# Automatic verification of cryptographic protocols

Liana Lazar (bozga)

**UNIVERSITE JOSEPH FOURIER - GRENOBLE I**
**SCIENCES ET GEOGRAPHIE**

**THESE**

**pour obtenir le grade de**

**DOCTEUR DE L'UNIVERSITE JOSEPH FOURIER**

**présentée et soutenue publiquement**

**par :** **Liana LAZAR**
**Discipline :** **Informatique**

le 9 décembre 2004

# Automatic Verification of Cryptographic Protocols

*Directeur de thèse :*
**Yassine Lakhnech**

*COMPOSITION DU JURY :*

| | |
|---|---|
| *Rapporteurs* | **Gilles BARTHE** |
| | **Michael RUSINOWITCH** |
| *Directeur de thèse* | **Yassine LAKHNECH** |
| *Examinateurs* | **Florian BOIAN** |
| | **Francis KLAY** |

*to my husband*

# Acknowledgements

*There are lots of people I would like to thank for a huge variety of reasons.*

*First of all I would like to express my gratitude to* Yassine Lakhnech *for having supervised my thesis and trained me as a researcher. I was always impressed by his knowledge of computer science research and the soundness of his critics. Shukran Yassine.*

*I thank* Gilles Barthe *for accepting to referee my thesis and to chair the PhD jury. I thank* Michael Rusinowitch *for accepting to referee my thesis. Also thanks to* Florian Boian *and* Francis Klay *for accepting to take part in my PhD jury and for their useful advices.*

*I thank my husband, for the many discussions we had over different aspects of my work. He helped me a lot, he read and corrected this thesis and he trusted in me. He loves me and understands me any time. Mulţumesc Marius.*

*I would like to thank* Joseph Sifakis*, director of the Verimag laboratory, for giving me the opportunity to start this thesis.*

*It has been a great pleasure for me to work closely with* Cristian, Michael, Romain *and* Laurent*. I thank them for reading my thesis and for their detailed comments which greatly improved the quality of the presentation.*

*I am grateful to all my colleagues from Verimag for their support and friendliness. Also, I would like to say a big "thank you" to all those whom made pleasant my time in France. Particularly merci* Ana*, thanks to you I have a good time at Verimag and especially in France. Mulţumesc* Ileana, Iulian, Alexandra, Iuliana, Cristian, Andreea, Mihaela *and* Marian*. Merci* Jeamy*. Kiitos* Katri*, efharisto* Stavros *and* Mikko*. Dziekuje* Maja*. Grazias* Manuel *and* Lizbeth*.*

*I would like to offer special thanks to my family and all my Romanian friends.*

# Contents

# Introduction

## I.1 Motivation

Cryptographic protocols play a major role in any application where data integrity, confidentiality, authenticity and other security related properties are crucial. Such applications include Smart-Cards, e-business and internet-based voting. In these cases, cryptographic protocols are used respectively to exchange confidential data such as pin numbers and passwords, to authenticate users or to guarantee anonymity of participants.

A cryptographic protocol can be seen as a set of rules defining the messages exchanged between a fixed set of participants, called principals, to implement a functionality such as secret message exchange or authentication.

Cryptography is not sufficient for implementing secure message exchange. Indeed, even under the idealized assumption of perfect cryptography, logical flaws in the protocol design may lead to incorrect behavior with undesired consequences. Maybe the most prominent example showing that cryptographic protocols are notoriously difficult to design and test is the Needham-Schroeder protocol for authentication. It was introduced in 1978 [NS78]. An attack on this protocol was found by G. Lowe using the CSP model-checker FDR in 1995 [Low95]; and this led to a corrected version of the protocol [Low96]. Consequently there has been a growing interest in developing and applying formal methods for validating cryptographic protocols [Mea00, CS02].

Most of this work adopts the so-called Dolev and Yao model. This model assumes perfect cryptography and a nondeterministic intruder that has total control of the communication network and the capacity to forge new messages. Also the intruder has an infinite memory and no bounds are imposed on the size of his computations. This means that the intruder may intercept and remember all messages (sent during the execution of the protocol), forge new messages from received ones and send forged/fake messages. The perfect cryptography means that algebraic properties of encryption functions are not taken into account and therefore, an encrypted message may be decrypted by the intruder only if he knows the appropriate inverse key.

## I.2 Cryptographic Protocols

Cryptographic protocols are sequences of messages that use cryptography to allow two or more entities to authenticate each other and agree on new shared secrets.

### I.2.1 Basic elements of cryptography

Let us introduce some notions of cryptography that are fundamental to understand cryptographic protocols. For a complete survey of cryptography refer e.g. to [MvOV96, Sch94, BG01].

#### Encryption algorithms

An *encryption algorithm* transforms a message, called *plaintext*, into a message that is unintelligible, called *ciphertext*, using an encryption key. Conventionally, the encryption of a plaintext $M$ with a key $K$ is a ciphertext $C$ denoted as follows:

$$C = \{M\}_K \text{ or } C = \mathbf{encr}(M, K)$$

A *decryption algorithm* transforms a ciphertext back into original plaintext using the adequate decryption key. Conventionally, the decryption key corresponding to an encryption key $K$ is denoted $K^{-1}$.

$$\mathbf{decr}(\mathbf{encr}(M, K), K^{-1}) = M$$

There are two classes of encryption algorithms: *symmetric key algorithms* and *asymmetric key algorithms* also called *public key algorithms*.

A *symmetric key algorithm* is an algorithm where the same key is used for both encryption and decryption.

$$K = K^{-1}$$

The symmetric algorithm model is depicted in figure 1.

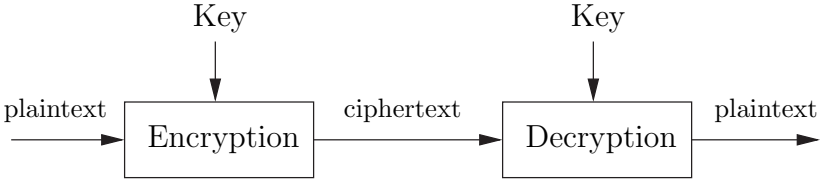


Figure 1: Encryption and decryption with symmetric key

The symmetric algorithms are relatively fast but, in order to exchange a message both the sender and the recipient, must share the same secret key. Hence, they have to find a method to agree upon a shared key before exchanging any messages.

An *asymmetric key algorithm* is an algorithm where the key used for encryption is different from the key used for decryption (see figure 2).

$$K \neq K^{-1}$$

Furthermore, it is computationally infeasible to deduce the decryption key from the encryption key and vice versa. Public key cryptography does not require a shared secret key.

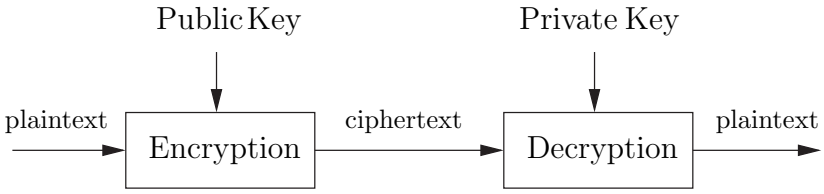


Figure 2: Encryption and decryption with asymmetric key

In this case a participant $P$ has a key pair $(PK(P), SK(P))$. $PK(P)$ is called *public key* and everybody may know this key. $SK(P)$ is called *private key* and only $P$ knows this key. Hence, any participant may encrypt a message $M$ using the public key $PK(P)$ and only $P$ can decrypt the message using the private key $SK(P)$.

The fundamental disadvantage of public key systems is that they are relatively slow. The most popular public-key encryption methods are several orders of magnitude slower than the best known symmetric-key schemes. Therefore they are normally used for encrypting relatively small amounts of data, for example a symmetric key that will be later used for transferring larger amounts of data.

### Digital signature

Most of the algorithms for digital signatures rely on public key cryptography. The private key is used to sign a message and the public key is used to verify the signature.

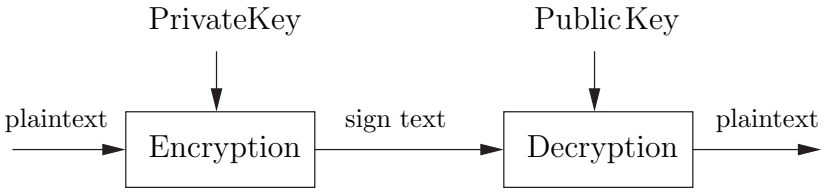


Figure 3: Digital signature model

**Example I.1**

*Consider Alice and Bob. Bob wants to send a message to Alice and wants to be able to prove it came from him (but doesn't care whether anybody else reads it). In this case, Bob sends a plaintext copy of the message to Alice, along with a copy of the message encrypted with his private key. Alice can then check whether the message really came*

*from Bob by decrypting the encrypted message with Bob's public key and comparing it with the plaintext version. If they match, the message was really from Bob, because the private key was needed to create the signature and no one but Bob has it. The ciphertext is Bob's digital signature for the message because anyone can use Bob's public key to verify that Bob created it.*

■

### I.2.2 Cryptographic protocol description

The participants of a cryptographic protocol are called *principals* and are usually denoted by A(Alice) and B(Bob). In order to ensure confidentiality properties the principals use cryptographic algorithms to encrypt messages. Furthermore, they use timestamps and nonces for authentication and freshness purposes:

- A *timestamp* is a digital data created with a purpose to prove temporal relationship between different messages.
- A *nonce* is a "number used once". It is often a random number issued to ensure that messages from old communications cannot be reused in other communications.

Cryptographic protocols have typically been described using the Message Sequence Charts (MSC) notation [IT94]. For example the following is a standard notation description of the most popular cryptographic protocol, the Needham-Schroeder public-key protocol:

$$\begin{aligned} A \rightarrow B: &\ \{A, N_1\}_{pbk(B)} \\ B \rightarrow A: &\ \{N_1, N_2\}_{pbk(A)} \\ A \rightarrow B: &\ \{N_2\}_{pbk(B)} \end{aligned}$$

This describes a protocol run, the sequence of messages exchanged between principals during an *ideal session* execution of the protocol:

- first, the initiator A sends to B its identity and a nonce $N_1$, both encrypted with B's public key $pbk(B)$. The responder B receives the message, decrypts it in order to obtain the nonce $N_1$ and the identity A of the sender.
- second, B sends to A the nonce $N_1$ along with a new nonce, created by him, $N_2$, both encrypted with A's public key $pbk(A)$. A receives this message, it concludes that it is talking to B (since only B should be able to decrypt A's initial message containing nonce $N_1$).
- third, A sends to B the nonce $N_2$ encrypted with B's public key, which allows B to authenticate A in the same manner.

This protocol is designed to ensure principal authentication: at the end of the protocol, both participants $A$ and $B$ should be convinced about the identity of their respective correspondent.

We remark that each step of the protocol consist in a sending action and a receiving action. Therefore, a session may be described also by a sequence of such communication actions.

### I.2.3 Properties of cryptographic protocols

#### Secrecy

A *secrecy* goal states that a designated message should not be public. A secret is public when it is deducible from the set of messages intercepted by the intruder. For instance, in the Needham-Schroeder public-key protocol presented above a secret goal is $N_2$, the nonce sent by the participant B. If the intruder may obtain the nonce $N_2$ then he can encrypt it with the public key of B and send it to B. Hence, B will be convinced that he is talking with A but in fact he is talking with the intruder.

A different and stronger definition of secrecy is used in the spi-calculus model [AG99]. This property is also called *opacity* [Boi03, Maz04]. In this setting, secrecy properties are expressed as properties on execution traces of a protocol. It should not be possible for the intruder to make any difference between a trace execution where a secret $s$ appears and one where instead of $s$ another value $s'$ appears.

In other words, it should not be possible for the intruder to obtain any information about the secret. We remark that this definition is stronger than the first, because it may be violated even if the intruder cannot obtain the secret message. In this thesis we will consider only the first definition of secrecy property.

#### Authentication

Despite the fact that agent authentication is the main, claimed goal of many cryptographic protocols, there exists significant potential for confusion about the interpretation of this term [BAN90, WL92, Low97b, Sch97, RSG+01]. A taxonomy due to Lowe [Low97b] may elucidate the matter identifying four levels of authentication.

1. **Aliveness** of A signifies that A has been running the protocol.
2. **Weak agreement** of B with A signifies that aliveness of A and A has been running the protocol with B.
3. **Non-injective agreement** of B with A on a set of messages M signifies weak agreement of B with A and that the two agents agreed on the set M of messages.
4. **Injective agreement** of B with A on M signifies non-injective agreement of B with A on M and that each run of B corresponds to a unique run of A.

In this thesis we deal only with secrecy and authentication properties, nevertheless we present several other properties.

#### Non-repudiation

Non-repudiation ensures that the author of a message cannot later claim not to be the author. There is a proof that the sender sent the message. This is an indispensable property for the electronic commerce protocols, the seller needing to prove to the bank that the client has really paid. This is often realized by a digital signature.

#### Fairness

Fairness ensures that one of the parties cannot end the protocol part-way through and gain some unfair advantages over the other party. For example, a contract signing protocol where either each agent receives the signed contract, or neither receives any .

#### Anonymity

Anonymity ensures that the identity of an agent is protected with respect to the message that he sent. For example, in a voting protocol the vote must not be linked back to the voter who cast it. In this case, the messages themselves need not be secret, only their association with a particular agent.

## I.3 Cryptographic Protocol Verification

The difficulty of cryptographic protocol verification is due to the unbounded nature of several parameters of the system to verify. There is no bound on:

1. the number of sessions instances that can be created,
2. the number of principals
3. the number of nonce that can be created,
4. the size of the messages that occur during execution of the protocol.

Moreover, the intruder can insert at each step of the protocol, any message that it produces from its knowledge (in the model Dolev-Yao [DY83], one supposes that the intruder has a total control of the network). Hence, the number of messages which the intruder may send at each step is infinite.

In the last few years several decidability and undecidability results about cryptographic protocols verification have been published. The most important results are given below:

- Testing the security for cryptographic protocols is undecidable in the general case [EG83], even when a bound is put on the height of messages [DLMS99] and even without pairing of messages [AC02]. The reachability remains undecidable also in the case of bounded nonce creation but unbounded message size [ALV02, CCM01, Cor03].

- In the case of unbounded number of sessions Dolev, Even and Karp [DEK82] have defined the class of ping-pong protocols and showed its decidability. Later Comon, Cortier and Mitchell [CCM01] extended this class by allowing pairing and binary encryption while the use of nonces still cannot be expressed in their model. The restrictions put on these protocols are, however, too strong and almost none of the known protocols falls in this class.

- In the case of bound number of sessions the results are more optimistic. In [AL00] is proved that reachability is decidable for bounded number of sessions and without restriction on nonce creation or size of messages. Moreover, reachability is shown NP-complete [RT01] for a model which deals in addition with asymmetric encryption and composed keys. Similar results regarding the decidability of the reachability problem for bounded number of sessions have been established also in [Bor01, MS01].

Nowadays, several verification methods have been applied to the the verification of cryptographic protocols. Without pretending to cover all of them, we enumerate several of them.

### I.3.1 Model checking methods

Model-checking methods and tools have been the first to be applied to the analysis of cryptographic protocols. The model-checking usually works on finite-state systems, hence for cryptographic protocols it requires a small and fixed number of sessions to be considered as well as a bounded size of messages exchanged. Many authors have used this approach, either they modeled the cryptographic protocols in order to use general model checkers [KD97, Low96, LR97, MMS97] or they developed special purpose model checkers [KMM94, Mea96, CJM97, Son99].

By using model-checking, many flaws of cryptographic protocols have been effectively discovered. However, if a model checker fails to find an attack it means that there is no attack only in the particular configuration analyzed. Nothing can be said about *all* the possible configurations on that protocol – they could be all correct or an attack may exist on a larger configuration.

There are cases when model checking can be proved complete. For example, Lowe has defined in [Low98] a class of cryptographic protocols for which one honest agent per role and one session, is sufficient to demonstrate attacks. In this class, the protocols must satisfy certain of the "prudent engineering practice" principles enumerated in [AN96] and other assumptions.

Finally, model-checking methods have been combined with theorem proving [KMM94, Mea96] in order to obtain a complete analysis for infinite systems.

### I.3.2 Deductive methods

Methods based on induction and theorem proving have been developed in [Pau97, Bol98, CMR01]. These methods are very general i.e., they can handle unbounded protocols and allows to obtain correctness proofs. But, unlike model-checking, they are not completely automatic and moreover, when the prover fails, it is difficult to obtain counter-examples, and possible attacks on the protocol.

For example, Paulson [Pau97] uses the theorem prover Isabelle [Pau94]. His method requires user interaction, however, a library of theorems and proof techniques is provided to help and guide the user. As an exception, the verification method presented in [CMR01] is fully automatic. Here, the authors proposed a general proof strategy for verifying cryptographic protocols. The strategy is implemented on top of the theorem prover PVS [ORS92]. In practice, it allows to handle many known protocols. But, the termination of this proof strategy is, however, not guaranteed.

### I.3.3 Logic programming based methods

These methods are based on modeling protocols in Horn Logic, e.g. Prolog programs, as in [Wei99, Bla01, AB02] and developing suitable proof strategies. These methods are automatic, they can handle unbounded protocols and prove correctness, however, the termination of the analysis is, in general, not guaranteed.

For example, the method presented in [Wei99] is based on saturation of sets of Horn clauses. The convergence of the saturation process depends on the Horn clauses used to encode the protocol. Decidability and undecidability results have been obtained for certain restricted classes of Horn clauses.

In [Bla01] an abstract representation of cryptographic protocols as Prolog programs is defined. The abstraction allows to prove security properties without limiting the number of runs of protocols. Nevertheless, abstraction may also lead to false attacks.

### I.3.4 Static analysis based methods

Type systems and type-checking have been advocated as a general method for verifying cryptographic protocols e.g, by [Aba97, GJ01, AB01]). This approach relies on specific process algebra to model the protocol and the intruder behavior, as well as to express properties. Typical examples are the spi-calculus [Aba97, GJ01] or the process calculus [AB01].

