of TM on w can be simulated by a run of $\mathcal{P}_{TM,w}$. Moreover, if TM halts on w, then the intruder learns x_0 and $\mathcal{P}_{TM,w}$ is leaky. Conversely, if a run of the protocol reveals the secret x_0, then TM halts on w. □

Remark 4.1.2 The proof of Theorem 4.1.6 does not work for protocols in the restricted form because the list of nonces is recursively processed and this cannot be done with rules in the restricted form (which avoid recursion).

Since the problem in Theorem 4.1.6 is a subproblem of the initial secrecy problem for protocols in the unrestricted form, with bounded length messages, bounded existentials, disequality tests and an intruder with existentials, we have the following corollary:

Corollary 4.1.7 The initial secrecy problem for protocols in the unrestricted form, with bounded length messages, bounded existentials, disequality tests and an intruder with existentials is undecidable.

We emphasize that the protocols in Corollary 4.1.7 are not necessarily in the restricted form. If we add this constraint too, then we reach the problem left open in [25] (the top row of Table 9, page 282) for which we do not have yet any solutions.

We also emphasize that if we bound the intruder's existentials in Corollary 4.1.7, then the problem becomes NEXPTIME-complete (see Corollary 4.1.5(4)).

The table below summarizes the results from this section that correct the results from Table 9 [25]:

		Unbounded # roles, Bounded $\exists$	
		restricted form	unrestricted form
I with $\exists$	$\neq$	?	Undecidable
	$=$	NEXPTIME-complete	NEXPTIME-complete
I no $\exists$	$\neq$	NEXPTIME-complete	NEXPTIME-complete
	$=$	NEXPTIME-complete	NEXPTIME-complete

Table 2: Complexity results for bounded protocols under the MSR formalism [67, 68]

4.2 Normal Protocols

Another way to get decidability of the secrecy problem is to define an equivalence relation on terms so that arbitrarily large terms over a finite subset $T \subseteq \mathcal{T}_0$ can be reduced to "small" terms. Such an equivalence relation was proposed in [53] but, unfortunately, the results developed in that paper are wrong. In what follows we will present this equivalence relation and show how it can be used to get decidability of secrecy. Moreover, we will give an example that shows why the results in [53] are wrong.

First, given a term t and a key K we define an unary operation on terms, denoted t_{-K}, called *encryption with key K removal*, as follows:

- $t_{-K} = t$, for any $t \in \mathcal{T}_0$;
- $(t, t')_{-K} = (t_{-K}, t'_{-K})$;
- $(\{t\}_{K'})_{-K} = \begin{cases} t_{-K}, & \text{if } K' = K \\ \{t_{-K}\}_{K'}, & \text{otherwise} \end{cases}$

Thus, t_{-K} is the term t with all encryptions by the key K removed.

Define now the binary relation $\equiv$ as being the smallest binary relation on terms which satisfies the following properties:

(A1) $t \equiv t$;

(A2) $(t, t) \equiv t$;

(A3) $(t, t') \equiv (t', t)$;

(A4) $(t, (t', t'')) \equiv ((t, t'), t'')$;

(A5) $\{t\}_K \equiv \{t_{-K}\}_K$, for any key K ;

(R1) (closure under symmetry) $\dfrac{t \equiv t'}{t' \equiv t}$;

(R2) (closure under transitivity) $\dfrac{t \equiv t',\ t' \equiv t''}{t \equiv t''}$;

(R3) (closure under pairing) $\dfrac{t_1 \equiv t_1',\ t_2 \equiv t_2'}{(t_1, t_2) \equiv (t_1', t_2')}$;

(R4) (closure under encryption) $\dfrac{t \equiv t'}{\{t\}_K \equiv \{t'\}_K}$, for any key K.

This equivalence relation ensures two main things:

- lists of terms are transformed into sets of terms (axioms (A2) and (A4));
- if we consider a finite number of keys, the depth of the encryption operator is bounded, because each key is used for encryption at most once in an encrypted term (axiom (A5)).

For a protocol signature $\mathcal{S}$ and two substitutions σ and σ', we say that $\sigma \equiv \sigma'$ if $\sigma(x) \equiv \sigma'(x)$, for all $x \in \mathcal{T}_0$. Moreover, for two events (r, σ, i) and (r', σ', i') we say that $(r, \sigma, i) \equiv (r', \sigma', i')$ if $r = r'$, $i = i'$, and $\sigma \equiv \sigma'$. Extend further this notation to sequences of events $\xi = e_1 \cdots e_k$ and $\xi' = e_1' \cdots e_k'$, and we say that $\xi \equiv \xi'$ if $e_i \equiv e_i'$, for all $i \leq k$.

If there exists a proof of $t \equiv t'$ which does not use (A2) and (A5), we will say that $t \equiv_1 t'$.

Definition 4.2.1 (1) A term t is called a *redex* if it has a subterm of the form (t', t') or of the form $\{t'\}_K$ with K encrypting in t'.

(2) A term t is called *normal* if it is not $\equiv_1$-equivalent with any redex.

As we can see, a redex is a term that can be reduced by (A2) or (A5); a normal term cannot be further reduced by (A2) and (A5).

The following property of $\equiv$ can be easily proved.

Lemma 4.2.1 Let $\mathcal{S}$ be a protocol signature and $T \subseteq \mathcal{T}_0$ a finite set. The set of all normal terms built upon T is finite.

An important consequence of the result above is that there exists an upper bound $B(T)$ for the size of all normal terms.

Definition 4.2.2 An event $e = (r, \sigma, i)$ of a protocol $\mathcal{P}$ is called *normal* if $t(e)$ is normal. A sequence of events ξ of $\mathcal{P}$ is called *normal* if all events appearing in it are normal.

Directly from the definition of $\equiv$ we obtain

$$analz(T' \cup \{t\}) \cap T = analz(T' \cup \{t'\}) \cap T,$$

for any $t \equiv t'$ and T' a set of terms built upon T, which leads to:

Lemma 4.2.2 If $\mathcal{P}$ is a protocol and ξ and ξ' are runs of $\mathcal{P}$ such that $\xi \equiv \xi'$, then ξ is leaky iff ξ' is leaky.

In what follows we define *normal protocols* in a different way from those defined in [53]. The results obtained using the definition from [53] are wrong as we will show later.

Definition 4.2.3 Let $\mathcal{P}$ be a protocol and ξ a run of $\mathcal{P}$. We say that ξ is an $\equiv$-*normal run* of $\mathcal{P}$ if there exists a normal run ξ' of $\mathcal{P}$ such that $\xi \equiv \xi'$.

When for a protocol $\mathcal{P}$ only $\equiv$-normal runs are considered we will say that it is a *protocol under* $\equiv$*-normal runs* or a *normal protocol*. The secrecy problem for normal protocols is formulated with respect to $\equiv$-normal runs only.

Theorem 4.2.1 The secrecy problem for normal protocols over finite sets of basic terms is decidable.

Proof Let $\mathcal{P}$ be a normal protocol over a finite set T of basic terms. Because any run of $\mathcal{P}$ is equivalent to a normal run of $\mathcal{P}$, following Lemma 4.2.2, verifying that $\mathcal{P}$ has a leaky run can be reduced to verify that $\mathcal{P}$ has a leaky normal run. Normal runs of $\mathcal{P}$ are $(T, B(T))$-runs and, therefore, $\mathcal{P}$ is a $(T, B(T))$-bounded protocol. The theorem follows now from the results obtained for bounded protocols. □

Now, let us explain why the results in [53] are wrong. A pair of actions $A!B : (M)t$ and $B?A : t$ is called a *matching send-receive pair*.

Definition 4.2.4 A sequence of actions $w = a_1 \cdots a_l$ of a protocol $\mathcal{P}$ is called *send-admissible* if any send action a_i, $1 \leq i \leq l$, is enabled at the state s_{i-1}, where $s_0[a_1\rangle \cdots [a_{i-1}\rangle s_{i-1}$.

In that paper, the authors say that a protocol $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w = a_1 b_1 \cdots a_l b_l)$ is normal if $t(a_i)$ is normal for each $1 \leq i \leq l$, where w is a send-admissible sequence of matching send-receive pairs $a_i b_i$ of actions. Then, they claimed that any leaky run of a normal protocol is equivalent to a leaky normal run of the same protocol. However, this claim is false. For example, if we consider the protocol

$$\begin{array}{lcl} A!B & : & (\{u\})u \\ B?A & : & u \\ B!C & : & \{u\}_K \\ C?B & : & \{u\}_K \\ C!D & : & (\{v\})v \\ D?C & : & v \\ D!E & : & K \\ E?D & : & K \end{array}$$

where K is long-term key shared by all honest participants, then we can see that it is a normal protocol in the sense that each term in the protocol is normal.

Consider now five substitutions:

- σ_1 and σ_5 are the identity functions;
- $\sigma_2(u) = \{n_I\}_{K_I}$, where $n_I \in \mathcal{N}_I$, $K_I \in \mathcal{K}_{0,I}$ and it is the identity for all the other elements;
- $\sigma_3 = \sigma_2$;
- $\sigma_4(v) = \{\{\{n_I\}_{K_I}\}_K\}_{K_I}$, where $n_I \in \mathcal{N}_I$, $K_I \in \mathcal{K}_{0,I}$ and it is the identity for all the other elements.