As an example, the method of [GJ01] works on protocol specifications annotated with *authentication assertions* which encode the protocol requirements. If the annotated specification passes the type checking, then the protocol is guaranteed to satisfy its

requirements. The type checking approach is therefore completely automatic and can handle unbounded protocols. However, as a drawback we must notice that the annotating phase is usually done manually and moreover, it requires a deep understanding both of the protocol as well as the underlying type system. Furthermore, the method is not complete: a type failure indicating either an incorrect behavior or a limitation of the type system considered.

### I.3.5 Abstraction-based methods

Partial algorithms based on abstract interpretation have been developed in [Bol97, Mon99, Bol00, GL00, GK00, BB01]. Particularly, in [Mon99, GL00, GK00] tree automata are used to represent the sets of messages that can be known by the intruder. In [Mon99] the method is restricted to bounded number of session instances. However, the methods of [GL00, GK00] work in the general case, the protocol being approximated by rewriting rules. Both methods compute the set of messages that the intruder may obtain using the protocol rules.

## I.4 Thesis Contribution

In this thesis we propose to apply/adapt and extend traditional verification techniques as logics, weakest precondition calculus or model checking and abstract interpretation to cryptographic protocols.

The contribution of this thesis focuses in two directions. First, the existence of an effective sound and complete inference system for bounded cryptographic protocols and second, a method for automatic verification of unbounded cryptographic protocols based on abstract interpretation. Also, a considerable attention has been given to the modeling of the cryptographic protocols.

### I.4.1 Inference system for bounded cryptographic protocols

A central question in the domain of program semantics and program verification is the existence of a sound and complete inference system for assertions of the form $\pi \models \varphi$ meaning that program $\pi$ satisfies property $\varphi$. Moreover, the question asks for an effective (decidable) complete inference system. This is the question of the relationship between the truth of formulae of the form $\pi \models \varphi$ and their provability.

For While-programs (or counter machines) and 1st-order logic, for instance, it has been proved that it is possible to design an inference system such that provability implies truth (i.e., soundness) but impossible to have a sound system that is complete and effective, i.e., it is impossible to have a decidable inference system such that truth implies provability (see [Cou90] for a complete survey). Roughly speaking, the reason is that one can describe transitive closures using while programs while this is not possible in general in 1st-order logics except when Peano arithmetic is included. In other words, one has to

sacrifice effectiveness (e.g., by including Peano arithmetic in the logic), or completeness and accept that some valid formulae $\pi \models \varphi$ cannot be proved or even expressed. This situation of While-programs led to what is called Cook's relative completeness: *is it possible to have a complete inference system for programs, if we assume all facts of the underlying logic as axioms, i.e., all facts about the considered data are given?*

In the first part of this thesis, we answer to the following question: *what is the situation for cryptographic protocols?*

We remark that undecidability results have been proved for unbounded number of sessions. For example, in [AC02] it is show that it is not difficult to encode a two-counter machine [Min61] as an unbounded cryptographic protocol. Therefore, we know that it is not possible to have an effective complete inference system for unbounded cryptographic protocol. We show that such a system exists for bounded protocols. This provides an alternative proof of the decidability of secrecy for bounded cryptographic protocols [AL00, RT01, MS01] for Dolev-Yao model with atomic keys. Moreover we deal with time-sensitive cryptographic protocols and our logic cover more other properties then secrecy.

Beyond the theoretic relevance of this result, there are several practical consequences. Indeed, if one can provide a complete inference system for cryptographic protocols this can serve as a basis to develop compositional proof theories as well as refinement theories. The latter would be of great interest as the problem of composing cryptographic protocols, i.e., which properties are preserved when cryptographic protocols are composed, as well as the relationship between the abstract specification of a cryptographic protocol and its real implementation remain two insufficiently investigated subjects (cf. [Mea03]).

### I.4.2 Verification algorithm based on abstract interpretation

Verification by abstraction (e.g. [CC77, CC92b, CC92a]) is a major technique for verifying infinite-state and very large systems. This technique consists in finding an abstract system that simulates the concrete one and that is amenable to algorithmic verification. One then checks that the abstract system satisfies an abstract version of the property to verify. Well established preservation results allow then to deduce that the concrete system satisfies the concrete property, if the abstract system satisfies the abstract one.

Hence, the approximation of concrete fix-points by abstract ones can be used to verify concrete properties by abstract properties if the abstract space is finite. The abstraction can also map to an infinite domain, in which case acceleration techniques like widening and narrowing [CC77, CC92a] have to be used to make the fix-point computations converge.

In the second part of this thesis, we present a correct but, in general, incomplete verification algorithm to prove secrecy for unbounded cryptographic protocols without putting any assumption on the size of messages nor on the nonce generation. Proving secrecy means proving that secrets, which are pre-defined messages, are not revealed to unauthorized agents. The main contribution is an original method for proving that a secret

is not revealed by a set of rules that models how the initial set of messages known by the intruder evolves. In contrast to almost all existing methods, we do not try to compute or approximate the sets of messages that can be known by the intruder. Our algorithm is rather based on the notion of "the secret being *guarded* in a message".

**Example I.2**

*For example, suppose that our secret is the nonce* $N_1$ *and that the key* $\mathrm{pvk}(B)$ *– the inverse of* $\mathrm{pbk}(B)$ *– is not known by the intruder. Then, any message that contains* $N_1$ *and that is encrypted with* $\mathrm{pbk}(B)$ *is a guard for* $N_1$*. For instance,* $N_1$ *is guarded in the message* $\{\{N_1, N_2\}_{\mathrm{pbk}(B)}\}_{\mathrm{pbk}(I)}$ *by the key* $\mathrm{pbk}(B)$*.*

■

The problem is, however, that there might be a participant of the protocol which executes the following sequence of actions: he receives $\{I, y\}_{pbk(B)}$ and then he sends $y$, i.e. he sends $y$ unencrypted to the intruder if the intruder produces the message $\{I, y\}_{pbk(B)}$. Hence, the key $pbk(B)$ will guard the secret except when it appears on the $y$ position in a message of the form $\{I, y\}_{pbk(B)}$.

The idea is then to compute the set of encrypted messages that may be useful to the intruder to obtain a secret even if he doesn't know the inverse key necessary to decrypt these messages. The difficulty here is that this set is, in general, infinite. Therefore, we use *terms* and position to represent it. For instance the term $\{I, y\}_{pbk(B)}$ and the position 01 says that the secret will not be guarded in any message of the form $\{I, m\}_{pbk(B)}$, where the secret isn't guarded by any message in $m$.

Thus, our abstract domain consists of pairs $(t, p)$ of terms and position, which we call *term transducers*. Those correspond to "bad exceptions", that is, the particular instances of terms encrypted with good keys that do not guard the secrets. By good keys we understand keys for which the inverse is not known by the intruder.

A weakness of terms is, however, that variables appear only at the leaves, and hence, they do not allow to describe, for instance, the set of terms that share a common sub-term. To mitigate this weakness, we introduce *pattern terms*, that is, terms with an interpreted constructor, *Sup*, where a term $Sup(t)$ is meant for the set of terms that contain $t$ as sub-term. We use patterns in our verification method to introduce a widening operator which help our method to terminate more often, at the price of a safe approximation of the results.

This method has been implemented and tested on various examples. In particular we have been able to verify all protocols that involve secret properties taken from Clark and Jacob survey [CJ97].

## I.5 Organization of the material

In the first part of the thesis we introduce a model for bounded cryptographic protocols and we show that it is possible to have a complete and effective Hoare logic for them and an expressive assertion language. In the second part we extend the model to

unbounded cryptographic protocols and we show how we can use the logic to develop an abstract interpretation based method for verifying secrecy properties. Finally, we present Hermes, a tool which implement the latter method.

In **chapter 1** we introduce the underlying model i.e., abstract syntax and operational semantics for bounded, timed and untimed, cryptographic protocols.

In **chapter 2** we define the logics SPL and TTL for the description of security properties. We establish several useful properties regarding their later use in verification and we study their expressive power. In particular, it turns out that security properties are expressible in the universal fragment of the logic $\text{TTL}_\forall$ (section 2.1).

In **chapter 3** we develop a weakest precondition calculus for bounded cryptographic protocols, using the $\text{TTL}_\forall$ logic. We show that the weakest liberal precondition of a universal formula is again expressible as a universal formula i.e., $\text{TTL}_\forall$ is closed under weakest precondition calculus.

In **chapter 4** we study the decidability of TTL and show that although the satisfiability (existence of a model) of TTL formulae is, in general, undecidable, it is decidable for its existential fragment.

In **chapter 5** we introduce the TTTL logic which is an extension of the TTL logic which deals with properties over time-sensitive cryptographic protocols. The TTTL logic combines constraints on the knowledge of the intruder with time constraints on clocks and time variables. We show that the results on time-independent cryptographic protocols proved for the TTL logic may be also extended for time-dependent cryptographic protocols and TTTL logic.

In **chapter 6** we extend the model introduced in the chapter 1 to unbounded cryptographic protocols. Then, we introduce a sound abstraction scheme allowing to throw away some of the infinite aspects, including the unbounded creation of sessions. The abstraction is sound with respect to secrecy properties.

In **chapter 7** we present a symbolic verification algorithm for secrecy properties on unbounded cryptographic protocols. The algorithm relies on a fixpoint computation of the knowledge (i.e, set of messages) that must not be known by the intruder in order to preserve the secret. We propose a symbolic representation of possible infinite sets of messages. Also, we propose an acceleration scheme of the fixpoint computation based on a widening principle.

Finally, in **chapter 8** we present HERMES a tool which implements the method presented in chapters 6 and 7. The tool is fully operational and has been integrated on a toolbox for the verification of cryptographic protocols in the context of the French National project EVA.

The chapters **2**, **3** and **4** have been the subject of the paper [BEL04a]. The chapter **5** has been the subject of the paper [BEL04b]. The chapters **6** and **7** have been the subject of the paper [BLP03a] and the technical report [BLP02b]. Finally, the chapter **8** has been the subject of the paper [BLP03b].

# Preliminary

### Terms and Term Representation

Let $\mathcal{X}$ be a countable set of variables and let $\mathcal{F}^i$ be a countable set of function symbols of arity $i$, for every $i \in \mathbb{N}$. Let $\mathcal{F} = \bigcup_{i \in \mathbb{N}} \mathcal{F}^i$. The set of *terms over* $\mathcal{X}$ *and* $\mathcal{F}$, denoted by $\mathcal{T}(\mathcal{X}, \mathcal{F})$, is the smallest set containing $\mathcal{X}$ and closed under application of the function symbols in $\mathcal{F}$, i.e., $f(t_1, \cdots, t_n)$ is a term in $\mathcal{T}(\mathcal{X}, \mathcal{F})$, if $t_i \in \mathcal{T}(\mathcal{X}, \mathcal{F})$, for $i = 1, \cdots, n$, and $f \in \mathcal{F}^n$. As usual, function symbols of arity 0 are called constant symbols. *Ground terms* are terms with no variables. We denote by $\mathcal{T}(\mathcal{F})$ the set of ground terms over $\mathcal{F}$.

A terms can be written as a tree:

- each node of the tree is labeled with a function symbol or a variable
- if the function symbol has arity n, then the node for the function symbol has n successor ordered nodes, each one is the root of the sub-tree for the related sub-term.
- if the function symbol is a constant or a variable, the node is a leaf node.

### Example II.3

*Let $f \in \mathcal{F}^2$ and $g \in \mathcal{F}^3$ be functions of arity 2 respective 3, let $a, b \in \mathcal{F}^0$ be constant symbols and let $x, y \in \mathcal{X}$ be variables. The tree in the figure 4 correspond to the term $t = f(y, g(f(a, b), b, f(x, b))$.*

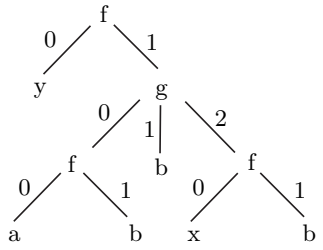


Figure 4: Example of tree representation of terms

■

Now, let us give the formal definition. Let $\omega$ denote the set of integers. A tree $tr$ is a function from a non-empty finite subset of $\omega^*$ to $\mathcal{X} \cup \mathcal{F}$ such that $tr(u) \in \mathcal{F}^n$ iff $u \cdot j \in dom(tr)$, for every $j \in \{0, \cdots, n-1\}$ and $u \cdot j \notin dom(tr)$ for every $j \geq n$, also $tr(u) \in \mathcal{X}$ implies $u \cdot j \notin dom(tr)$ for every $j \in \mathbb{N}$.

We identify terms with trees by associating to each term $t$ a tree $Tr(t)$ as follows:

1. if $x$ is a variable, then $dom(Tr(x)) = \{\varepsilon\}$ and $Tr(x)(\varepsilon) = x$,
2. if $a \in \mathcal{F}^0$ is a constant symbol, then $dom(Tr(a)) = \{\varepsilon\}$ and $Tr(a)(\varepsilon) = a$ and
3. for a term $t = f(t_0, \cdots, t_{n-1}), dom(Tr(t)) = \{\varepsilon\} \cup \bigcup_{i=0}^{n-1} i \cdot dom(Tr(t_i))$, where $\cdot$ is word concatenation extended to sets, $Tr(t)(\varepsilon) = f$ and $Tr(t)(i \cdot u) = Tr(t_i)(u)$.

Henceforth, we tacitly identify the term $t$ with $Tr(t)$.

The elements of $dom(t)$ are called *positions* in $t$. We use $\prec$ to denote the prefix relation on $\omega^*$.

We write $t(p)$ to denote the symbol at position $p$ in $t$ and $t_{|_p}$ to denote the sub-term of $t$ at position $p$, which corresponds to the tree $t_{|_p}(q) = t(p \cdot q)$ with $q \in dom(t_{|_p})$ iff $q \cdot p \in dom(t)$. We write $q^{-1}p$ to denote the position obtained from $p$ after removing the prefix $q$.

**Example II.4**

*We keep the term $t$ from the previous example for reference. $t(\varepsilon) = f$; $t(100) = a$; $t_{|_{12}} = f(x, b)$.*

■

Also, we write $t' \preceq t$ (respectively $t' \prec t$) to denote that $t'$ is a sub-term (respectively proper sub-term) of $t$ . Moreover, $t[t'/p]$ denotes the term obtained from $t$ by substituting $t'$ for $t_{|_p}$. The set of variables in a term $t$ is defined as usual and is denoted by $var\ (t)$.

**Substitution**

A substitution is a function that maps a set of variables to a set of terms. For any substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{X}, \mathcal{F})$ and term $t \in \mathcal{T}(\mathcal{X}, \mathcal{F})$ , we denote by $dom(\sigma)$ the domain of $\sigma$ and by $t\sigma$ the application to $t$ of the homomorphic extension of $\sigma$ to terms. Given a set $X$ of variables, we denote by $\Gamma(X)$ the set consisting of *ground* substitutions with domain $X$. We also write $\Gamma(x)$ instead of $\Gamma(\{x\})$ for a variable $x$. Given two substitutions $\sigma$ and $\rho$ with disjoint domains, $\sigma \oplus \rho$ is the substitution equal to $\sigma$ on $dom(\sigma)$, equal to $\rho$ on $dom(\rho)$, and undefined elsewhere.

**Example II.5**

*Let $\sigma = [x \to g(a); y \to b]$ be a substitution and consider again the term $t = f(y, g(f(a, b), b, f(x, b))$. We have $dom(\sigma) = \{x, y\}$ and*
*$t\sigma = f(b, g(f(a, b), b, f(g(a), b))$.*

■

A substitution is called *idempotent* if it is of the form $[x_1 \rightarrow t_1 \cdots x_n \rightarrow t_n]$ such that none of the variable $x_1 \cdots x_n$ appears in any of the terms $t_1 \cdots t_n$ except when $t_i = x_i$.

**Most General Unifier**

A substitution $\sigma$ is a unifier of $t_1$ and $t_2$ if $t_1\sigma = t_2\sigma$. A substitution $\sigma$ is a most general unifier(mgu) of $t_1$ and $t_2$ if

- $\sigma$ is a unifier of $t_1$ and $t_2$; and
- if substitution $\sigma'$ also unifies $t_1$ and $t_2$, then $t_i\sigma'$ is an instance of $t_i\sigma$ for $i = 1, 2$.

If two terms have a unifier, they have a most general unifier (for more details [BN98]).

For any $t_1, t_2 \in \mathcal{T}(\mathcal{X}, \mathcal{F})$, we denote with $\mu(t_1, t_2)$ the most general unifier (shortly mgu) of $t_1$ and $t_2$, if it exists. More precisely, by $\mu(t_1, t_2)$ we denote the representation of the mgu of $t_1$ and $t_2$ as a conjunction of equalities of the form $x = t$. If it does not exist then $\mu(t_1, t_2)$ should be the constant $false$. We write also $t_1 \sim t_2$, if $t_1, t_2$ can be unified.

**Example II.6**

*Let consider $t_1 = f(a, x)$ and $t_2 = f(y, z)$, where $x, y, z \in \mathcal{X}$. The mgu of $t_1$ and $t_2$ is the substitution $\sigma = [y \rightarrow a; x \rightarrow z]$ and we denote by $\mu(t_1, t_2)$ the conjunction $y = a \wedge x = z$. We note that the substitution $\sigma' = [y \rightarrow a; x \rightarrow a; z \rightarrow a]$ is a unifier of $t_1$ and $t_2$ but not the most general unifier.*

*Now if we consider $t_1 = f(a, x)$ and $t_2 = g(y, z, b)$ then the mgu of $t_1$ and $t_2$ does not exist and $\mu(t_1, t_2) = false$.*

■

# Part I

# Bounded Protocols

# Chapter 1

# Model

In this chapter we introduce our underlying model for the analysis of cryptographic protocols.