Applying $w|_A$ under σ_1, $w|_B$ under σ_2, $w|_C$ under σ_3, $w|_D$ under σ_4, and $w|_E$ under σ_5, we obtain the following leaky run ξ (for simplicity, it is written in the protocol notation):

$$\begin{array}{lcl} A!B & : & (\{u\})u \\ B?A & : & \{n_I\}_{K_I} \\ B!C & : & \{\{n_I\}_{K_I}\}_K \\ C?B & : & \{\{n_I\}_{K_I}\}_K \\ C!D & : & (\{v\})v \\ D?C & : & \{\{\{n_I\}_{K_I}\}_K\}_{K_I} \\ D!E & : & K \\ E?D & : & K \end{array}$$

It is easy to see that there is no leaky normal run equivalent to ξ because the term in the sixth event of the normal run should be $\{\{n_I\}_K\}_{K_I}$. However, the intruder can neither generate nor obtain the term $\{n_I\}_K$.

We close the section by recalling another approach proposed in [62]. In that paper, normal events are defined as events $e = (r, \sigma, i)$ for which $\sigma(x)$ is normal for all $x \in \mathcal{T}_0$. Under this definition for normal events, it can be shown that secrecy is decidable for normal protocols over finite sets of basic terms.

Summing up, decidability of secrecy for normal protocols requires finitely many nonces but leaves the possibility for arbitrary-length messages.

4.3 Tagged Protocols

Tagging is a syntactic operation done on the specification of the protocol. This operation consists in adding message identifiers, also called *tags*, to some syntactic constructions (terms) in the protocol. Tagging schemes has been first proposed in [33], later, other tagging schemes have been also proposed in [55, 7]. The tagging schemes from [33, 7] are'nt costly because they use only a finite number of tags (no new tags are generated during the execution of the protocol), while the tagging scheme from [55] is more expensive requireing the generation of nonces.

Tagging and Type Flaw Attacks In what follows we will describe the main idea of the tagging scheme from [33] that is used to prevent *type flaw attacks*. *A type flaw attack* is an attack where a field that was originally intended to have one type is interpreted as having another type. The type flaw attack on Π^1 Woo and Lam authentication protocol explained in Chapter 1, can be avoided if supplementary information about the structure and the type of messages is sent along with the messages. For example, we may encode the types used in protocol as follows:

$$\begin{array}{lcl} nonce & = & 1 \\ agent & = & 2 \\ 3-tuple & = & 3 \\ (agent, agent, nonce)_{secret} & = & 4 \\ (agent, agent, enc\_term)_{secret} & = & 5 \end{array}$$

Then, Π^1 Woo and Lam protocol under the tagging scheme from [33] gets the form:

$$
\begin{array}{lll}
1. & A \rightarrow B & : \ (2, A) \\
2. & B \rightarrow A & : \ (1, x) \\
3. & A \rightarrow B & : \ (4, \{(3, (2, A), (2, B), (1, x))\}_{K_{AS}}) \\
4. & B \rightarrow S & : \ (5, \{(3, (2, A), (2, B), \\
 & & \qquad (4, \{(3, (2, A), (2, B), (1, x))\}_{K_{AS}}))\}_{K_{BS}}) \\
5. & S \rightarrow B & : \ (4, \{(3, (2, A), (2, B), (1, x))\}_{K_{BS}})
\end{array}
$$

Tagging is done associating to each field its intended type. For example, we will write $(2, A)$ to represent that A is tagged in such a way to indicate that he is intended an agent. Respectively, we will write $(1, x)$ to represent that x is tagged to indicate that it is intended a nonce. We will write

$$(4, \{(3, (2, A), (2, B), (1, x))\}_{K_{AS}})$$

to indicate the encryption with a secret key of a 3-tuple formed by two agents and one nonce.

A message is correctly tagged if all its fields are tagged with tags that corresponds to their true types. When an honest agent receives a message it will check the correctness of tags as long as possible (i.e. those tags that are not inside an encryption that the agent can't decrypt). The intruder is not forced to tag the fields correctly, i.e. he can associate arbitrarely tags to accesible messages.

If we adopt this tagging scheme, then the intruder cannot mount any longer the type flaw attack on Π^1 Woo and Lam protocol, presented in Chapter 1. For example, the intruder cannot return $(1, x)$ in the third message, but he can re-tag the message in the form that B is expecting it, i.e., $(4, x)$. In this case, the message at step 4 will be:

$$(5, \{(3, (2, A), (2, B), (4, x))\}_{K_{BS}})$$

However, this message cannot be replayed at step 5 by the intruder, because B is expecting a message where the third field inside the encryption is tagged as being a nonce. The intruder can change the outer most tag to create

$$(4, \{(3, (2, A), (2, B), (4, x))\}_{K_{BS}})$$

but it cannot change the inner tag without access to the secret key. Therefore, the attack is prevented.

In [33], using the strand space formalism [66], it is shown that an attack upon a protocol under this tagging scheme can be transformed into an essentially similar attack upon the protocol in which all messages are correctly tagged. Ill-tagged messages do not affect the agent's behavior in any essential way, and so might as well be replaced by messages that do have the correct tags. Therefore, the tagging scheme in [33] prevents all the type flaw attacks.

Tagging Enforces Termination In [7], a tagging scheme is used to obtain termination of a resolution-based verification algorithm for secrecy. With this method, protocols and the intruder's actions are translated into Horn clauses. The algorithm has two phases: the first uses the resolution to complete a given set of clauses and the second uses backward depth-first search to verify that a fact that breach secrecy is derivable from the set of clauses. If the fact can not be derived then the protocol preserves secrecy.

For instance, Π^1 Woo and Lam protocol under the tagging scheme from [7] is as follows:

$$\begin{array}{lcl} 1. & A \rightarrow B & : \quad A \\ 2. & B \rightarrow A & : \quad x \\ 3. & A \rightarrow B & : \quad \{c_1, A, B, x\}_{K_{AS}} \\ 4. & B \rightarrow S & : \quad \{c_2, A, B, \{c_1, A, B, x\}_{K_{AS}}\}_{K_{BS}} \\ 5. & S \rightarrow B & : \quad \{c_3, A, B, x\}_{K_{BS}} \end{array}$$

There is a significant difference between this tagging scheme and the one in [33]. First of all, this tagging scheme applies only to encrypted terms, but the tagging scheme in [33] applies to any element: agent, nonce, encrypted term, pair etc. Using the tagging scheme from [33] two different encrypted terms with the same structure (see message 3 and 5 from Π^1 Woo and Lam protocol) are tagged in the same way, but this is not the case for the tagging scheme in [7]. This tagging scheme avoides the eventualy loops (as in the Needham-Schroeder secret-key protocol [48]) because different constants cannot be unified.

A Tagging Scheme for unbounded nonces and unbounded-length messages Tagging schemes can not only be used to prevent type flaw attacks [33], but also to obtain classes of protocols for which secrecy is decidable. Such a tagging scheme has been proposed in [55]. The need of using this tagging scheme come from analysis of undecidability proofs. The protocols used in Chapter 3 to obtain undecidability of secrecy have the following property: encrypted subterms of distinct communications are unifiable. This property is used by the intruder to transfer information from one role to another by using unbounded many nonces or non-well-typed substitutions. To avoid this, the following tagging scheme is proposed.

A *well-formed protocol* is a protocol $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$, where $w = a_1 b_1 \cdots a_l b_l$ is a send-admissible sequence of matching send-receive pairs $a_i b_i$ of actions.

Definition 4.3.1 A well-formed protocol $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w = a_1 b_1 \cdots a_l b_l)$ is called *tagged* if for any $i \leq l$ and $t \in ESub(t(a_i))$ there exists a constant $c_{t,i} \in \mathcal{C}$, and for any send action a_i there exists $n_i \in M(a_i) \cap \mathcal{N}$ such that:

1. $(\forall i, j \leq l)(\forall t \in ESub(t(a_i)))(\forall t' \in ESub(t(a_j)))$
$(t \neq t' \vee i \neq j \Rightarrow c_{t,i} \neq c_{t',j})$;

2. $(\forall i \leq l)(\forall t \in ESub(t(a_i)))(\exists u, K)(t = \{(c_{t,i}, (n_i, u))\}_K)$.

The first property says that distinct encrypted subterms have associated distinct constants. Moreover, if an encrypted subterm occurs in two distinct send actions, then it has distinct constants. Using a nonce in every encrypted subterm is essential in obtaining of the decidability result. This tagging scheme ensures that no two encrypted subterms of distinct communications are unifiable.

The tagging scheme in [7] is more general than the tagging scheme in [55] by that it allows the use of the same tag for two distinct occurrences of the same term. In the tagging scheme in [55], encrypted terms are tagged distinctly, taking into consideration the position where they occur as well. Due to this fact, the tagging scheme from [55] cannot be used to handle *blind copies* as in Π^1 Woo-Lam protocol [70], but this is possible with the tagging scheme in [33, 7]. Another difference between the tagging scheme from [7] and [55] is that first obtains the termination of a verification algorithm of secrecy, while the second obtains the decidability of secrecy.

Definition 4.3.2 Let $\mathcal{P}$ be a tagged protocol and $\xi = e_1 \cdots e_k$ be a well-typed run of $\mathcal{P}$.