Messages are a key concept in our modeling. They represent the information exchanged between participants during the protocol execution. Messages are represented by syntactic terms over a relatively simple $\Sigma$-algebra. Constants include participant names, nonces and keys. Functions are binary composition (or concatenation) of messages and encryption of messages with a key.

The model of the intruder is defined also in terms of messages. The intruder knowledge is defined as the set of messages the intruder knows at some moment in time. Concerning the intruder capabilities we consider the so-called Dolev-Yao model. Within this model, the intruder has complete control over the communication network as well as strong computational capabilities. That is, it can intercept and remove any message in transit in the network. Moreover, it can forge new messages from his knowledge and sent them to participants. Nevertheless, it obeys the perfect cryptography assumption: it can decrypt an encrypted message only if it knows the inverse of the encryption key. Formally, the intruder is described as a deductive system over an infinite set of messages, his knowledge.

Cryptographic protocols are described by parameterized sessions, that is, a set of parameterized agents running in parallel and exchanging messages. Basically, the parameters are agent names. Each agent (also called principal) is described by the sequence of actions - input or output of messages - he execute during the protocol. In addition to parameters, agents can use local variables to store information from messages received. A parameterized session can be instantiated by assigning concrete principal names to parameters and thus creating a so-called session instance. Several session instances in parallel form a bounded protocol instantiation, which is our entry point for analysis in the bounded case, while in the unbounded case we work directly on the parameterized description of the protocol.

The operational semantics describing all the possible evolutions of a bounded protocol is given in section 1.4.

In the last section we present an extension including real-time. This model allows to model cryptographic protocols where real-time is used, for example, to ensure freshness of messages. In addition to the initial model, agents can now use timestamps and clocks to manipulate time related information. Timestamps are variables which store time values. The notion of clocks is the one existing in timed automata [AD94]. They are special variables which increase implicitly as time progresses. Moreover they can be reset to zero or tested against integer values.

The operational semantics for the timed bounded protocols is also provided.

## 1.1 Messages and Terms

### 1.1.1 Messages

The set of messages that we use to describe cryptographic protocols and also the intruder behavior is denoted by $\mathcal{M}$ and contains terms constructed from constant symbols and the function symbols $\mathbf{encr} : \mathcal{M} \times \mathcal{K} \to \mathcal{M}$ and $\mathbf{pair} : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$. Constant symbols are also called atomic messages and are defined as follows:

1. *Principal names* are used to refer to the principals in a protocol. The set of all principals is denoted by $\mathcal{P}$.

2. *Nonces* can be thought as randomly generated numbers. As no one can predict their values, they are used to convince for the freshness of a message. We denote by $\mathcal{N}$ the set of nonces.

3. *Keys* are used to encrypt messages. We have the following atomic keys for each $A_1, \cdots, A_r \in \mathcal{P}^r$ where *pbk*, *pvk* and *smk* stand respectively for *public*, *private* and *symmetric* keys:

$$pbk([i,][n,]A_1, \cdots, A_r) \mid pvk([i,][n,]A_1, \cdots, A_r) \mid smk([i,][n,]A_1, \cdots, A_r).$$

   where $i \in \mathbb{N}$ and $n \in \mathcal{N}$. The first parameter is an integer in order to distinguish between severals keys that belong to the same principals, when needed. For example, if a participant A has two public keys then we model them by $pbk(1, A)$ and $pbk(2, A)$, while if A has just one public key we model it by $pbk(A)$, denoted by $K_A$ for short. The second parameter is a nonce if we want to specify fresh keys, e.g. *short term keys* generated by a trusted server in key distribution protocols. In Neuman Stubblebine Protocol (see example **??**), the key $Kab$ generated by the server must be a fresh key for any execution of the protocol, we model it by $smk(n, A, B)$ where $n$ is a nonce. We use both of them in order to distinguish between several fresh keys that belong to the same principals.

   We denote by $\mathcal{AK}([i,][n,]A_1, \cdots, A_r)$ this set of keys and let $\mathcal{K} = \bigcup_{\vec{A} \in \mathcal{P}^+, i \in \mathbb{N}, n \in \mathcal{N}} \mathcal{AK}([i,][n,]\vec{A})$ denote the set of all keys. The key $pbk([i,][n,]A_1, \cdots, A_r)$ is an inverse of the key $pvk([i,][n,]A_1, \cdots, A_r)$ and

vice versa; a key $smk([i,][n,]A_1, \cdots, A_r)$ is its self-inverse. We define the predicate $\text{inv}(k, k')$ that is true if the key $k'$ is the inverse of the key $k$ and vice versa. If $k$ is a key then we use $k^{-1}$ to denote its inverse.

The set of atomic messages $\mathcal{P} \cup \mathcal{N} \cup \mathcal{K}$ is denoted by $\mathcal{A}$.

The function **pair** is used to model the concatenation of two messages. As usual, we write $(m_1, m_2)$ for $\mathbf{pair}(m_1, m_2)$.

In the case of asymmetric keys, with the function **encr** and the second argument a public key we model *encryption*. If the second argument is a private key we model *signing*. As usual, we write $\{m\}_k$ instead of $\mathbf{encr}(m, k)$.

### 1.1.2 Terms

Let $\mathcal{F} = \mathcal{A} \cup \{\mathbf{encr}, \mathbf{pair}\}$ and $\mathcal{X}$ a set of variables. *Message terms* are the elements of $\mathcal{T}(\mathcal{X}, \mathcal{F})$, that is, terms over the atoms $\mathcal{A}$, the set of variables $\mathcal{X}$ and the binary function symbols $\mathbf{encr} : \mathcal{T}(\mathcal{X}, \mathcal{F}) \times \mathcal{K} \to \mathcal{T}(\mathcal{X}, \mathcal{F})$ and $\mathbf{pair} : \mathcal{T}(\mathcal{X}, \mathcal{F}) \times \mathcal{T}(\mathcal{X}, \mathcal{F}) \to \mathcal{T}(\mathcal{X}, \mathcal{F})$. *Messages* are ground terms in $\mathcal{T}(\mathcal{X}, \mathcal{F})$ i.e, $\mathcal{M} = \mathcal{T}(\mathcal{F})$.

In order to describe the actions that can be performed by a principal in a session of a cryptographic protocol we introduce *role terms*. Let $\mathcal{X}_N$ be a set of variables that range over nonces and $\mathcal{X}_P$ be a set of variables that range over principals. We assume that $\mathcal{X}$, $\mathcal{X}_N$ and $\mathcal{X}_P$ are pairwise disjoint.

*Role terms* are terms constructed from variables in $\mathcal{X} \cup \mathcal{X}_N \cup \mathcal{X}_P$ using the binary function symbols **encr** and **pair** and where constants are not allowed. More, precisely role terms are defined by the following tree grammar:

$$\begin{array}{rcl} \mathit{Key} & ::= & \mathit{pbk}([i,][n,]p_1, \cdots, p_r) \mid \mathit{pvk}([i,][n,]p_1, \cdots, p_r) \mid \mathit{smk}([i,][n,]p_1, \cdots, p_r) \\ \mathit{RoleTerm} & ::= & n \mid p \mid \mathit{Key} \mid x \mid \mathbf{pair}(\mathit{RoleTerm}_1, \mathit{RoleTerm}_2) \mid \mathbf{encr}(\mathit{RoleTerm}, \mathit{Key}) \end{array}$$

where $p, p_1, \cdots, p_r \in \mathcal{X}_P$, $n \in \mathcal{X}_N$, $i \in \mathbb{N}$ and $x \in \mathcal{X}$.

## 1.2 The Intruder Model

We model the intruder using the most commonly used approach, introduced by Dolev and Yao [DY83]. In this model, the intruder has complete control over the network and can derive messages from his knowledge. The intruder can intercept messages, use them to create fake messages and deliver these to the principals.

In our model, the intruder knowledge is represented by the set $E$ of known, intercepted and derived messages. The following rules define how a message $m$ can be derived from a set of messages $E$.

- If $m \in E$ then $E \vdash m$ (axiom).
- If $E \vdash (m_1, m_2)$ then $E \vdash m_1$ and $E \vdash m_2$. This rule is called *projection*.

- If $E \vdash \{m\}_k$, $E \vdash k'$ and $k$ and $k'$ are inverses then $E \vdash m$. This rule is called *decryption*.
- If $E \vdash m_1$ and $E \vdash m_2$ then $E \vdash (m_1, m_2)$. This rule is called *pairing*.
- If $E \vdash m$ and $E \vdash k \in \mathcal{K}$ then $E \vdash \{m\}_k$. This rule is called *encryption*.

The last two rules are called *composition rules*, the others are called *decomposition rules*. A derivation of a message that uses only composition rules is denoted by $E \vdash_c m$ and we say that the message $m$ has been obtained only by composition. Also, we denote by $E \vdash_d m$ a derivation of a message $m$ that uses only decomposition rules.

We say that a message $m$ is derivable from a set of messages $E$, denoted $E \vdash m$, if there is a valid derivation tree using the $\vdash$ rules such that the message $m$ appears at the root of the tree and all messages that appear in the leaves are from $E$. A derivation tree is a *normalized* derivation tree if it does not contain a composition rule which has as conclusion a message $m$ and a decomposition rules which has as premise the same message $m$.

For a set of messages $M$, we write $E \vdash M$, if $E \vdash m$ holds for every $m \in M$. For a term $t$, we use the notation $E \vdash t$ to denote that there exists a substitution $\sigma : \mathcal{X} \to \mathcal{M}$ such that $E \vdash t\sigma$.

We now define *critical* and *non-critical* positions in a message. The idea is that since there is no way to deduce from an encrypted message the key with which it has been encrypted, the key position in messages of the form $\mathbf{encr}(m, k)$ is not critical. This notion appears in many other models, for instance it corresponds to the subterm relation in the strand space model [FHG88, THG98]. Formally, given a term $t$, a position $p$ in $t$ is called *non-critical*, if there is a position $q$ such that $p = q \cdot 1$ and $t(q) = \mathbf{encr}$; otherwise it is called *critical* position. We will also use the notation $s \in_c m$ to denote that $s$ appears in $m$ at a critical position, i.e., there exists $p \in dom(m)$ such that $p$ is critical and $m_{|p} = s$.

An important result of the deduction system of the intruder is the decidability of derivability $E \vdash m$ [Pra65, CJM98, RT01]. The proof of [RT01] is based on the following result:

**Theorem 1.1**

*[CJM98, Bol96] Any derivation tree $T$ for a message $m$ depending on assumption $A$ can be transformed into a normalized derivation tree $T'$ for the message $m$ depending on the same assumption $A$. Moreover, no composition rule appears above a decomposition rule in the normalized derivation tree.*

■

Hence, if there exists a derivation tree for a message $m$ then there exists one where first we apply only decomposition rules and then only composition rules. This theorem is proved for the case where the key cannot be obtained by decomposition.

**Theorem 1.2**

*The number of rules used in the normalized derivation tree for proof that $E \vdash m$ is bounded by the DAG-size of $E \cup \{m\}$.*

■

Where, the DAG-size of a set of terms $E$, is defined to be the number of subterms of $E$.

## 1.3 Cryptographic Protocol Specification

In the literature, cryptographic protocols are often presented as a sequence of message exchanges corresponding to one correct and complete execution of the protocol (see Clark-Jacobs for a survey [CJ97]). Each message exchange has the following form:

$$S \rightarrow R : messg$$

That we read $S$ sends, and $R$ receives the message $messg$. Such a description is not appropriate for verification because it is restricted to a single run of a fixed session with no external interaction. Nevertheless, it can be used to extrapolate a general parameterized description where identities of principals and other constants become parameters, the actions of each principal become independent from the others, and consequently, the sending and the reception of the message $messg$ are separated. The latter aspect allows us to model the interference among several sessions, as well as the presence of an intruder.

### 1.3.1 Protocol Description

In our setting, a cryptographic protocol is described by a *parameterized session description* where the parameters are the involved principals, the fresh nonces and fresh keys. Such a session description is formally given by a tuple $(P, act, fresh)$, where

- $P$ is a vector $(p_1, \cdots, p_r)$, $r \geq 1$, of distinct principal variables in $\mathcal{X}_P$,
- $act$ is a function that associates to each principal variable $p$ in $P$ a finite sequence of actions,
- $fresh$ is a function that associates to each principal variable $p$ in $P$ a disjoint finite set of nonce variables in $\mathcal{X}_N$. By abuse of notation we write $fresh(P)$ to denote $\bigcup_{p \in P} fresh(p)$.

The actions are respectively sending (!) and receiving (?) of messages and are defined by the following grammar:

$$\alpha ::= \ell : \ !t \ \mid \ \ell : \ ?t(\tilde{x})$$

where $\ell$ are action labels and $t$ are role terms. Labels are used as control points in the sequence of actions for a principal. Role terms define the parameterized messages sent and received. Moreover, in the case of an input, $\tilde{x}$ denotes the set of variables that are instantiated (read) by the action. For simplicity, we assume that each variable $y$ occurs exactly once bind in an input action, that is $y \in \tilde{x}$ in action $?t(\tilde{x})$, and moreover, this occurrence precedes any other occurrence of $y$ in any action. It is not difficult to see that restriction can be easily handled. By abuse of notation we use $\tilde{y}$ in an input term to denote that the variable $y$ will be instantiated by that action. Hence, in a protocol description each variable appears under a tilde first time, i.e. when it is instantiated.

**Example 1.1**

*Consider again the Needham-Schroeder public-key protocol. In our setting it is described by the session description where* $P = (p_1, p_2)$, $\text{fresh}(p_1) = \{n_1\}$ *and* $\text{fresh}(p_2) = \{n_2\}$ *and the action lists are:*

$$\begin{array}{ll} \text{act}(p_1): & \text{act}(p_2): \\ 0: \ !\{p_1, n_1\}_{\text{pbk}(p_2)} & 0: \ ?\{\tilde{y}, \tilde{z}\}_{\text{pbk}(p_2)} \\ 1: \ ?\{n_1, \tilde{x}\}_{\text{pbk}(p_1)} & 1: \ !\{z, n_2\}_{\text{pbk}(y)} \\ 2: \ !\{x\}_{\text{pbk}(p_2)} & 2: \ ?\{n_2\}_{\text{pbk}(p_2)} \end{array}$$

*We point out here that, in this protocol description, the initiator* $p_1$ *knows the identity of the responder* $p_2$ *but the responder does not. The responder receives this value in the first message that he reads and stores it in the variable* $y$*. This choice has been made following the principle that a participant only knows the information needed for the protocol execution.*

■

### 1.3.2 Session Instance

Let $\mathcal{S} = (P, act, fresh)$ be a given session description. A session instance is fixed by a pair $(i, \pi)$, where $i$ is its identifier and $\pi$ is a vector of principals $\mathcal{P}^r$ that instantiate the principal variables $p_1, \cdots, p_r$. Therefore, we introduce the set $\text{Inst} = \mathbb{N} \times \mathcal{P}^r$ of session instances. As we impose that the variables in $P$ are distinct, we can use $\pi(p_j)$ to refer the $j^{th}$ principal name in the vector $\pi$, i.e., we can identify $\pi$ with a function, $\pi : P \to \mathcal{P}$. We refer to a session instance by its identifier.

We assume that we have for each fresh variable $n \in fresh(P)$ an injective function which associates for each session instance a fresh nonce value, $n : \mathbb{N} \to \mathcal{N}$ such that $n_1(i_1) \neq n_2(i_2)$, if $n_1 \neq n_2$ or $i_1 \neq i_2$. That is, any fresh parameter is instantiated with different values in different sessions. Moreover, different fresh parameters are instantiated with different values in the same or in different sessions. We write $N^i$ for the value of $n(i)$ where $n$ is a nonce fresh variable and $i$ is the session instance identifier. Intuitively, we use $N^i$ as the nonce corresponding to the value of the fresh variable $n$ in the session instance $(i, \pi)$.

In order to produce an instance of the session description, we have to choose a session

number and a substitution that associates a constant name to each principal variable in $P$. Hence, given $(i, \pi) \in \text{Inst}$, we generate a session instance, denoted by $(\mathcal{S})^i_\pi$, by applying the following transformations to all role terms that appear in the actions of the principals:

- we replace each principal parameter $p$ by $\pi(p)$,
- each nonce parameter $n \in \mathit{fresh}(P)$ by $N^i$.

We denote by $t^i_\pi$ the message term obtained from $t$ by applying the transformations above. Then, the $(i, \pi)$-instance of an action $\ell : \ !/?t$ is $\ell : \ !/?t^i_\pi$. Given $p \in P$, we denote by $\mathit{act}^i(\pi(p))$ the list of $(i, \pi)$-instantiated actions obtained from $\mathit{act}(p)$. A session instance $(\mathcal{S})^i_\pi$ is described by the set of action sequences $\mathit{act}^i(\pi(p))$ with $p \in P$.

**Example 1.2**

*We illustrate the instantiation process on the Needham-Schroeder example. Let* $(i = 0, \pi = (A, B))$ *be a session instance. Moreover,* $n_1(0) = {N_1}^0$ *and* $n_2(0) = {N_2}^0$. *Then, the* $(\mathcal{S})^i_\pi$ *instance of the Needham-Schroeder protocol is described by the action sequences given below:*

$\text{act}^0(A)$ :
$0 : \ !\{A, {N_1}^0\}_{\text{pbk}(B)}$
$1 : \ ?\{{N_1}^0, \tilde{x}\}_{\text{pbk}(A)}$
$2 : \ !\{x\}_{\text{pbk}(B)}$

$\text{act}^0(B)$ :
$0 : \ ?\{\tilde{y}, \tilde{z}\}_{\text{pbk}(B)}$
$1 : \ !\{z, {N_2}^0\}_{\text{pbk}(y)}$
$2 : \ ?\{{N_2}^0\}_{\text{pbk}(B)}$

■

## 1.4 Cryptographic Protocols Semantics

Let $\mathcal{S} = (P, \mathit{act}, \mathit{fresh})$ be a protocol. In the bounded case, the semantics is defined by the parallel execution of a bounded number of instantiated sessions $\mathcal{S}^i_\pi$. We have seen in section 1.3 that an instantiated session is fixed by a pair $(i, \pi)$ and is described by the set of instantiated action sequences.