(1) e_j is a *good successor* of e_i (and e_i is a *good predecessor* of e_j) if $i < j$ and one of the following conditions holds:

- $e_i \rightarrow e_j$;
- e_i is a send event, e_j is a receive event, and $ESub(t(e_i)) \cap ESub(t(e_j)) \neq \emptyset$.

(2) e_i is a *good event in* ξ if either it is the last event in ξ or it has good successors.

(3) e_i is a *bad event in* ξ if e_i is not a good event in ξ.

(4) ξ is a *good run* if all its events are good.

(5) A *good path in* $\xi = e_1 \cdots e_k$ is any sequence $e_{i_1} \cdots e_{i_r}$ such that $e_{i_j} \in \{e_1, \cdots, e_k\}$ for all $j \leq r$ and $e_{i_{j+1}}$ is a good successor of e_{i_j} for all $j < r$.

Theorem 4.3.1 ([55]) The secrecy problem for tagged protocols is decidable.

Proof (sketch)
Let $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w = a_1 b_1 \cdots a_l b_l)$ be a tagged protocol. The proof follows three main steps, whose proof rely havily on the fact that the protocol is tagged.

1. It is shown that if $\mathcal{P}$ has a leaky run then it has a well-typed leaky run.

Given a substitution σ and a nonce z, define σ_z as being the substitution $\sigma_z(x) = z$, for all $x \in \mathcal{N}$ with $\sigma(x) \notin \mathcal{N}$, and $\sigma_z(x) = \sigma(x)$, otherwise.

Given a run ξ of $\mathcal{P}$, we denote by ξ' the run obtained from ξ by replacing each substitution σ wich appears in ξ by σ_n, where $n \in \mathcal{N}_I$. It can be shown that ξ' is a run of $\mathcal{P}$ which is leaky iff ξ is leaky.

2. It is shown that if $\mathcal{P}$ has a well-typed leaky run then it has a good leaky run.

Let $\xi = s_0[e_1\rangle \cdots s_{m-1}[e_m\rangle s_m$ be a well-typed leaky run of $\mathcal{P}$ such that no proper prefix of ξ is leaky. If ξ is not a good run then we modify it as follows:

a) remove the latest bad event in ξ. Let e_i be this event and $\xi' = e_1 \cdots e_{i-1} e_{i+1} \cdots e_m$ be the sequence such obtained. The messages after e_{i-1} may not be constructible, so ξ' is not necessarely a run.

b) let $T = (analz(s_{iI}) - analz(s_{(i-1)I})) \cap \mathcal{T}_0$ the subset of basic terms that I can deduced only by $t(e_i)$. It is easy to see that $T \subseteq M(e_i)$. Define the substitution σ that maps every $x \in T \cap \mathcal{N}$ into $n \in \mathcal{N}_I$, every $x \in T \cap \mathcal{K}_0$ into $K \in \mathcal{K}_{0,I}$ and it is the identity otherwise. By applying σ to ξ' we obtain a leaky run of $\mathcal{P}$;

c) repeat a) and b) with $\xi = \xi'$ until ξ' is a good run.

The good run obtained in this way is leaky.

3. It is shown that all good runs are of bounded length with respect to the number of distinct events.

Let $\xi = e_1 \cdots e_m$ be a good run of $\mathcal{P}$. Note that there might be multiple occurrences of some event in ξ, but this is not relevant for leakiness analysis because the intruder can not deduce more information if some event have more than one occurrence.

From the fact that ξ is a good run, it follows that for every event e from ξ there is a good path in ξ that starts with e and ends with e_m. Any good path in ξ has the length at most $|w|$. Therefore, the set of events of ξ is partitioned into $E_1, \ldots, E_{|w|}$ where, for all $1 \leq i \leq |w|$, E_i is the set of events e from ξ such that the shortest good path in ξ that starts with e and ends with e_m has length i.

Because an event can have at most two good predecessors, any good run has length at most $2^{|w|} - 1$. Therefore, the messages appearing in all good runs of $\mathcal{P}$ can be constructed from a set T of basic terms exponential in the protocol's body. Good runs of $\mathcal{P}$ are (T, k)-runs, where $k = max\{|t(a_i)| \,/\, 1 \leq i \leq l\}$. Therefore,

$\mathcal{P}$ is a (T, k)-bounded protocol and decidability follows from the results obtained for bounded protocols with freshness check. □

Summing up, decidability of secrecy for tagged protocols leaves the possibility for infinitely many nonces and arbitrary-length messages. Verifying that a tagged protocol is leaky can be done using the algorithm A2 in Section 4.1.2 and, therefore, the secrecy problem for tagged protocols is in NEXPTIME.

Chapter 5

Conclusions

We summarise the work done in this thesis below:

- Chapter 1 is an introduction in the theory of security protocols. We defined security protocols, presented some example of protocols used in practice and gave a glimpse of what a desired property for security protocols means. We pointed out the neccesity to verify the security protocols, given their importance in applications for which security is needed. Security protocols rely on cryptographic primitives, so we briefly describe at them. Next, we illustrate the most known attack strategies on security protocols, emphasize the main difficulties in modelling and analyzing securiy protocols and give a brief survey on the formal methods for security protocol analysis. Finally, we outline the author's contributions to this thesis.

- Chapter 2 begins with a deeper insight of the most known formal models for security protocols. Then, we introduce a security protocol model that is used to develop the results from the next chapters.

- In Chapter 3, we proved that the secrecy problem for security protocols is undecidable, both under infinitely many nonces but bounded-length messages, and finitely many nonces but arbitrary-length messages. This results are obtained by simple and uniform reductions based on the halting problem for two-counter machines and Post's correspondence problem. We also correct the reduction from Post's correspondence problem shown in [25].

- In Chapter 4, we provide decidability and complexity results for three subclasses of security protocols: bounded protocols, normal protocols and tagged protocols. For the subclass of bounded protocols we prove DEXPTIME-completeness for the initial secrecy problem without freshness check,

NEXPTIME-completeness for the (initial) secrecy problem with freshness check and NP-completeness for the (initial) secrecy problem under bounded number of sessions. Then, we provide the relationship between the model used in the thesis and MSR formalism, and correct some flawed statements in [25]. We also provide an undecidability result closely related to a problem left open in [25] (Table 9, page 282). For the subclass of normal protocols we correct a result from [53] and for the last subclass of protocols, we show three types of tagging schemes.

Now, we will give some ideas for future work:

- An important open problem is the complexity of the secrecy problem for bounded protocols without freshness check. We feel that this problem can be solved in DEXPTIME for a representative subclass of bounded protocols obtained by imposing some syntactic restrictions;
- The decision algorithms from Chapter 4 could be converted in practical verification algorithms and incorporated in a tool for security protocol analysis;
- The decidability and complexity results from Chapter 4 can be a starting point to cover other security goals like authentication;
- To make the formalism considered in this thesis more general, composed key can be take into consideration.

Bibliography

[1] M. Abadi, A.D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*. ACM Press, 36-47,1997. Also appeared in *Information and computation*, 148(1):1-70, 1999.

[2] M. Abadi, R. Needham. Prudent engineering practices for cryptographic protocols. *IEEE Transactions of Software Engineering*, 22:6-15,1996.

[3] R.M. Amadio, D. Lugiez, V. Vanackere. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science* 290(1):695-740, 2002.

[4] M. Abadi, M. Tuttle. A Semantics for a Logic of Authentication. In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing* 201-216, August 1991.

[5] G. Bella. Message reception in the inductive approach. Technichal Report 460, CUCL, 1999.

[6] M. Boreale. Symbolic Trace Analysis of Cryptographic Protocols. In *Proceedings of the 28th International Conference on Automata, Languages and Programming (ICALP), volume 2076 of Lecture Notes in Computer Science*, 667-681, 2001.

[7] B. Blanchet, A. Podelski. Verification of Cryptographic Protocols: Tagging Enforces Termination. In Andrew D. Gordon, editor, *Proceedings of FoSSaCS 2003, volume 2620 of Lectures Notes in computer Science*, 136-152, 2003. Also appeared in *Theoretical Computer Science*, 333(1-2):67-90, 2005.

[8] D. Bolignano. Towards a mechanization of cryptographic protocol verification. In *Proceedings of CAV'97 volume 1254 of Lecture Notes in Computer Science*, 131-142, 1997.

[9] M. Burrows, M. Abadi, R. Needham. A Logic of Authentication. *Proceedings of the Royal Society*, Series A, 426(1871), 233-271, 1989. Also appeared in *ACM Transaction on Computer Systems* 8(1):18-36, February 1990.

[10] M. Brackin. A HOL extension of GNY for automatically analyzing cryptographic protocols. In *Proceedings of the 9th Computer Security Foundations Workshop*. IEEE Press, 1996.

[11] *CCITT Draft Recomendation X.509. The Directory Authentication Framework*. Version 7, 1987.

[12] I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell, A. Scedrov. A meta-notation for protocol analysis. In P. Syverson, editor, *12th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 1999.

[13] I. Cervesato, N. Durgin, J. Mitchell, P. Lincoln, A. Scedrov. Relating Strands and Multiset Rewriting for Security Protocol Analysis. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, 35-51, 2000.