For the sake of uniformity, the set of instantiated actions is represented as a set of guarded commands. The ordering of actions is encoded as guards that bear as program counter. We represent explicitly control points using finite variables $pc^i_p$, where $p \in P$ is a principal parameter and $i \in \mathbb{N}$ is a session identifier. Moreover, output actions $!t$ are modeled as adding an instance of the term $t$ to some message set $X$. Input actions $?t$ are modeled as the containment test of an instance of $t$ to the message set $X$. The precise syntax is the following:

$$\begin{array}{rcll} \alpha & ::= & [pc^i_p = \ell_1] \rightarrow \mathit{add}(X, t); pc^i_p := \ell_2 & \text{(output action)} \\ & | & [pc^i_p = \ell_1 \wedge \mathit{in}(X, t(\tilde{x}))] \rightarrow pc^i_p := \ell_2 & \text{(input action)} \end{array}$$

In the finite case, a protocol $\Pi$ can therefore be described as the finite set of all possible interleavings of the instantiated actions existing in all parallel sessions:

$$\Pi = \sum_{i=1}^{n} \alpha_1^i \cdots \alpha_{m_i}^i$$

where $\sum$ is the usual non-deterministic choice and $(\alpha_j^i)_{i=1,n;j=1,m_i}$ are input and output actions.

The configurations set $\mathcal{C}onf$, of a protocol run, is given by triplets $(E, \sigma, pc)$ consisting of

- a set of messages $E \subseteq \mathcal{M}$, which corresponds to the intruder knowledge
- a ground substitution $\sigma : \mathcal{X} \to \mathcal{M}$ defined over free variables occurring in the protocol,
- a vector of control points $pc$ which has a component for each principal of each session instance. We denote by $pc_p^i$ the component which corresponds to the control point of principal $p \in P$ in the session $S_\pi^i$.

The operational semantics is defined as a labelled transition system over the set of configurations $\mathcal{C}onf$. The transition relation

$$(E, \sigma, pc) \xrightarrow{\alpha} (E', \sigma', pc')$$

is defined by the following two rules:

- *output actions:* $\alpha = [pc_p^i = \ell_1] \to add(X, t); pc_p^i := \ell_2$

$$\frac{pc_p^i = \ell_1}{(E, \sigma, pc) \xrightarrow{\alpha} (E \cup \{\!|t\sigma|\!\}, \sigma, pc[pc_p^i \leftarrow \ell_2])}$$

  That is, sending the message $t\sigma$ amounts to adding $t\sigma$ to the knowledge of the intruder.

- *input actions:* $\alpha = [pc_p^i = \ell_1 \wedge in(X, t(\tilde{x}))] \to pc_p^i := \ell_2$

$$\frac{pc_p^i = \ell_1 \quad \rho \in \Gamma(\tilde{x}) \quad E \vdash t(\sigma \oplus \rho)}{(E, \sigma, pc) \xrightarrow{\alpha} (E, \sigma \oplus \rho, pc[pc_p^i \leftarrow \ell_2])}$$

  That corresponds to receiving any message that matches with $t\sigma$ and is known by the intruder.

The initial configuration is given by a set of messages $E_0$, initially known by the intruder, an empty substitution $\sigma_0$, and the vector of initial control points for any principal, in any session.

**Example 1.3**

*Let consider again the Needham-Schroeder public-key protocol example. We write $v^i$ to specify a variable, a nonce or a participant $v$ involved in the session instance $S^i_\pi$. Let take $S_1 = S^1_{(A,B)}$ and $S_2 = S^2_{(A,C)}$ two sessions instances. The nonce instances are ${N_1}^1$, ${N_2}^1$, ${N_1}^2$ and ${N_2}^2$. We give below the actions of the two sessions:*

$S^1_{(A,B)}$

| $\mathrm{act}^1(p_1)$ : | $\mathrm{act}^1(p_2)$ : |
|---|---|
| $[pc^1_{p_1} = 0] \rightarrow add(X, \{A, {N_1}^1\}_{\mathrm{pbk}(B)}); pc^1_{p_1} := 1$ | $[pc^1_{p_2} = 0 \wedge in(X, \{\tilde{y}^1, \tilde{z}^1\}_{\mathrm{pbk}(B)})] \rightarrow pc^1_{p_2} := 1$ |
| $[pc^1_{p_1} = 1 \wedge in(X, \{{N_1}^1, \tilde{x}^1\}_{\mathrm{pbk}(A)})] \rightarrow pc^1_{p_1} := 2$ | $[pc^1_{p_2} = 1] \rightarrow add(X, \{z^1, {N_2}^1\}_{\mathrm{pbk}(y^1)}); pc^1_{p_2} := 2$ |
| $[pc^1_{p_1} = 2] \rightarrow add(X, \{x^1\}_{\mathrm{pbk}(B)}); pc^1_{p_1} := 3$ | $[pc^1_{p_2} = 2 \wedge in(X, \{{N_2}^1\}_{\mathrm{pbk}(B)})] \rightarrow pc^1_{p_2} := 3$ |

$S^2_{(A,C)}$

| $\mathrm{act}^2(p_1)$ : | $\mathrm{act}^2(p_2)$ : |
|---|---|
| $[pc^2_{p_1} = 0] \rightarrow add(X, \{A, {N_1}^2\}_{\mathrm{pbk}(C)}); pc^2_{p_1} := 1$ | $[pc^2_{p_2} = 0 \wedge in(X, \{\tilde{y}^2, \tilde{z}^2\}_{\mathrm{pbk}(C)})] \rightarrow pc^2_{p_2} := 1$ |
| $[pc^2_{p_1} = 1 \wedge in(X, \{{N_1}^2, \tilde{x}^2\}_{\mathrm{pbk}(A)})] \rightarrow pc^2_{p_1} := 2$ | $[pc^2_{p_2} = 1] \rightarrow add(X, \{z^2, {N_2}^2\}_{\mathrm{pbk}(y^2)}); pc^2_{p_2} := 2$ |
| $[pc^2_{p_1} = 2] \rightarrow add(X, \{x^2\}_{\mathrm{pbk}(C)}); pc^2_{p_1} := 3$ | $[pc^2_{p_2} = 2 \wedge in(X, \{{N_2}^2\}_{\mathrm{pbk}(C)})] \rightarrow pc^2_{p_2} := 3$ |

*One possible interleaving of actions, where the session $S_2$ starts before the session $S_1$ ends, is:*

$$
\begin{array}{ll}
[pc^1_{p_1} = 0] & \rightarrow add(X, \{A, {N_1}^1\}_{\mathrm{pbk}(B)}); pc^1_{p_1} := 1 \\
[pc^2_{p_1} = 0] & \rightarrow add(X, \{A, {N_1}^2\}_{\mathrm{pbk}(C)}); pc^2_{p_1} := 1 \\
[pc^1_{p_2} = 0 \wedge in(X, \{\tilde{y}^1, \tilde{z}^1\}_{\mathrm{pbk}(B)})] & \rightarrow pc^1_{p_2} := 1 \\
[pc^2_{p_2} = 0 \wedge in(X, \{\tilde{y}^2, \tilde{z}^2\}_{\mathrm{pbk}(C)})] & \rightarrow pc^2_{p_2} := 1 \\
[pc^1_{p_2} = 1] & \rightarrow add(X, \{z^1, {N_2}^1\}_{\mathrm{pbk}(y^1)}); pc^1_{p_2} := 2 \\
[pc^1_{p_1} = 1 \wedge in(X, \{{N_1}^1, \tilde{x}^1\}_{\mathrm{pbk}(A)})] & \rightarrow pc^1_{p_1} := 2 \\
[pc^1_{p_1} = 2] & \rightarrow add(X, \{x^1\}_{\mathrm{pbk}(B)}); pc^1_{p_1} := 3 \\
[pc^1_{p_2} = 2 \wedge in(X, \{{N_2}^1\}_{\mathrm{pbk}(B)})] & \rightarrow pc^1_{p_2} := 3 \\
[pc^2_{p_2} = 1] & \rightarrow add(X, \{z^2, {N_2}^2\}_{\mathrm{pbk}(y^2)}); pc^2_{p_2} := 2 \\
[pc^2_{p_1} = 1 \wedge in(X, \{{N_1}^2, \tilde{x}^2\}_{\mathrm{pbk}(A)})] & \rightarrow pc^2_{p_1} := 2 \\
[pc^2_{p_1} = 2] & \rightarrow add(X, \{x^2\}_{\mathrm{pbk}(C)}); pc^2_{p_1} := 3 \\
[pc^2_{p_2} = 2 \wedge in(X, \{{N_2}^2\}_{\mathrm{pbk}(C)})] & \rightarrow pc^2_{p_2} := 3
\end{array}
$$

■

## 1.5 Time-sensitive Model

Some cryptographic protocols rely upon timestamps, recipients used to verify timeliness of messages. The timestamps allow a principal to recognize and reject replays of messages communicated in the past. The presence of timestamps makes the specification and verification of cryptographic protocols a challenging problem.

In this section, we extend our model to time-dependent cryptographic protocols using primitives existing in timed automata theory [AD94, HNSY92, AFH91, BL95].

Our model of timed cryptographic protocols will include clocks, time variables and timestamps. Clocks are variables that range over the time domain and advance with the same rate as time. Each agent has its own set of clocks that he can reset. That is clocks can be used to measure the time that elapses between two events, for instance, sending a message and receiving the corresponding response. Also, we allow a global clock, called "*now*", that is never reset and that can be read and tested by all participants. Time variables are variables over the time domain. They are used to store timestamps, that are time values, obtained through received messages. Such values can be used together with clocks to define conditions on the acceptance of a message.

### 1.5.1 Terms and the intruder model

In addition to the usual terms considered in Dolev-Yao model, we add:

1. *clocks*, i.e. variables that range over the underlying time model. We denote the set of clocks by $\mathcal{C}$.
2. *timestamps*, that is values in the time domain.
3. *time variables*, that is variables that range over the time domain.

It is important to understand the difference between these three distinct notions:

- a time stamp is just a constant;
- clocks and time variables are variables.

Moreover, in the latter case, the difference is that the value of a clock advances with rate one with time while the value of a time variable does not. A time variable is simply a variable that ranges over the time domain.

We fix the time domain to be the set of non-negative real numbers. The sets of atomic messages are $\mathcal{P}$ for principal names, $\mathcal{N}$ for nonces and $\mathcal{K}$ the set of keys and time stamps, $\mathcal{A} = \mathcal{P} \cup \mathcal{N} \cup \mathcal{K} \cup \mathbb{R}_{\geq 0}$. $\mathcal{X}$ denotes the set of variables that range over terms and $\mathcal{Y}$ denotes the set of time variables. The set of function symbols is $\mathcal{F} = \mathcal{A} \cup \{\mathbf{encr}, \mathbf{pair}\}$. We consider terms build from constant symbols in $\mathcal{A}$, clocks in $\mathcal{C}$, time variables in $\mathcal{Y}$ and term variables in $\mathcal{X}$ using the function symbols in $\mathcal{F}$. For conciseness, we write $\mathcal{T}_c$ instead of $\mathcal{T}(\mathcal{X} \cup \mathcal{Y} \cup \mathcal{C}, \mathcal{F})$.

A *Clock-free term* is a term in which no clock appears; time variables and time stamps may appear in a clock-free term. We denote the set of clock-free terms by $\mathcal{T}(\mathcal{X} \cup \mathcal{Y}, \mathcal{F})$.

*Messages* are ground terms in $\mathcal{T}(\mathcal{X} \cup \mathcal{Y}, \mathcal{F})$.

We assume Dolev-Yao's intruder model except that, since we are dealing with timed protocols, we add a rule that allow the derivation of any time stamp.

Formally, the usual model of Dolev and Yao is augmented with the axiom:

$$\text{(Time stamp) If } r \in \mathbb{R}_{\geq 0} \text{ then } E \vdash r.$$

### 1.5.2 Syntax and semantics

Timed cryptographic protocols are build from timed actions. A time constraint is associated to an action and describes when the action is possible.

**Definition 1.1 (time constraints)**

*Time constraints are defined by:*

$$g ::= \top \mid \sum_{i=1}^{n} a_i c_i + \sum_{j=1}^{m} b_j T_j \bowtie d \mid g_1 \wedge g_2 \mid g_1 \vee g_2$$

*where* $m, n \in \mathbb{N}$, $c_i \in \mathcal{C}$ *are clocks,* $T_j \in \mathcal{Y}$ *are time variables,* $a_i, b_j \in \mathbb{Z}$, $d \in \mathbb{Z}$, *and* $\bowtie \in \{<, \leq\}$. *The set of time constraints is denoted by* $\mathcal{TC}$.

■

A time constraint is interpreted with respect to a valuation $\nu$ defined over the finite set of clocks $\{c_1, \ldots, c_n\}$ that associates values in the time domain to clocks, and the substitution $\sigma$ that assigns ground clock-free terms to variables. The interpretation of a time constraint is given by:

- $[\![\top]\!]_{\nu,\sigma} = 1$, for any $\nu$ and $\sigma$.
- $[\![\sum_{i=1}^{n} a_i c_i + \sum_{j=1}^{m} b_j T_j \bowtie d]\!]_{\nu,\sigma} = 1$ iff $\sum_{i=1}^{n} a_i \nu(c_i) + \sum_{j=1}^{m} b_j \sigma(T_j) \bowtie d$;
- $[\![g_1 \wedge g_2]\!]_{\nu,\sigma} = 1$ iff $[\![g_1]\!]_{\nu,\sigma} = [\![g_2]\!]_{\nu,\sigma} = 1$;
- $[\![g_1 \vee g_2]\!]_{\nu,\sigma} = 1$ iff $[\![g_1]\!]_{\nu,\sigma} = 1$ or $[\![g_2]\!]_{\nu,\sigma} = 1$

Then $(\nu, \sigma)$ is said to be a *model* for a time constraint $g$, if $[\![g]\!]_{\nu,\sigma} = 1$.

Notice that omitting the negation for time constraints is not essential as any negation of a time constraint can be put in a positive form.

Given a time constraint $g$ and a set $\mathcal{R}$ of clocks, we denote by $g[\mathcal{R}]$ the time constraint obtained by substituting 0 for all clocks in $\mathcal{R}$. We also use the notation $g + \delta$ to denote the time constraint obtained from $g$ by substituting each clock $c$ in $g$ by $c + \delta$, where $\delta \in \mathbb{R}_{\geq 0}$.

**Definition 1.2 (timed actions)**

*We define the input and the output actions for timed cryptographic protocols as follows:*

- *A* timed input *action is of the form*
  $[pc_p^i = \ell_1 \wedge in(X, t(\tilde{x})) \wedge g] \rightarrow reset(\mathcal{R}); pc_p^i := \ell_2$
  - $g \in \mathcal{TC}$ *is a time constraint called the guard,*
  - $t(\tilde{x}) \in \mathcal{T}(\mathcal{X} \cup \mathcal{Y}, \mathcal{F})$ *is a term and* $\tilde{x} \subseteq \mathcal{X} \cup \mathcal{Y}$ *is the set of variables instantiated by the input action.*

  - $\mathcal{R} \subseteq \mathcal{C}$ *is a subset of clocks and*
  - $\ell_1, \ell_2$ *are action labels*

- *A* timed output *action is of the form* $[pc_p^i = \ell_1 \wedge g] \rightarrow reset(\mathcal{R}); add(X, t_c); pc_p^i := \ell_2$ *where where* $g$, $\ell_1, \ell_2$ *and* $\mathcal{R}$ *are as above and* $t_c \in \mathcal{T}_c$ *is a clock dependent term.*

■

Let $\mathcal{R} \subseteq \mathcal{C}$ be a subset of clocks, $\delta \in \mathbb{R}_{\geq 0}$ a constant, $\nu : \mathcal{C} \longrightarrow \mathbb{R}_{\geq 0}$ a valuation for clocks, and let $t_c \in \mathcal{T}_c$ be a clock dependent term. We denote by $\nu[\mathcal{R}]$ the valuation obtained from $\nu$ by resetting all clocks in $\mathcal{R}$, i.e. $\nu[\mathcal{R}](c) = 0$ for any $c \in \mathcal{R}$ and $\nu[\mathcal{R}](c) = \nu(c)$ for any $c \notin \mathcal{R}$; $\nu + \delta$ denotes the valuation which advances all clocks by the same delay $\delta$, i.e. $(\nu + \delta)(c) = \nu(c) + \delta$; and $t_c[\nu]$ is the term obtained from $t_c$ by replacing all occurrences of $c$ by the value of $\nu(c)$.

Now we are able to define the operational semantics of timed actions. The configurations set $\mathcal{C}onf$, of a protocol run, is given by tuples $(E, \sigma, \nu, pc)$, where, in addition to the untimed case, we have one more component $\nu$ which represent the valuation of the clocks. Note also that the substitution $\sigma$ contains in addition a valuation of time variables.