[14] E.M. Clarke, S. Jha and W. Marrero. Verifying security protocols with Brutus. *ACM Transactions on Software Engineering and Methodology*, 9(4):443-487, 2000.

[15] W. Charatonik, A. Podelski, J.M. Talbot. Paths vs. Trees in Set-based Program Analysis. *Proceedings of the 27th Annual ACM Symposium on Principles of Programming Languages*, 330-338, 2000.

[16] H. Comon, V. Cortier. Tree Automata with One Memory, Set Constraints and Cryptographic Protocols. unpublished paper, 2001 (`http://www.lsv.ens-cachan.fr/~comon/biblio.html`).

[17] H. Comon, V. Cortier, J.Mitchell. Tree Automata with One Memory, Set Constraints and Ping-Pong Protocols. In *Proceedings of ICALP 2001 volume 2076 of Lecture Notes in Computer Science*, 682-693, 2001.

[18] V. Cortier, J. Millen, H. Rueβ. Proving Secrecy is easy enough. *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, 97-110, 2001.

[19] D. Denning, G. Sacco. Timestamps in Key Distribution Protocols. *Comunications of the ACM* 24(8), August 1981.

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

[21] D. Dolev, S.Even, R. Karp. On the Security of Ping-Pong Protocols. *Information and Control* 55:57-68, 1982.

[22] D. Dolev, A. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory* 29(2):198-208, 1983.

[23] N. Durgin, J. Mitchell. Analysis of security protocols. *In "Calculational System Design, Series F: Computer and System Sciences", volume 173. IOS Press*, 369-395, 1999.

[24] N. Durgin, P. Lincoln, J. Mitchell, A. Scedrov. Undecidability of bounded security protocols. *Proceedings of the Workshop on Formal Methods and Security Protocols - FMSP'99*, Trento, Italy, July 1999.

[25] N. Durgin, P. Lincoln, J. Mitchell, A. Scedrov. Multiset Rewriting and the Complexity of Bounded Security Protocols. *Journal of Computer Security* 12:247-311, 2004.

[26] S. Even, O. Goldreich. On the Security of Multi-Party Ping-Pong Protocols. *Technical Report* 285, Computer Science Department, Technion, Haifa (Israel), June 1983.

[27] M. Fiore, M. Abadi. Computing symbolic models for verifying cryptographic protocols. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, 160-173, 2001.

[28] Federal Information Processing Standard Publication 197. Advanced Encryption Standard, National Institute of Standards and Technology (NIST), 2001.

[29] Formal Systems (Europe) Ltd. Failures Divergence Refinement–User Manual and Tutorial. Version 1.3, 1993.

[30] A.O. Freier, P. Karlton, P.C. Kocher. The SSL protocol version 3.0. `draft-ietf-tls-ssl-version3-00.txt`, November 18, 1996.

[31] L. Gong, R. Needham, R. Yahalom. Reasoning About Beliefs in Cryptographic Protocols. In *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*. IEEE Computer Society, 234-248, 1990.

[32] J. Goubault Larrecq. A method for automatic cryptographic protocol verification. In *Proceedings of the 15th IPDPS Workshops 2000 volume 1800 of Lecture Notes in Computer Science*, 977-984, 2000.

[33] J. Heather, G. Lowe, S. Schneider. How to Prevent Type Flaw Attacks on Security Protocols. In *Proceedings of the 13rd IEEE Computer Security Foundations Workshop (CSFW 13)* 255-268, July 2000.

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

[35] N. Heintze, D. Tygar. A model for secure protocols and their composition. *IEEE Transactions on Software Engineering* 22:16-30, 1996.

[36] J.E. Hopcroft, J.E. Ullman. Introduction to Automata Theory, Languages and Computation. *Addison Wesley*, 1979.

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

[38] G. Lowe. An Attack on the Needham-Schroeder Public-Key Authentication Protocol. *Information Processing Letters*, 56:131-133, 1995.

[39] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In *Proceedings of TACAS'96, volume 1055 of Lecture Notes in Computer Science*, 147-166, 1996. Also in *Software-Concepts and Tools*, 17:93-102, 1996.

[40] G. Lowe. Towards a Completeness Result for Model Checking of Security Protocols. *Journal of Computer Security* 7:89-146, 1999.

[41] G. Lowe, B. Roscoe. Using CSP to detect errors in the TMN protocol. *IEEE Transaction of Software Engineering*, 23(10):659-669, 1997.

[42] C. Meadows. The NRL Protocol Analyzer: An overview. *Journal of Logic Programming*, 26(2):113-131, 1996.

[43] J.K. Millen, V. Shmatikov. Constraint Solving for Bounded-process Cryptographic Protocol Analysis. In *Proceedings of the ACM Conference on Computer and Communications Security*, 166-175, 2001.