**Definition 1.3 (timed operational semantics)**

*The operational semantics is defined as a labelled transitional system over the set of configurations* $\mathcal{C}onf$*. The transition relation*

$$(E, \sigma, \nu, pc) \xrightarrow{\alpha} (E', \sigma', \nu', pc')$$

*is defined as follows:*

- output action: $\alpha = [pc_p^i = \ell_1 \wedge g] \rightarrow reset(\mathcal{R}); add(X, t_c); pc_p^i := \ell_2$

$$\frac{pc_p^i = \ell_1 \;\; [\![g]\!]_{\nu,\sigma} = 1}{(E, \sigma, \nu, pc) \xrightarrow{\alpha} (E \cup \{t_c(\sigma \oplus \nu[\mathcal{R}])\}, \sigma, \nu[\mathcal{R}], pc[pc_p^i \leftarrow \ell_2])}$$

  *That is, sending the message* $t_c$ *(provided that guard* $g$ *is satisfied by the actual configuration) amounts to reset clocks in* $\mathcal{R}$ *and adding* $t_c$ *evaluated with respect to the substitution* $\sigma$ *and the valuation of clocks* $\nu[\mathcal{R}]$*, to the knowledge of the intruder.*

- input action: $\alpha = [pc_p^i = \ell_1 \wedge in(X, t(\tilde{x})) \wedge g] \rightarrow reset(\mathcal{R}); pc_p^i := \ell_2$

$$\frac{pc_p^i = \ell_1 \quad \rho \in \Gamma(\tilde{x}) \quad E \vdash t(\sigma \oplus \rho) \;\; [\![g]\!]_{\nu,\sigma\oplus\rho} = 1}{(E, \sigma, \nu, pc) \xrightarrow{\alpha} (E, \sigma \oplus \rho, \nu[\mathcal{R}], pc[pc_p^i \leftarrow \ell_2])}$$

  *That is,* $t$ *corresponds to receiving any message, known to the intruder, that matches with* $t\sigma$ *by a ground substitution* $\rho$*, such that* $g$ *is satisfied by the pair* $(\nu, \sigma \oplus \rho)$*; in addition, this action resets clocks in* $\mathcal{R}$*.*

- time passing: $(E, \sigma, \nu, pc) \xrightarrow{\delta} (E, \sigma, \nu + \delta, pc)$, *for any* $\delta \in \mathbb{R}_{\geq 0}$.

  *This action represents the passage of* $\delta$ *time units; passage of an arbitrary time is denoted by* $\xrightarrow{\tau} = \bigcup_{\delta \in \mathbb{R}_{\geq 0}} \xrightarrow{\delta}$.

*The initial configuration is given by a set of messages* $E_0$*, initially known by the intruder, an empty substitution* $\sigma_0$*, a valuation* $\nu_0$ *and the vector of initial control points for any principal, in any session.*

■

Now, let us give an example of a timed protocol and its description in our model.

**Example 1.4**

*The Denning-Sacco shared key protocol [DS81]. Using the standard notation for cryptographic protocols, it is described as follows:*

$$\begin{array}{ll} A \to S : & A, B \\ S \to A : & \{B, Kab, T, \{Kab, A, T\}_{Kbs}\}_{Kas} \\ A \to B : & \{Kab, A, T\}_{Kbs} \end{array}$$

*The keys* $Kas$ *and* $Kbs$ *are shared keys between the participant* $A$ *respectively* $B$ *and the server* $S$*. The goal of the Denning-Sacco shared key protocol is to allow two principals* $A$ *and* $B$ *to obtain a secret symmetric key from a trusted server* $S$*. Timestamps are used to ensure the freshness of this key.*

*The next table shows how we describe the protocol. The constant parameters* $\delta_1$*,* $\delta_2$ *represent network delays for* $A$ *respectively* $B$*.*

*For convenience of notation we omit the guard when it is the constant* $\top$ *and the set of clocks to be reset when it is empty.*

$$[pc_A^i = 0] \to add(X, (A, B)); pc_A^i := 1$$
$$[pc_A^i = 1 \wedge in(X, \{B, \tilde{x}, T_1, \tilde{y}\}_{\mathrm{smk}(A,S)}) \wedge (now - T_1 < \delta_1)] \to pc_A^i := 2$$
$$[pc_A^i = 2] \to add(X, y); pc_A^i := 3$$

$$[pc_S^i = 0 \wedge in(X, (\tilde{z}, \tilde{v}))] \to pc_S^i := 1$$
$$[pc_S^i = 1] \to add(X, \{v, K, now, \{K, z, now\}_{\mathrm{smk}(v,S)}\}_{\mathrm{smk}(z,S)}); pc_S^i := 2$$

$$[pc_B^i = 0 \wedge in(X, \{\tilde{u}, \tilde{p}, T_2\}_{\mathrm{smk}(B,S)}) \wedge (now - T_2 < \delta_2)] \to pc_B^i := 1$$

*First, the participant* $A$ *sends his identity* $A$ *and the identity of* $B$ *to the server. Then,* $A$ *receives back the message* $\{B, x, T_1, y\}_{\mathrm{smk}(A,S)}$*. If* $T_1$*, is "timely", i.e. the difference between the current time and the value of* $T_1$ *is less than the constant parameter* $\delta_1$ *then* $A$ *accepts* $x$ *as session key and forwards the message* $y$ *to* $B$*.*

*On the other side, $B$, when he receives the message $\{u, p, T_2\}_{\mathrm{smk}(B,S)}$, he checks if $T_2$ is "timely" and, if so, he accepts $u$ as a session key with $p$. The server $S$, every time when he receives a pair of two participants $(z, v)$ he generates a new session key $K$ and sends it together with his current time "now" in a message of the form $\{v, K, now, \{K, z, now\}_{\mathrm{smk}(v,S)}\}_{\mathrm{smk}(z,S)}$ to $z$, the first participant of the received pair.*

■

# Chapter 2

# Security Properties Logics

This chapter introduces two logics for the description of security properties of cryptographic protocols.

The first logic, called SPL, is an extension of the first-order logic on terms with a *Secret* predicate modeling secrecy. This logic is rich enough to express usual secrecy and authentication properties of cryptographic protocols such as aliveness, weak agreements, etc. Nevertheless, SPL is not powerful enough to support a weakest precondition calculus with respect to input and output actions present in protocols.

Therefore, we have to investigate some extensions of the logic allowing to express preconditions. We found that a rather strong and particularly useful notion in this context are message transducers, that are, functions describing structured transformations between messages. In fact, both intruder capabilities to decompose messages as well as to take advantage on the actions of the participants can be modeled directly as message transducers.

Henceforth, we define our second logic, called TTL, as an extension of the first order logic on terms with a *transducer*-protected modality. This modality (used as $m\langle w\rangle_K s$) expresses that a message ($s$) is protected by keys of $K$ in another message ($m$), despite the use of a transducer ($w$) and implicitly of decomposition rules.

The main properties of message transducers and the related modality are investigated. In particular, two key notions allowing for the syntactical reasoning with TTL are defined. First, the notion of closed set of messages allows to take into account also the composition capability of the intruder. More precisely, we show that a message $m$ cannot be obtained by the intruder iff in each closed set consisting of sub-messages of $m$, there exists one protected w.r.t decomposition rules. The second important notion concerns stability of protection modality i.e, that augmenting the intruder knowledge with messages which are already deductible from his knowledge does not influence the protection. We show that the stability property holds for the so-called well-formed modalities, which are a well identified syntactic category of modalities.

Finally, the embedding of SPL into TTL is formally established. The keypoint is to show how the *Secret* predicate of SPL is definable in TTL.

For the sake of clarity, first, we decide to consider only untimed protocols. Later, in chapter 5 we extend the TTL logic and the results obtained on this logic in order to deal with time properties also.

## 2.1 Security Properties Logic SPL

### 2.1.1 Syntax and Semantics

In this section, we introduce an intuitive logic, which allow us to express security properties about cryptographic protocols. The set of formulae of this logic, called *SPL*, is defined in table 2.1. In the definition, $x$ is a meta-variable that ranges over the set $\mathcal{X}$ of first-order variables; first-order variables range over messages; $t$ is a meta-variable over terms. The local control point $pc_p^i$ range over the set of labels $\mathcal{L}$.

$$\varphi, \psi ::= \quad Secret(t) \mid x = t \mid pc_p^i = \ell \mid \top \mid \bot \mid \varphi \wedge \psi \mid \neg\varphi \mid \forall x\varphi$$

Table 2.1: The set of formulas *SPL*

The proposition $Secret(t)$ expresses secrecy in the following (usual) sense: $Secret(t)$ is true in a configuration $(E, \sigma, pc)$, if $t\sigma$ cannot be derived by the intruder from the set of messages $E$. The formal semantics of SPL is given below.

**Definition 2.1**

*The interpretation of a formula $\varphi$ is given by the set of its models, i.e., the set of configurations $[\![\varphi]\!]$ that satisfy the formula:*

- $[\![Secret(t)]\!] = \{(E, \sigma, pc) \mid E \not\vdash t\sigma\}$
- $[\![x = t]\!] = \{(E, \sigma, pc) \mid \sigma(x) = \sigma(t)\}$.
- $[\![pc_p^i = \ell]\!] = \{(E, \sigma, pc) \mid pc_p^i = \ell\}$
- $[\![\top]\!] = \mathcal{C}onf$
- $[\![\bot]\!] = \emptyset$
- $[\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!]$
- $[\![\neg\varphi]\!] = \mathcal{C}onf \setminus [\![\varphi]\!]$
- $[\![\forall x\varphi]\!] = \bigcap_{\{\rho \mid \rho \in \Gamma(x)\}} [\![\varphi[\rho(x)/x]]\!]$.

■

For convenience of notations, we will consider also the disjunction $\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi)$ and implication $\varphi \Rightarrow \psi \equiv \neg\varphi \vee \psi$ defined in the usual way.

### 2.1.2 Expressiveness of SPL

The purpose of this section is to state how security properties such as secrecy and authentication can be described in the SPL logic. Obviously, we can express semantic

secrecy in our logic using the predicate *Secret*. Further we show, by means of an example, how the introduced logic allows to specify the authentication properties.

For confidentiality properties it seems that there exists a sort of consensus, that a message is confidential (or secret) if it can not be derived from the intruder knowledge. The situation is not the same for authentication properties, where several different definitions exist in the literature. Hereafter, we choose the Lowe's hierarchy definitions [Low97b] to present how they can be formalized using SPL.

Let us consider again the Needham-Schroeder public-key protocol with $S_1 = S^1_{(A,B)}$ and $S_2 = S^2_{(A,C)}$ two session instances. We give below the encoding of *aliveness*, *weak agreement*, *non-injective agreement* and *agreement* properties in our logic.

### Aliveness

*Aliveness* of the initiator is guaranteed to the participant $B$ in session $S_1$:

> *if $B$ completes a run of the protocol in session $S_1$, as responder, with one participant, let say $p$, $y^1 = p$, then the participant $p$ has previously been running the protocol (not necessarily with $B$ neither necessarily as initiator)*

In SPL logic we write this property as follows:

$$\begin{aligned} pc^1_{p_2} = 3 \quad \Rightarrow \quad & ((y^1 = A \wedge (pc^1_{p_1} \neq 0 \vee pc^2_{p_1} \neq 0)) \\ & \vee (y^1 = C \wedge pc^2_{p_2} \neq 0) \\ & \vee y^1 = B) \end{aligned}$$

In general, we want to guarantee the aliveness property for a participant $X$ in a session $S$. That is, when the control point of the participant $X$ has moved after the last action, at least one of the control points of the participant with whom $X$ thinks is involved must be different from its first action label.

In our example, the value of the variable $y^1$ gives the identity of the participant with whom $B$ is involved in the session $S_1$. Therefore, depending on the value of $y^1$ we add conditions over the corresponding control points. For example, in our configuration $A$ plays the role $p_1$ in the session $S_1$ and the role $p_1$ in the session $S_2$. So, if $y^1 = A$, then the control point $pc^1_{p_1}$ must be different from its initial value or the control point $pc^2_{p_1}$ must be different from its initial value.

### Weak Agreement

*Weak agreement* of the initiator is guaranteed to the responder $B$ in session $S_1$:

> *if $B$ completes a run of the protocol in session $S_1$, as responder, with one participant, let say $p$, then $y^1 = p$ and the participant $p$ has previously been running the protocol* **with B** *(not necessarily as initiator)*

This property is expressed in SPL as follows:

$$pc^1_{p_2} = 3 \;\Rightarrow\; ((y^1 = A \wedge pc^1_{p_1} \neq 0) \vee (y^1 = C \wedge pc^2_{p_2} \neq 0 \wedge y^2 = B) \vee y^1 = B)$$

In general, we want to guarantee the weak agreement property for a participant $X$ in a session $S$. For weak agreement in addition to aliveness it is required that, the participant $Y$ with whom $X$ plays the session $S$ must play with $X$ at least one time. Therefore, depending on the value of $y^1$ we add the conditions over the corresponding control point and the identity of the participant $Z$ with whom $Y$ plays in that session.

In our example, $A$ plays with $B$ in the session $S_1$, so we have to add a *true* condition for $pc^1_{p_1} \neq 0$. Respectively in the session $S_2$, $A$ plays with $C$, which is different from $B$, so we have to add a *false* condition for $pc^2_{p_1} \neq 0$. Hence, the condition of aliveness $(y^1 = A \wedge (pc^1_{p_1} \neq 0 \vee pc^2_{p_1} \neq 0))$ becomes $(y^1 = A \wedge pc^1_{p_1} \neq 0)$.

**Non-injective agreement**

*Non-injective agreement on* $N_1$ of the initiator is guaranteed to the responder $B$ in session $S_1$:

> *if $B$ completes a run of the protocol in session $S_1$, as responder, with one participant, let say $p$, then $y^1 = p$ and the participant $p$ has previously been running the protocol,* **as initiator***, with $B$ and they have the* **same value** *for $N_1$*

In SPL this property is written as follows:

$$pc^1_{p_2} = 3 \Rightarrow (y^1 = A \wedge pc^1_{p_1} \neq 0 \wedge z^1 = {N_1}^1)$$

In general, we want to guarantee the non-injective agreement property for a participant $X$ in a session $S$ on a set of data $D$. In addition to weak agreement it is required that the participant with whom $X$ plays in the session $S$, be in the expected role. As for a given configuration we know the role played by each participant we remove from the weak agreement formula the case which corresponds to the participants that are not in the expected role. Moreover, to verify that we have the same value for data in $D$, we use the equality between terms.

In our configuration, $B$ and $C$ do not play as initiators, so we keep only the case in which $y^1 = A$.

**Agreement**

*Agreement on* $N_1$ *and* $N_2$ of the initiator is guaranteed to the responder $B$ in session $S_1$:

> *if $B$ completes a run of the protocol in session $S_1$, as responder, with one participant, let say $p$, then $y^1 = p$ and the participant $p$ has previously been running the protocol, as initiator, with $B$ and they have the same values for $N_1$ and $N_2$. Moreover each such run of $B$* **corresponds to a unique run** *of $p$*

This property is written in SPL as:

$$pc^1_{p_2} = 3 \Rightarrow y^1 = A \wedge (pc^1_{p_1} \neq 0 \wedge z^1 = {N_1}^1 \wedge x^1 = {N_2}^1)$$

It should be clear that given an authentication property and a bounded cryptographic protocol, one can systematically derive a formula expressing the property. Also, it is interesting to notice that the formulae that express these properties do not use the $Secret(t)$ predicate. But, there is a general intuition that we can express authentication properties in terms of secret. We will see later how the *Secret* predicate appears by weakest precondition calculus, that we develop in the chapter 3.

Also, we need to express in our logic the initial conditions. In fact, this conditions are "initial the intruder knows such or such message" or "initial the intruder does not know such or such message". All this conditions can be expressed using the *Secret* predicate and its negation.

### 2.1.3 Why the $Secret(t)$ is not enough?

The SPL logic is an intuitive logic that is sufficient to express authentication and secrecy properties for cryptographic protocols. But, our purpose is to develop a complete inference system for cryptographic protocols and if we try to define the weakest precondition in SPL logic we realize that the *Secret* predicate is not expressive enough to do that.

Let us consider the following protocol:

$$\begin{array}{ll} [pc = \ell_0 \wedge in(X, \{\tilde{x}\}_k)] & \rightarrow pc := \ell_1 \\ [pc = \ell_1] & \rightarrow add(X, x); pc := \ell_2 \end{array}$$

and the property $E \not\vdash s$. What should be the weakest precondition that ensures this property at the end of this protocol? We want to say something like the intruder can not derive $s$ from $E$ even if it can decrypt one time a message encrypted with a key $k$ for which he does not know its inverse. In the SPL logic we can not express that. Therefore, in the next section we introduce a modality that allows to express such conditions, which is proper for induction and also provides a *syntactic* characterization of secrecy.

## 2.2 A syntactic characterization of secrecy

From now on we consider $K \subseteq \mathcal{K}$ be a fixed but arbitrary set of keys, such that $\emptyset \neq K \neq \mathcal{K}$. Intuitively, this is the set of keys for which their inverses are supposed to be

unknown for the intruder. Moreover, we denote by $K^{-1} = \{\!|k \mid k' \in K \wedge \mathrm{inv}(k,k')|\!\}$ the set of keys for which their inverses are in $K$.

### 2.2.1 Message Transducers

A *message transducer* is a program that takes a set of messages and returns a set of messages. Formally, message transducers are defined by the following grammar:

$$\begin{array}{rcl} mt & ::= & \lambda x \cdot \mathsf{if}\ x = \{m\}_k\ \mathsf{then}\ x_{|_r} \\ & | & pr_1 \mid pr_2 \mid \sum\limits_{k \notin K} decr(\cdot, k) \\ & | & mt_1 + mt_2 \mid mt^* \\ & | & mt_1 \cdot mt_2 \end{array}$$

Intuitively, $\lambda x \cdot \mathsf{if}\ x = \{m\}_k\ \mathsf{then}\ x_{|_r}$ is a function, which takes a message and if this is equal with $\{m\}_k$ return the submessage of the position $r$ of it. $pr_1, pr_2, decr$ are decomposition functions that correspond respectively to left, right projection and decryption. $+$ is the usual non-deterministic choice, $^*$ is the unbounded iteration and $\cdot$ is the sequential concatenation.

The operational semantics of a message transducer is defined as a labelled transition system over sets of messages by the following rules:

$$\frac{-}{M \xrightarrow{\lambda x \cdot \mathsf{if}\ x = \{m\}_k\ \mathsf{then}\ x_{|_r}} \{\!|\{m\}_{k|_r} \mid \{m\}_k \in M|\!\}}$$

$$\frac{-}{M \xrightarrow{pr_1} \{\!|m_1 \mid (m_1, m_2) \in M|\!\}} \qquad \frac{-}{M \xrightarrow{pr_2} \{\!|m_2 \mid (m_1, m_2) \in M|\!\}}$$

$$\frac{-}{M \xrightarrow{\sum\limits_{k \notin K} decr(\cdot, k)} \{\!|m \mid \{m\}_k \in M \wedge k \notin K|\!\}}$$

$$\frac{M \xrightarrow{mt_1} M' \quad M' \xrightarrow{mt_2} M''}{M \xrightarrow{mt_1 + mt_2} M' \cup M''} \qquad \frac{-}{M \xrightarrow{mt^*} M} \qquad \frac{M \xrightarrow{mt} M' \quad M' \xrightarrow{mt^*} M''}{M \xrightarrow{mt^*} M' \cup M''}$$

$$\frac{M \xrightarrow{mt_1} M' \quad M' \xrightarrow{mt_2} M''}{M \xrightarrow{mt_1 \cdot mt_2} M''}$$

Let $M$ be a set of messages and $w$ a message transducer. We denoted by $w(M)$, the set of messages $M'$ such that $M \xrightarrow{w} M'$.

**Definition 2.2 (accessible message)**

*The set of messages which are accessible by a message transducer $w$ from a set of messages $M$, denoted by $w_{\preceq}(M)$ is defined as follows:*

$$w_{\preceq}(M) = \bigcup_{w' \preceq w} w'(M)$$

*where $\preceq$ denotes the prefix relation with respect to concatenation "$\cdot$".*

■

In the following corollary we give some immediate consequences of the message transducer semantics and the above definition.

**Corollary 2.1**

*Let $M, M'$ two sets of messages and $w$ a message transducer. It holds*

1. $w(M \cup M') = w(M) \cup w(M')$
2. $w_{\preceq}(M \cup M') = w_{\preceq}(M) \cup w_{\preceq}(M')$
3. $w(M) = w'(mt(M))$, *if* $w = mt \cdot w'$
4. $w_{\preceq}(M) = mt_{\preceq}(M) \cup w'_{\preceq}(mt(M))$, *if* $w = mt \cdot w'$

■

We will consider $I_d = (pr_1 + pr_2 + \sum_{k \notin K} decr(\cdot, k))^*$ is a particular message transducer. This message transducer encodes the decomposition ability of the intruder. More precisely, the set of accessible messages by $I_d$ can be also obtained by the intruder only by decomposition rules, starting from the same set $E$ and knowing all keys except $K^{-1}$. This result is established formally by the following lemma.

**Lemma 2.2**

*Let $E$ be a set of messages, such that $\mathcal{K} \setminus K^{-1} \subseteq E$. Then $m \in I_{d\preceq}(E) \Rightarrow E \vdash_d m$.*

■

**Proof:**

Let us take an arbitrary $m \in I_{d\preceq}(E)$. By definition, it must exists some $w \preceq I_d$ such that $m \in w(E)$. Given that $I_d = (pr_1 + pr_2 + \sum_{k \notin K} decr(\cdot, k))^*$ it follows that any $w \preceq I_d$ are finite concatenations of elementary transducers $pr_1$, $pr_2$ and $\sum_{k \notin K} decr(\cdot, k))^*$. Now, it is easy to show by induction on the length of the $w$ sequence that if $m \in w(E)$ then $E \vdash_d m$. The basic case, $|w| = 0$ is obvious: $m \in w(E)$ means that $m \in E$ and in turn

$E \vdash_d m$. Consider now the induction step. Assume the property holds for any $w' \preceq I_d$ of length $n-1$ and take an arbitrary $w \preceq I_d$ with length $n$. Obviously, $w$ has the form $mt \cdot w'$ where $mt$ is one of the elementary transducers used in $I_d$ and the length of $w'$ is $n-1$. Moreover, from the corollary 2.1 and $m \in w(E)$ it follows that $m \in w'(mt(E))$. From the induction hypothesis we know that $mt(E) \vdash_d m$. Now we are almost done, it remains to show that, for any $m' \in mt(E)$ we have $E \vdash_d m'$. This is now obvious because, $mt$ can be either $pr_1$ or $pr_2$ or $\sum_{k \notin K} decr(\cdot, k))^*$ and, in each case, it represents a decomposition ability of the intruder (resp. left, right projection and decryption with known keys from $k \notin K$.

■

Remark that we have not the equivalence. The intruder starting from a set of messages $E$ such that $\mathcal{K} \setminus K^{-1} \subseteq E$ only by decomposition rules can obtain more messages than those accessible by $I_d$ from the same set. The intruder can also decrypt with keys from $K^{-1}$ if it can derive such keys by decomposition.

### 2.2.2 The protected modality

Let $w \in (\mathcal{M} \times \mathcal{P}os)^*$ be a sequence of pairs of form $(\{m\}_k, p)$. We define the message transducer corresponding to $w$, denoted $f(w)$, inductively on the length of $w$ as follows:

$$\begin{aligned} f(\varepsilon) &= I_d \\ f((\{m\}_k, p) \cdot w) &= I_d \cdot (\lambda x \cdot \mathsf{if}\ x = \{m\}_k \mathsf{then}\ x_{|_p}) \cdot f(w) \end{aligned}$$

From now on we use only message transducers of the form $f(w)$ where $w \in (\mathcal{M} \times \mathcal{P}os)^*$ is a finite sequence of pairs of the form $(\{m\}_k, p)$. Henceforth, the message transducer $f(w)$ is tacitly identified with the sequence $w$.

**Definition 2.3 (protected modality)**

*Let $m, s$ be two messages and $w \in (\mathcal{M} \times \mathcal{P}os)^*$ be a message transducer. The predicate $m\langle w \rangle_K s$, is true and we say that "the message $s$ is* **protected** *in $m$ despite $w$" if the message $s$ is not accessible by $w$ from the message $m$, i.e. $s$ can not be obtained from $m$ by means of decomposition or transducer use.*

*Formally,*

$$m\langle w \rangle_K s \Leftrightarrow s \notin w_{\preceq}(m)$$

*This definition is easily generalized to sets of messages: Let $M$ and $S$ be sets of messages. We say that the secrets $S$ are protected in $M$ despite $w$, denoted by $M\langle w \rangle_K S$, if it holds* $\bigwedge_{m \in M, s \in S} m\langle w \rangle_K s$.

■

Intuitively, the protected modality $E\langle w \rangle_K S$ is a predicate asserting that given the intruder's knowledge $E$, any message $s \in S$ is protected by a key in $K$ in any message

the intruder can obtain from $E$ by the decomposition rules and several oracle rules that are deduced from the protocol. The oracle rules are represented as transducers.

Let us give an example to explain what we mean by "oracle rules".

**Example 2.1**

*Let $K = \{\!|k|\!\}$, that means the inverse of $k$ is not known by the intruder, and let be the following sequence of protocol run:*

$$\begin{array}{ll} [pc = \ell_0 \wedge in(X, \{I, \tilde{x}\}_k)] & \rightarrow pc := \ell_1 \\ [pc = \ell_1] & \rightarrow add(X, x); pc := \ell_2 \end{array}$$

*In this case, if the message $(\{I, N\}_k)$ can be derived by the intruder knowledge then the intruder can obtain the value $N$ even if the inverse key of $k$ is unknown to him. That is what we call "oracle rules that are deduce from the protocol". Such a rule is described by the pair $(\{I, N\}_k, 01)$ and corresponds to the elementary transducer $\lambda y \cdot$ if $y = \{I, N\}_k$then $N$.*

■

A direct consequence of the protected definition over sets of messages is the induction stability.

**Proposition 2.3**

*Let $M, S$ be sets of messages, $m$ be a message and $w$ be a message transducer. We have the following equivalence:*

$$M\langle w\rangle_K S \wedge m\langle w\rangle_K S \equiv M \cup \{\!|m|\!\}\langle w\rangle_K S$$

■

Also, the following equivalences hold:

**Proposition 2.4**

*Let $m$ and $s$ be messages, $w \in (\mathcal{M} \times \mathcal{P}os)^*$ be a message transducer and $(b, p) \in (\mathcal{M} \times \mathcal{P}os)$ be a pair of the form $(\{m\}_k, r)$. The following equations hold:*

$$m\langle (b,p)\cdot w\rangle_K s \Leftrightarrow m\langle \varepsilon\rangle_K s \wedge (m\langle \varepsilon\rangle_K b \vee b_{|_p}\langle w\rangle_K s) \tag{2.1}$$

$$(m_1, m_2)\langle w\rangle_K s \Leftrightarrow s \neq (m_1, m_2) \wedge m_1\langle w\rangle_K s \wedge m_2\langle w\rangle_K s \tag{2.2}$$

$$\{m'\}_k\langle w\rangle_K s \Leftrightarrow s \neq \{m'\}_k \wedge m'\langle w\rangle_K s \text{ if } k \notin K \tag{2.3}$$

■

**Proof:**

$$\begin{array}{lll} m\langle (b,p)\cdot w\rangle_K s & \Leftrightarrow & s \notin ((b,p)\cdot w)_{\preceq}(m) \\ & \Leftrightarrow & s \notin (I_d(m) \cup (b,p)(I_d(m)) \cup w_{\preceq}((b,p)(I_d(m)))) \\ & \Leftrightarrow & s \notin I_d(m) \wedge s \notin (b,p)(I_d(m)) \wedge s \notin w_{\preceq}((b,p)(I_d(m))) \end{array}$$

$\Leftrightarrow \quad m\langle\varepsilon\rangle_K s \wedge s \notin (b,p)(I_d(m)) \wedge s \notin w_{\preceq}((b,p)(I_d(m)))$
$\Leftrightarrow \quad m\langle\varepsilon\rangle_K s \wedge (b \notin I_d(m) \vee s \neq b_{|_p}) \wedge (b \notin I_d(m) \vee s \notin w_{\preceq}(b_{|_p}))$
$\Leftrightarrow \quad m\langle\varepsilon\rangle_K s \wedge (b \notin I_d(m) \vee s \notin w_{\preceq}(b_{|_p}))$
$\Leftrightarrow \quad m\langle\varepsilon\rangle_K s \wedge (m\langle\varepsilon\rangle_K b \vee (b_{|_p}\langle w\rangle_K s))$

Following, we denote by $w'$ the message transducer $w$ from which we remove the first occurrence of $I_d$.

$$\begin{array}{rcl}(m_1,m_2)\langle w\rangle_K s & \Leftrightarrow & s \notin w_{\preceq}((m_1,m_2)) \\ & \Leftrightarrow & s \notin w_{\preceq}(I_d((m_1,m_2))) \\ & \Leftrightarrow & s \notin w'_{\preceq}(\{(m_1,m_2)\} \cup I_d(m_1) \cup I_d(m_2)) \\ & \Leftrightarrow & s \neq (m_1,m_2) \wedge s \notin w'_{\preceq}(I_d(m_1)) \wedge s \notin w'_{\preceq}(I_d(m_2)) \\ & \Leftrightarrow & s \neq (m_1,m_2) \wedge m_1\langle w\rangle_K s \wedge m_2\langle w\rangle_K s\end{array}$$

The equivalence 2.3 is similar to the previous case where $I_d(\{m'\}_k) = \{\{m'\}_k\} \cup I_d(m')$.

■

To illustrate the definition 2.3 of protected modality we give the following example:

**Example 2.2**

*Let $m = (\{I, \{N\}_{k_1}\}_{k_2}, A)$ and $K = \{k_1, k_2\}$. Then, $m\langle\varepsilon\rangle_K N$ is true since the set of messages accessible from $m$ by $\varepsilon$ is $\{\{I, \{N\}_{k_1}\}_{k_2}; A\}$ and $N$ does not belong to it.*

*Let now consider the message transducer $w = (\{I, \{N\}_{k_1}\}_{k_2}, 01).(\{N\}_{k_1}, 0)$. Then, $m\langle w\rangle_K N$ is false since the set of messages accessible by $w$ is:*

$$\begin{array}{rcl}M & = & \{(\{I, \{N\}_{k_1}\}_{k_2}, A)\} \\ & \cup & \{\{I, \{N\}_{k_1}\}_{k_2}; A\} \\ & \cup & \{\{N\}_{k_1}\} \\ & \cup & \{N\}\end{array}$$

*As $N$ belongs to $M$, we have $\neg m\langle w\rangle_K N$.*

■

### 2.2.3 Closure of messages

The protected modality takes into account only the decomposition ability of the intruder. To ensure that the intruder cannot derive a message by composition rules, we used the protected modality for a special set of secrets, which we called closed sets. Let us consider an example:

**Example 2.3**

*Let $E = \{s_1, s_2\}$ be a set of messages. Then we have $E\langle w\rangle_K(s_1, s_2)$. But we have both $E \vdash (s_1, s_2)$ and $\neg(s_1, s_2)\langle w\rangle_K(s_1, s_2)$.*

■

This example shows that we have to give particular care to the treatment of composed secrets as they can be obtained either by composition or decomposition. To do so, we define the closure under decomposition of a term.

Let $M$ be a set of sets of messages and let $m$ be a message. We use the notation: $m \uplus M = \{M_i \cup \{m\} \mid M_i \in M\}$.

**Definition 2.4 (closure for message)**

*Let $m \in \mathcal{M}$ a message. We define recursively the set of message sets $mc(m)$, where each message set of $mc(m)$ is a* **closure** *for $m$ as follows:*

$$mc(m) = m \uplus \begin{cases} mc(m1) \cup mc(m2) & \text{if } m = (m1, m2) \\ mc(m') \cup mc(k) & \text{if } m = \{m'\}_k \\ \{K^{-1}\} & \text{if } m \text{ is atomic} \end{cases}$$

*where $K^{-1} = \{k^{-1} \mid k \in K\}$. A set $M$ of messages is called* **closed**, *if for any $m \in M$ there exists $M' \in mc(m)$ such that $M' \subseteq M$. In particular $\forall A \in mc(m)$, $A$ is closed.*

■

Intuitively, a set $A$ of messages is closed if for any message $m$ of it there is a path in the tree representation of $m$ such that each message on this path is in $A$. Also $A$ must contain all keys in $K^{-1}$.

**Example 2.4**

*Consider the message $m = (\{A, N\}_k, B)$. Then $mc(m)$ consists of the following sets:*

$$K^{-1} \cup \{(\{A, N\}_k, B), \{A, N\}_k, (A, N), A\}$$
$$K^{-1} \cup \{(\{A, N\}_k, B), \{A, N\}_k, (A, N), N\}$$
$$K^{-1} \cup \{(\{A, N\}_k, B), \{A, N\}_k, k\}$$
$$K^{-1} \cup \{(\{A, N\}_k, B), B\}.$$

*The closure computation helps in preventing the intruder from making $m$ by composition: it tells us that it is sufficient to ensure that one of these sets of messages remains completely unknown to the intruder.*

■

In the remaining of this subsection we establish some key properties relating closed sets of messages and the protected modality.

**Proposition 2.5**

*Let $S$ be a closed set of messages. Let $E$ be a set of messages such that $S \cap E = \emptyset$. Then, $E \not\vdash_c S$. In other words, if $S$ is closed then no message in $S$ can be derived uniquely by the composition rules.*

■

**Proof:**

By contradiction, let assume $\exists s \in S$ such that $E \vdash_c s$. Choose $s$ such that to be *minimal*,

i.e. there is a derivation for $E \vdash_c s$ which does not contain any strict sub-derivation $E \vdash_c s'$ of a message $s'$ with $s' \in S$. Three cases are possible:

- $s$ is atomic. Then $E \vdash_c s \Rightarrow s \in E \Rightarrow E \cap S \neq \emptyset$ contradiction.
- $s = (s_1, s_2)$ is a pair. Then $E \vdash_c s \Rightarrow$
  - $s \in E \Rightarrow E \cap S \neq \emptyset$ contradiction or
  - $E \vdash_c s_1 \wedge E \vdash_c s_2$. Since $S$ is closed then at least one of $s_1$ or $s_2$ is in $S$, hence $s$ is not minimal, contradiction.
- $s = \{m\}_k$ is an encrypted message. Similar to the previous case.

■

**Proposition 2.6**

*Let $m, s$ be two messages and $E$ be a set of messages such that $\mathcal{K} \setminus K^{-1} \subseteq E$. If $\neg m\langle\epsilon\rangle_K s$ then $E, m \vdash s$.*

■

**Proof:**

By definition $\neg m\langle\epsilon\rangle_K s \Leftrightarrow s \in I_{d\preceq}(\{\!|m|\!\})$. Using lemma 2.2 we obtain $E, m \vdash_d s$ for all $E$ such that $\mathcal{K} \setminus K^{-1} \subseteq E$. Consequently $E, m \vdash s$.

■

**Corollary 2.7**

*Let $s$ be a message and $E$ be a set of messages such that $\mathcal{K} \setminus K^{-1} \subseteq E$. If $E \nvdash s$ then $E\langle\epsilon\rangle_K s$.*

■

**Proof:**

If we suppose that $\neg E\langle\epsilon\rangle_K s$ we have that there is $m \in E$ such that $\neg m\langle\epsilon\rangle_K s$ and using Proposition 2.6 we obtain that $E, m \vdash s$. Since $m \in E$ we obtain $E \vdash s$, contradiction.

■

We come now to the main result about closed sets of messages. Intuitively, a message $m$ cannot be deduced by the intruder from its knowledge $E$ iff there exists a closed set belonging to $\mathrm{mc}(m)$ and protected in $E$.

**Theorem 2.8**

*Let $m$ be a message and $E$ a set of messages such that $\mathcal{K} \setminus K^{-1} \subseteq E$. Then, $E \nvdash m$ iff there exists a set of messages $A \in mc(m)$ s.t. $E\langle\epsilon\rangle_K A$.*

■

**Proof:**

"⇒": $E \not\vdash m \Rightarrow \exists A \in \mathrm{mc}(m)$ s.t. $E\langle\epsilon\rangle_K A$

By induction on the structure of $m$.