[44] S.P. Miller, C. Neumann, J.I. Schiller, J.H. Saltzer. Kerberos authentication and authorization system. Project Athena Technical Plan Section E.2.1, MIT, 1987.

[45] R. Milner. Communication and Concurrency. Prentice-Hall, 1989.

[46] J. Mitchell, M. Mitchell, U. Stern. Automated analysis of cryptographic protocols using Murϕ. In *Proceedings of IEEE Symposium on Security and Privacy*, 141-151, May 1997.

[47] D. Monniaux. Abstracting cryptographic protocol with tree automata. In *Proceedings of the 6th International Static Analysis Symposium (SAS'99), volume 1694 of Lecture Notes in Computer Science*, 1999.

[48] R. Needham, M. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM* 21(12):993-999, 1978.

[49] D.M. Nesset. A Critique of Burrows, Abadi and Needham Logic. *ACM Operating Systems Review* 24(2):35-38, 1990.

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

[51] E.L. Post. A Variant of a Recursively Unsolvable Problem. *Bulletion of the American Mathematical Society*, 52:264-268, 1946.

[52] R. Ramanujam, S.P. Suresh. A Decidable Subclass of Unbounded Security Protocols. *Proceedings of WITS'03*, 11-20, April 2003.

[53] R. Ramanujam, S.P. Suresh. An Equivalence on Terms for Security Protocols. *Proceedings of AVIS'03*, 45-56, April 2003.

[54] R. Ramanujam, S.P. Suresh. Undecidability of the secrecy problem for security protocols. manuscript, July 2003 (`http://www.imsc.res.in/~jam/`).

[55] R. Ramanujam, S.P. Suresh. Tagging makes secrecy decidable for unbounded nonces as well. *Proceedings of the 23rd FST TCS 2003, volume 2914 of Lecture Notes in Computer Science*, 363-374, December 2003.

[56] R. Rivest, A. Shamir, L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120-126, February 1978.

[57] M. Rusinowitch, M. Turuani. Protocol Insecurity with Finite Number of Sessions is NP-complete. In *14th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 174-187, 2001. Also appeared in *Theoretical Computer Science*, 299:451-475, 2003.

[58] S. Schneider. Security properties and CSP. In *Proceedings of the 1996 Computer Society Symposium on Research in Security and Privacy.* IEEE Computer Society Press, 1996.

[59] SET Secure Electronic Transaction Specification, version 1.0, May 1997. (`http://www.visa.com/set/`).

[60] D. Song, S. Berezin, A. Perrig. Athena: A Novel Approach to Efficient Automatic Security Protocol Analysis. *Journal of Computer Security*, 9(1/2):47-74, 2001.

[61] W. Stallings. Cryptography and Network Security: Principles and Practice. Second Edition, Prentice Hall, 1999.

[62] S.P. Suresh. Foundations of Security Protocol Analysis. Ph.D. Thesis, University of Madras, November 2003.

[63] P. Syverson, C. Meadows, I. Cervesato. Dolev-Yao is no better than Machiavelli. *Proceedings of the 1st Workshop on Issues in the Theory of Security (WITS 2000)*, Geneva, Switzerland, 87-92, July 2000.

[64] P. Syverson. Towards a Strand Semantics for Authentication Logic. *Electronic Notes in Theoretical Computer Science*, 20, 1999.

[65] P. Syverson, P.C. van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of the 13th IEEE Symposium on security and privacy*. IEEE Press, 14-28, 1994.

[66] F.J. Thayer F$\acute{a}$brega, J.C. Herzog, J.D. Guttman. Strand Spaces: Proving Security Protocols Correct. *Journal of Computer Security* 7:191-230, 1999.

[67] F.L. Ţiplea, C.V. Bîrjoveanu, C. Enea, I. Boureanu. Secrecy for Bounded Security Protocols With Freshness Check is NEXPTIME-complete, *Journal of Computer Security* (to appear).

[68] F.L. Ţiplea, C.V. Bîrjoveanu, C.Enea. Complexity of the Secrecy Problem for Bounded Security Protocols. In *Proceedings of the NATO Advanced Research Workshop on Information Security in Wireless Networks 2006*, Suceava, România, IOS Press, 2006 (to appear).

[69] F.L. Ţiplea, C.Enea, C.V. Bîrjoveanu. Decidability and Complexity Results for Security Protocols. In *Proceedings of the NATO Advanced Research Workshop on Verification of Infinite-State Systems with Applications to Security VISSAS 2005*, Timişoara, România, IOS Press, 185-211, 2006.

[70] T.Y.C. Woo, S.S. Lam. A Lesson on Authentication Protocol Design. *Operating Systems Review* 28(3):24-37, 1994.