1. Case $m$ atomic. Then, $A = \mathrm{mc}(m) = \{\!\{m\}\!\}$ and using the Corollary 2.7 we obtain that $E\langle\epsilon\rangle_K A$.
2. Case of pair $m = (m_1, m_2)$. From $E \not\vdash m$ we have $E \not\vdash m_1$ or $E \not\vdash m_2$. Using the induction hypothesis we have $\exists A_1 \in \mathrm{mc}(m_1)$ such that $E\langle\epsilon\rangle_K A_1$ or $\exists A_2 \in \mathrm{mc}(m_2)$ such that $E\langle\epsilon\rangle_K A_2$. From $E \not\vdash m$ and Corollary 2.7 we obtain $E\langle\epsilon\rangle_K m$. Hence, for $A = \{\!\{m\}\!\} \cup A_1$ or $A = \{\!\{m\}\!\} \cup A_2$ we have $A \in \mathrm{mc}(m)$ and $E\langle\epsilon\rangle_K A$.
3. Case of encrypted message with a key $K$, $m = \{m_1\}_{k_1}$. Similar to the previous case.

"⇐":$\exists A \in \mathrm{mc}(m)$ s.t. $E\langle\epsilon\rangle_K A \Rightarrow E \not\vdash m$.

Let us first prove that we have $E \not\vdash A$ if $A$ closed and $E\langle\epsilon\rangle_K A$.

By contradiction, assume there exists messages $a \in A$ such that $E \vdash a$. Let consider such an $a$ which is minimal in the sense that, in its normal derivation from $E$ there are no any other message from $A$. We distinguish three cases:

- $a \in E$, contradiction with $E\langle\epsilon\rangle_K A$.
- last derivation step is a composition step:
  - case of pairing, $a = (a_1, a_2)$. Since $a \in A$ and $A$ is closed it means that either $a_1$ or $a_2$ are in $A$. Therefore, $a$ is not minimal and hence contradiction
  - case of encryption, $a = \{a'\}_k$, this case is similar to the previous one.
- last derivation step is a decomposition step:

  Since the derivation is in normal form we conclude that all steps are decomposition steps. We distinguish again two cases: between decomposition steps
  - there are decryption steps using keys $k \in K^{-1}$. Again we have a contradiction with the minimality of $a$ because $K^{-1} \subseteq A$.
  - there are no decryption steps using keys $k \in K^{-1}$. In this case $a \in I_{d_{\preceq}}(E)$ and because $a \in A$ this contradicts the hypothesis $E\langle\epsilon\rangle_K A$.

Now let $A \in \mathrm{mc}(m)$ s.t. $E\langle\epsilon\rangle_K A$. $A \in \mathrm{mc}(m)$ implies that $A$ is closed. So we have $E \not\vdash A$. Since $m \in A$ it holds $E \not\vdash m$.

■

### 2.2.4 Well-formed modalities

Let us introduce the notation $E\langle w_i, S_i\rangle_I$ for $\bigwedge_{i\in I} E\langle w_i\rangle_K S_i$. Our purpose now is to define conditions on $w_i$ and $S_i$ such that for any set $E$ of messages, if $E\langle w_i, S_i\rangle_I$ then $m\langle w_i, S_i\rangle_I$, for any message $m$ derivable from $E$. In other words, we look for a condition that ensures the stability of protection under the derivation rules that define Dolev and Yao's intruder.

We have seen in the example 2.3 that we need to consider only closed sets of secrets. But this is not sufficient, as showed by the following example.

**Example 2.5**

*Let $E = \{\{s\}_{k_1}, k_2\}$ be a set of messages with $k_1, k_2$ symmetric keys and let $b = \{\{s\}_{k_1}\}_{k_2}$. The message transducer $(b, 00)$ does not help getting the secret $s$ since it can not be applied to any message of $E$ ($E$ does not contain the message $b$). Therefore, the predicate $E\langle(b, 00)\rangle_K s$ holds.*

*However, the term $b$ is derivable from $E$ using the encryption rule and then, the message transducer $(b, 00)$ can be used to get the secret $s$.*

■

Hence, we need to deal also with the inner message transducers. To do so, let us introduce some notation. Let $(b, p)$ be a message transducer. Then, we denote by $\mathrm{NT}(b, p)$ the next message transducer in $b$ from the top that dominates $b_{|_p}$, if it exists. Before giving a formal definition let us consider an example.

**Example 2.6**

*Let $b$ be the message $\{(\{N\}_{k'}, A)\}_k$ with $k, k' \in K$. Then, $NT(b, 000) = (\{N\}_{k'}, 0)$. On the other hand $NT(b, 01)$ and $NT(b, 00)$ are not defined.*

■

To formalize the definition of the next message transducer we define the *first protecting position.*

**Definition 2.5 (first protecting position FP)**

*Let $m$ be a message and $p$ a critical position in $m$. Then, the* **first protecting position***, denoted by $FP(m, p)$, is defined recursively on the structure of $m$ as follows:*

- *if $m$ is a constant or a variable then $FP(m, p)$ is undefined.*
- *if $m = (m_1, m_2)$ and $p = 0 \cdot p'$ then $FP(m, p) = 0 \cdot FP(m_1, p')$. Similarly, when $p = 1 \cdot p'$.*
  *If $p = \varepsilon$ then $FP(m, p)$ is undefined.*
- *if $m = \{m'\}_k$ and $k \in K$ then $FP(m, p) = \varepsilon$.*
- *if $m = \{m'\}_k$ and $k \notin K$ then $FP(m, p) = 0 \cdot FP(m', 0^{-1}p)$.*
  *If $p = \varepsilon$ then $FP(m, p)$ is undefined.*

*This definition is illustrated in Fig. 2.1 where we suppose that on the path leading from the root to $q$ there is no other position $q'$ such that $m(q') =$* **encr** *and $m(q' \cdot 1) \in K$.*

■

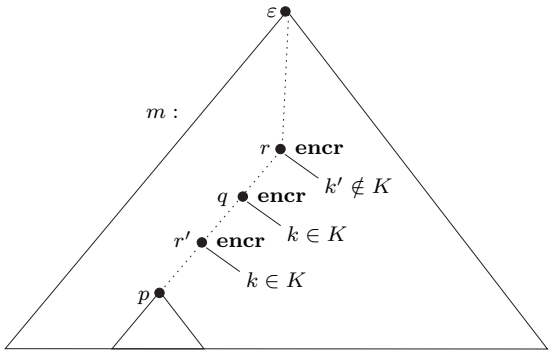


Figure 2.1: The position $q$ is the first protecting position $\mathrm{FP}(m, p)$

Intuitively, $\mathrm{FP}(m, p)$ is the position of the first submessage in $m$ from the root that dominates $p$, and which protects the submessage of the position $p$ by a key from $K$ if it exists.

**Example 2.7**

*Consider the message $m = (\{A, \{N\}_{k_1}\}_{k_2}, N)$, where $k_1, k_2 \in K$. Let $p = 0010$ and $p' = 1$. Thus, $m_{|_p} = m_{|_{p'}} = N$. Then, we have $FP(m, p) = 0$, which corresponds to the key $k_2$; $FP(m, p')$ is, however, undefined.*

■

Now, we are ready to give the formal definition for the *next message transducer* of a message transducer.

**Definition 2.6 (next message transducer NT)**

*Let $(b, p)$ be a message transducer. Then* **the next message transducer** *in $b$ from the top that dominates $p$ (denoted by $NT(b, p)$) is defined as follows:*

$$NT(b,p) = \begin{cases} (b_{|_{0q}}, 0q^{-1}p) & \text{if } FP(b_{|_0}, 0^{-1}p) = q \\ undefined & \text{otherwise} \end{cases}$$

*This definition is illustrated in Fig. 2.2 where $q = FP(b_{|_0}, 0^{-1}p)$.*

■

We have now everything we need to express the conditions that guarantee stability under the intruder's derivations:

**Definition 2.7 (well-formed)**

*$(w_i, S_i)_{i \in I}$ is called* **well-formed**, *if the following conditions are satisfied for every $i \in I$:*

1. *$S_i$ is closed,*

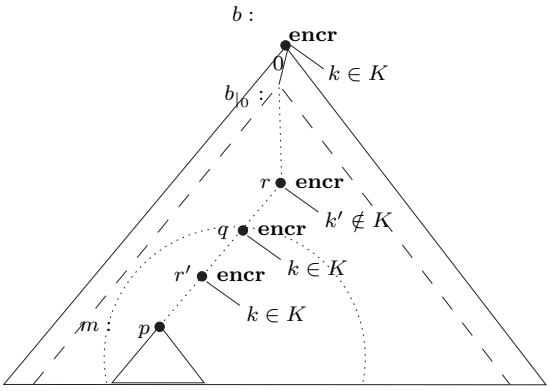


Figure 2.2: The message transducer $(m, q^{-1}p)$ is the next message transducer $\mathrm{NT}(b, p)$

2. *if* $w_i = (b, r).w$, *then the following conditions are satisfied:*

   (a) *there exists* $j \in I$ *such that* $w_j = w$ *and* $S_i \subseteq S_j$,

   (b) *if there exists a term transducer* $(b_1, r_1) = NT(b, r)$, *then there exists* $k \in I$ *such that either* $b \in S_k$ *or* $w_k = (b_1, r_1).w$ *and* $S_i \subseteq S_k$.

■

The main property of $E\langle w_i, S_i\rangle_I$ is that it is stable under the intruder's deduction rules. Indeed, we have:

**Theorem 2.9**

*Let* $E$ *be a set of messages such that* $E\langle w_i, S_i\rangle_I$ *and let* $(w_i, S_i)_{i\in I}$ *be well-formed. Moreover, let* $m$ *be a message with* $E \vdash m$. *Then,* $m\langle w_i, S_i\rangle_I$.

■

**Proof:**

Before tackling the proof, we introduce the following definition: We say that $m$ is a *derivation-minimal counter-example*, if the following conditions are satisfied:

1. $E \vdash m$,
2. $\neg m\langle w_i, S_i\rangle_I$ and
3. there is a derivation for $E \vdash m$ which does not contain any strict sub-derivation $E \vdash m'$ of a message $m'$ with $\neg m'\langle w_i, S_i\rangle_I$.

We derive a contradiction by case analysis on the last derivation step in $E \vdash m$.

1. $m \in E$. This, contradicts the assumption $E\langle w_i, S_i\rangle_I$.
2. Case of encryption with a key from $K$. Thus, $m = \{m_1\}_{k_1}$, $E \vdash m_1$ and $E \vdash k_1$ with $k_1 \in K$. Since $m$ is a derivation-minimal counter-example, we

have $m_1\langle w_i, S_i\rangle_I$ and $k_1\langle w_i, S_i\rangle_I$. Since $\neg m\langle w_i, S_i\rangle_I$, there exists $i \in I$ such that $\neg m\langle w_i\rangle_K S_i$. It follows that $w_i \neq \epsilon$ and hence $w_i = (b, r).w$ and $m = b$ and $\neg b_{|r}\langle w\rangle_K S_i \qquad (*)$.

If $\mathrm{NT}(b, r)$ does not exist then we have $\neg m_1\langle\varepsilon\rangle_K S_i$, and hence, $\neg m_1\langle w_i\rangle_K S_i$, which contradicts the derivation-minimally of $m$.

So, let $(b_1, r_1) = \mathrm{NT}(b, r)$. From definition, we have that $b_{|r} = b_{1|_{r_1}} \qquad (**)$.

Since $(w_i, S_i)_{i\in I}$ is well-formed, following the definition 2.7(2b) there exists $j \in I$ such that either $b \in S_j$ or $w_j = (b_1, r_1).w$ and $S_i \subseteq S_j$.

If we suppose that $b \in S_j$, since $S_j$ is closed and $m = b$, we obtain that either $m_1 \in S_j$ or $k_1 \in S_j$ and hence either $\neg m_1\langle w_j\rangle_K S_j$ or $\neg k_1\langle w_j\rangle_K S_j$, contradiction.

Hence $w_j = (b_1, r_1).w$ and $S_i \subseteq S_j$. From $m_1\langle w_j\rangle_K S_j$, we obtain $b_1|_{r_1}\langle w\rangle_K S_j$ and using $(**)$ we obtain $b_{1|_{r_1}}\langle w\rangle_K S_j \quad (***)$.

From $(*)$, $(**)$ and the definition 2.7(2a) we obtain a contradiction.

3. Case of encryption with a key which is not in $K$. Thus, $m = \{m_1\}_{k_1}$, $E \vdash m_1$ and $E \vdash k_1$ with $k_1 \notin K$. Since $m$ is a derivation-minimal counter-example, we have $m_1\langle w_i, S_i\rangle_I$, and then we obtain that $m\langle w_i, S_i\rangle_I$, contradiction.

4. Case of pairing. Similar to the previous case.

5. Case of projection. This also contradicts the derivation-minimally assumption.

6. Case of decryption. Thus, $m_1 = \{m\}_{k_1}$, $E \vdash m_1$ and $E \vdash k_1^{-1}$. Since $m$ is a derivation-minimal counter-example, we have $m_1\langle w_i, S_i\rangle_I$ and $k_1^{-1}\langle w_i, S_i\rangle_I$. If we suppose that $k_1 \notin K$, then we obtain that either $\neg m_1\langle w_i, S_i\rangle_I$ or $m\langle w_i, S_i\rangle_I$, contradiction.

   If $k_1 \in K$, since for all $i \in I$, $S_i$ are closed, we obtain that $k_1^{-1} \in S_i$, contradiction with $k_1^{-1}\langle w_i, S_i\rangle_I$.

■

## 2.3 Term Transducer Logic TTL

In this section we enrich the SPL logic in such way that we can provide a complete and effective weakest precondition calculus for bounded cryptographic protocols.

### 2.3.1 Syntax and Semantics

By analogy to message transducers $w$ defined for finite sequences from $(\mathcal{M} \times \mathcal{P}os)^*$ we will consider here *term transducers*, defined for finite sequences from $(\mathcal{T} \times \mathcal{P}os)^*$. Intuitively, message transducers have been restricted to messages (i.e, ground terms) whereas, in the case of term transducers we want to consider general terms i.e, containing free variables. Nevertheless, contrarily to message transducers for which an

operational semantics exists and describes their ability to transform sets of messages, term transducers remain only a syntactic notion used in the definition of the TTL logic below. Term transducers $w$ are intended to be *interpreted*, that is, given a ground substitution $\sigma$ instantiateing all the free variables occurring in $w$, we shall denote by $w\sigma$ the message transducer obtained from $w$ by substituting all free variables according to $\sigma$, formally:

$$\big((t_1, p_1) \cdot ... \cdot (t_n, p_n)\big)\sigma \equiv (t_1\sigma, p1) \cdot ... \cdot (t_n\sigma, p_n)$$

Also, in order to express general secrecy properties that involve variables, we introduce a new set of function symbols $\mathcal{B}$. We denote by $\mathcal{B}\mathcal{X}$ the set $\{x.f \mid f \in \mathcal{B}, x \in \mathcal{X}\}$. Given a substitution $\sigma$ that associates a message $m$ to $x$, it will associate a set in $\mathrm{mc}(m)$ to $x.f$. We defined **extended terms** as terms, except that we allow function symbols in $\mathcal{B}$ to occur applied to variables.

The syntax of TTL is defined in Table 2.2, where $X$ is a fixed second-order variable that ranges over sets of messages, $S$ is second-order variable that ranges over closed sets of extended terms, $f$ is a meta-variable that ranges over $\mathcal{B}$, and $x$ is a meta-variable that ranges over the set $\mathcal{X}$ of first-order variables. First-order variables range over messages; $t$ is a meta-variable over terms. Moreover, $w \in (\mathcal{T} \times \mathcal{P}os)^*$ is a term transducer. The formulae are interpreted over a restricted set of configurations $\mathcal{C}onf_0 = \{(E, \sigma, pc) \mid (E, \sigma, pc) \in \mathcal{C}onf \wedge \mathcal{K} \setminus K^{-1} \subseteq E\}$. This means that we only consider the configurations in which all the keys that are not in $K$ are known by the intruder.

$$\varphi, \psi \quad ::= \quad X{\lessdot}w{\gtrdot}_K S \mid x{\lessdot}w{\gtrdot}_K S \mid t{\lessdot}\varepsilon{\gtrdot}_K x.f \mid x = t \mid pc_p^i = \ell \mid \top \mid \varphi \wedge \psi \mid \forall x\varphi \mid \exists f\varphi \mid \neg\varphi$$

Table 2.2: The set of formulae TTL

**Definition 2.8** (TTL **semantics**)

*The semantics of* TTL *is defined inductively on the structure of formula:*

- $[\![X{\lessdot}w{\gtrdot}_K S]\!] = \{(E, \sigma, pc) \mid E\langle w\sigma\rangle_K S\sigma\}$
- $[\![x{\lessdot}w{\gtrdot}_K S]\!] = \{(E, \sigma, pc) \mid x\sigma\langle w\sigma\rangle_K S\sigma\}$
- $[\![t{\lessdot}\varepsilon{\gtrdot}_K x.f]\!] = \{(E, \sigma, pc) \mid t\sigma\langle\varepsilon\rangle_K (x.f)\sigma\}$
- $[\![x = t]\!] = \{(E, \sigma, pc) \mid \sigma(x) = \sigma(t)\}$.
- $[\![pc_p^i = \ell]\!] = \{(E, \sigma, pc) \mid pc_p^i = \ell\}$
- $[\![\neg\varphi]\!] = \mathcal{C}onf_0 \setminus [\![\varphi]\!]$
- $[\![\top]\!] = \mathcal{C}onf_0$
- $[\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!]$

- $[\![\forall x\varphi]\!] = \bigcap_{x_0 \in \mathcal{M}} \{(E,\sigma,pc) \mid (E, \sigma \oplus [x \mapsto x_0], pc) \in [\![\varphi]\!]\}$
- $[\![\exists f\varphi]\!] = \bigcup_{f_0 \in \mathcal{B}} \{(E,\sigma,pc) \mid (E, \sigma \oplus [f \mapsto f_0], pc) \in [\![\varphi]\!]\}$

■

We use the notations $(E,\sigma,pc) \models \varphi$ for $(E,\sigma,pc) \in [\![\varphi]\!]$ and $X\!\lessdot \not w \gtrdot_K S$ for $\neg X\!\lessdot w \gtrdot_K S$. Also, given a term $s$, we write $X\!\lessdot w \gtrdot_K s$ instead of $X\!\lessdot w \gtrdot_K \{s\}$. We identify formulae modulo the usual properties of boolean connectives such as associativity and commutativity of $\wedge$, $\vee$, distributivity etc... and use $\Rightarrow$ as the classical logical implication.

In addition, for convenience of notation, we extend the set of formulae TTL with the following two formulae:

$$(X,x)\!\lessdot w \gtrdot_K S \text{ and } t\!\lessdot w \gtrdot_K S$$

The semantics of the newly introduced formulae is:

- $[\![(X,x)\!\lessdot w \gtrdot_K S]\!] = [\![X\!\lessdot w \gtrdot_K S]\!] \cap [\![x\!\lessdot w \gtrdot_K S]\!]$
- $[\![t\!\lessdot w \gtrdot_K S]\!] = \{(E,\sigma,pc) \mid t\sigma\langle w\sigma\rangle_K S\sigma\}$

We notice that any formula of the first form $(X,x)\!\lessdot w \gtrdot_K S$ may also be expressed directly in TTL i.e, $(X,x)\!\lessdot w \gtrdot_K S \equiv X\!\lessdot w \gtrdot_K S \wedge x\!\lessdot w \gtrdot_K S$. Further, we prove that also formula of the second form $t\!\lessdot w \gtrdot_K S$ is definable in TTL.

**Proposition 2.10**

*Let $t$ be a term, let $S$ be a set of extended terms, let $w$ be a term transducer and let $\mathcal{J}$ be defined as follows:*

$$\mathcal{J}(t,w,S) = \begin{cases} x\!\lessdot w \gtrdot_K S & \text{if } t = x \in \mathcal{X} \\ \mathcal{G}(t,S) & \text{if } t = a \in \mathcal{A} \\ \mathcal{J}(t_1,w,S) \wedge \mathcal{J}(t_2,w,S) \wedge \mathcal{G}(t,S) & \text{if } t = (t_1,t_2) \\ \mathcal{J}(t_1,w,S) \wedge \mathcal{G}(t,S) & \text{if } t = \{t_1\}_k \wedge k \notin K \\ \mathcal{G}(t,S) & \text{if } t = \{t_1\}_k \wedge k \in K \wedge \\ & w = \epsilon \\ \mathcal{G}(t,S) \wedge (\mu(b,t) \Rightarrow \mathcal{J}(b|_r, w_1, S)) & \text{if } t = \{t_1\}_k \wedge \\ & k \in K \wedge w = (b,r).w_1 \end{cases}$$

*where* $\mathcal{G}(t,S) = \bigwedge_{s \in S \setminus \mathcal{B}\mathcal{X})} \neg\mu(t,s) \wedge \bigwedge_{s \in S \cap \mathcal{B}\mathcal{X}} t\!\lessdot \varepsilon \gtrdot_K s$

*Then, $t\!\lessdot w \gtrdot_K S \equiv \mathcal{J}(t,w,S)$, i.e., both formulae are equivalent.*

■

**Proof:**

First, notice that if $\mu(t_1,t_2) \neq \bot$, then $(E,\sigma,pc) \in \mu(t_1,t_2)$ iff $t_1\sigma = t_2\sigma$, and if $\mu(t_1,t_2) = \bot$, then for any $(E,\sigma,pc)$, it holds $t_1\sigma \neq t_2\sigma$.

We give the proof for the first case. We prove by induction on $depth(t) + |w|$ that $t{\lhd}w{\rhd}_K S \equiv \mathcal{J}(t, w, S)$.

1. If $t = x \in X$, then $\mathcal{J}(t, w, S) = x{\lhd}w{\rhd}_K S = t{\lhd}w{\rhd}_K S$.
2. If $t = a \in \mathcal{A}$, then $\mathcal{J}(a, w, S) = \mathcal{G}(a, S) = \bigwedge_{s \in S \setminus \mathcal{BX})} \neg\mu(a, s) \wedge \bigwedge_{s \in S \cap \mathcal{BX}} a{\lhd}\varepsilon{\rhd}_K s$.

   Then we have
   $(E, \sigma, pc) \in [\![ \bigwedge_{s \in S \setminus \mathcal{BX})} \neg\mu(a, s) \wedge \bigwedge_{s \in S \cap \mathcal{BX}} a{\lhd}\varepsilon{\rhd}_K s ]\!]$ iff
   $\bigwedge_{s \in S \setminus \mathcal{BX}} s\sigma \neq a \wedge \bigwedge_{s \in S \cap \mathcal{BX}} a\langle\varepsilon\rangle_K s\sigma$ iff
   $\bigwedge_{s \in S \setminus \mathcal{BX}} a\langle w\sigma\rangle_K s\sigma \wedge \bigwedge_{s \in S \cap \mathcal{BX}} a\langle\varepsilon\rangle_K s\sigma$ iff
   $(E, \sigma, pc) \in [\![ a{\lhd}w{\rhd}_K S ]\!]$.
3. If $t = (t_1, t_2)$, then $\mathcal{J}(t, w, S) = \mathcal{J}(t_1, w, S) \wedge \mathcal{J}(t_2, w, S) \wedge \mathcal{G}(t, S)$.
   By induction hypothesis, we have $\mathcal{J}(t_1, w, S) \equiv t_1{\lhd}w{\rhd}_K S$ and
   $\mathcal{J}(t_2, w, S) \equiv t_2{\lhd}w{\rhd}_K S$. We obtain
   $(E, \sigma, pc) \in [\![ \mathcal{J}(t, w, S) ]\!]$ iff
   $(E, \sigma, pc) \in [\![ t_1{\lhd}w{\rhd}_K S \wedge t_2{\lhd}w{\rhd}_K S \wedge \mathcal{G}(t, S) ]\!]$ iff
   $t_1\sigma\langle w\sigma\rangle_K S\sigma \wedge t_2\sigma\langle w\sigma\rangle_K S\sigma \wedge \bigwedge_{s \in S \setminus \mathcal{BX}} t\sigma\langle w\sigma\rangle_K s\sigma \wedge \bigwedge_{s \in S \cap \mathcal{BX}} t\sigma\langle\varepsilon\rangle_K s\sigma$ iff (prop. 2.4)
   $t\sigma\langle w\sigma\rangle_K S\sigma$ iff
   $(E, \sigma, pc) \in [\![ t{\lhd}w{\rhd}_K S ]\!]$.
4. The case $t = \{t_1\}_k \wedge k \notin K$ is similar to the previous one.
5. If $t = \{t_1\}_k \wedge k \in K \wedge w = \epsilon$, then we have $(E, \sigma, pc) \in [\![ t{\lhd}w{\rhd}_K S ]\!]$ iff
   $t\sigma\langle\epsilon\rangle_K S\sigma$ iff
   $\bigwedge_{s \in S \setminus \mathcal{BX}} t\sigma \neq s\sigma \wedge \bigwedge_{s \in S \cap \mathcal{BX}} t\sigma\langle\varepsilon\rangle_K s\sigma$ iff
   $(E, \sigma, pc) \in [\![ \mathcal{G}(t, S) ]\!]$ iff
   $(E, \sigma, pc) \in [\![ \mathcal{J}(t, w, S) ]\!]$.
6. If $t = \{t_1\}_k \wedge k \in K \wedge w = (b, r).w_1$, then
   $\mathcal{J}(t, w, S) = \mathcal{G}(t, S) \wedge ((\mu(b, t) \Rightarrow \mathcal{J}(b|_r, w_1, S))$
   By induction hypothesis, we have $b_{|_r}{\lhd}w_1{\rhd}_K S \equiv \mathcal{J}(b_{|_r}, w_1, S)$. We obtain
   $(E, \sigma, pc) \in [\![ t{\lhd}w{\rhd}_K S ]\!]$ iff
   $t\sigma\langle\epsilon\rangle_K S\sigma \wedge (b\sigma = t\sigma \Rightarrow (b_{|_r})\sigma\langle w_1\sigma\rangle_K S\sigma)$ iff
   $(E, \sigma, pc) \in [\![ t{\lhd}\varepsilon{\rhd}_K S \wedge (\neg\mu(b, t) \Rightarrow b_{|_r}{\lhd}w_1{\rhd}_K S)) ]\!]$ iff
   $(E, \sigma, pc) \in [\![ t{\lhd}\varepsilon{\rhd}_K S \wedge (\neg\mu(b, t) \Rightarrow \mathcal{J}(b_{|_r}, w_1, S)) ]\!]$ iff

$(E, \sigma, pc) \in [\![\mathcal{G}(t, S) \wedge (\neg\mu(b, t) \Rightarrow \mathcal{J}(b|_r, w_1, S))]\!]$ iff
$(E, \sigma, pc) \in [\![\mathcal{J}(t, w, S)]\!]$.

■

From now on, we tacitly identify $t{\lessdot}w{\gtrdot}_K S$ and $\mathcal{J}(t, w, S)$.

### 2.3.2 Embedding of SPL in TTL

In this section we prove that the predicate $Secret(t)$ of SPL logic can be expressed in TTL logic. Therefore, the secret and authentication properties that can be expressed in SPL logic (see section 2.1.2) can also be expressed in TTL logic.

Let us first to extend the notion of closure sets of messages to sets of extended terms. The definition is similar except that we have to consider two new cases:

$$\mathrm{tc}(t) = \begin{cases} t \uplus (\mathrm{tc}(t1) \cup \mathrm{tc}(t2)) & \text{if } t = (t1, t2) \\ t \uplus (\mathrm{tc}(t') \cup \mathrm{tc}(k)) & \text{if } t = \{t'\}_k \\ t \uplus \{K^{-1}\} & \text{if } t \text{ is atomic or } t \in \mathcal{BX} \\ x.f \uplus \{K^{-1}\} & \text{if } t = x \text{ is a variable} \end{cases}$$

where $f$ is a fresh function symbol.

Given a term $t$, let $\mathtt{Secret}(t)$ denote the TTL formula $\exists \overrightarrow{f} \bigvee_{S' \in \mathrm{tc}(t)} X{\lessdot}\epsilon{\gtrdot}_K S'$ where $\overrightarrow{f}$ is the set of all fresh variables $f \in \mathcal{B}$ that occur in $\mathrm{tc}(t)$.

**Theorem 2.11**

*Let $t$ be a term. Then, $[\![Secret(t)]\!] = [\![\mathtt{Secret}(t)]\!]$.*

■

**Proof:**

Using the Definitions 2.1 and 2.8, we have
$(E, \sigma, pc) \in [\![\mathtt{Secret}(t)]\!]$ iff
$(E, \sigma, pc) \in \exists \overrightarrow{f} \bigcup_{S' \in \mathrm{tc}(t)} [\![X{\lessdot}\epsilon{\gtrdot}_K S']\!]$ iff
$\exists \overrightarrow{f}\, \exists S' \in \mathrm{tc}(t)$ *s.t.* $(E, \sigma, pc) \in [\![X{\lessdot}\epsilon{\gtrdot}_K S']\!]$ iff
$\exists \overrightarrow{f}\, \exists S' \in \mathrm{tc}(t\sigma)$ *s.t.* $E\langle\epsilon\rangle_K S'\sigma$ iff (using Proposition 2.8)
$E \not\vdash t\sigma$ iff $(E, \sigma, pc) \in [\![Secret(t)]\!]$.

■

# Chapter 3

# Weakest Precondition Calculus

In this chapter we introduce a weakest precondition calculus for bounded cryptographic protocols using TTL. More precisely, we show that the weakest liberal precondition for input and respectively output actions can be expressed in the universal fragment of TTL called $\text{TTL}_\forall$.

Our formalization of bounded cryptographic protocol consists of the actions, sequential composition and non-deterministic choice. The rules for composition and non-deterministic choice are standard. Therefore, we focus on the axioms for the actions. That is, for each action, output respectively input action, we show that we can express the *weakest liberal precondition* in $\text{TTL}_\forall$. We choose weakest liberal precondition and not weakest precondition, because, in cryptographic protocol verification, nontermination is not considered as an incorrect behavior.

Later on, we will study the decidability of the satisfiability problem of a TTL formula. We prove decidability only for the existential fragment of TTL (i.e., formulae in $\text{TTL}_\exists$) and undecidability in the general case. This result entails the non-existence of a complete and effective Hoare logic [Hoa69] for bounded cryptographic protocols and full TTL.

**Definition 3.1 (weakest liberal precondition [Dij76])**

*The* **weakest liberal precondition** *of a set of configurations $\mathcal{C} \subseteq \mathcal{C}onf$ with respect to an action $\alpha$, denoted $wlp(\alpha, \mathcal{C})$, is defined to be the set of configurations $c$, such that whenever action $\alpha$ is allowed in $c$, it leads to a configuration in $\mathcal{C}$. Formally*

$$wlp(\alpha, \mathcal{C}) ::= \{(E, \sigma, pc) \mid \forall (E', \sigma', pc') \cdot (E, \sigma, pc) \xrightarrow{\alpha} (E', \sigma', pc') \;\Rightarrow\; (E', \sigma', pc') \in \mathcal{C}\}.$$

■

Given a formula $\varphi$, we use $wlp(\alpha, \varphi)$ instead of $wlp(\alpha, [\![\varphi]\!])$ to denote the weakest liberal precondition of a formula $\varphi \in$ TTL. We will show that $wlp(\alpha, \varphi)$ is effectively expressible in the universal fragment of TTL, denoted by $\text{TTL}_\forall$. The formulae of $\text{TTL}_\forall$ are of the form:

$$\varphi, \psi \quad ::= \quad X{\lhd}w{\rhd}_K S \mid (X,x){\lhd}w{\rhd}_K S \mid t{\lhd}\varepsilon{\rhd}_K x.f \mid x = t \mid pc_p^i = \ell \mid x \neq t \mid pc_p^i \neq \ell$$
$$\mid \top \mid \bot \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \forall \tilde{x}\varphi \mid \exists f \varphi$$

Let us remark that, as shown in chapter 2, most security properties (authentication and secrecy at least) can be expressed by such formulae. Further we define $wlp(\Pi, \varphi)$ action by action.

We deal only with formula of the form $(X,x){\lhd}w{\rhd}_K S$ which is equivalent with $X{\lhd}w{\rhd}_K S \wedge x{\lhd}w{\rhd}_K S$. In fact, in order to compute the weakest liberal precondition of a formula $x{\lhd}w{\rhd}_K S$ we need to relate the variable $x$ to its context $X$.

## 3.1 Output actions

Throughout this section let $\alpha = [pc_p^i = \ell] \rightarrow add(X,t); pc_p^i := \ell'$ be an output action. We show that we can express $wlp(\alpha, \varphi)$ for any formula $\varphi \in \text{TTL}_\forall$.

**Definition 3.2**

*We define the function $\widehat{wlp}$, which gives the weakest liberal precondition for an output action $\alpha = [pc_p^i = \ell] \rightarrow add(X,t); pc_p^i := \ell'$ in* $\text{TTL}_\forall$*, as follows:*

| | | | |
|---|---|---|---|
| $\widehat{wlp}(\alpha, \varphi)$ | $\stackrel{def}{=}$ | $pc_p^i = \ell \Rightarrow \varphi \wedge \mathcal{J}(t,w,S)$ | *if* $\varphi \in \{X{\lhd}w{\rhd}_K S, (X,x){\lhd}w{\rhd}_K S\}$ |
| $\widehat{wlp}(\alpha, \varphi)$ | $\stackrel{def}{=}$ | $pc_p^i = \ell \Rightarrow \varphi$ | *if* $\varphi \in \{x \neq t', x = t', pc_q^j = \ell'',$ $t{\lhd}\varepsilon{\rhd}_K x.f, \top, \bot\}$ *and* $q \neq p$ *or* $j \neq i$ |
| $\widehat{wlp}(\alpha, pc_p^i = \ell'')$ | $\stackrel{def}{=}$ | $pc_p^i = \ell \Rightarrow \ell' = \ell''$ | |
| $\widehat{wlp}(\alpha, \varphi \vee \psi)$ | $\stackrel{def}{=}$ | $\widehat{wlp}(\alpha, \varphi) \vee \widehat{wlp}(\alpha, \psi)$ | |
| $\widehat{wlp}(\alpha, \varphi \wedge \psi)$ | $\stackrel{def}{=}$ | $\widehat{wlp}(\alpha, \varphi) \wedge \widehat{wlp}(\alpha, \psi)$ | |
| $\widehat{wlp}(\alpha, \forall \tilde{x}\varphi)$ | $\stackrel{def}{=}$ | $\forall \tilde{x} \widehat{wlp}(\alpha, \varphi)$ | *if* $var\,(\alpha) \cap \tilde{x} = \emptyset$ |
| $\widehat{wlp}(\alpha, \exists f \varphi)$ | $\stackrel{def}{=}$ | $\exists f \widehat{wlp}(\alpha, \varphi)$ | |

■

We can see that for any formula $\varphi \in \text{TTL}_\forall$, $\widehat{wlp}(\alpha, \varphi) \in \text{TTL}_\forall$. Moreover, we have the following result.

**Theorem 3.1**

*The wlp-calculus of definition 3.2 is sound and complete for output actions* $\alpha = [pc_p^i = \ell] \rightarrow add(X,t); pc_p^i := \ell'$

$$wlp(\alpha, [\![\varphi]\!]) = [\![\widehat{wlp}(\alpha, \varphi)]\!].$$

■

**Proof:**

We have to prove the equivalence:

$(E,\sigma,pc) \xrightarrow{\alpha} (E',\sigma',pc') \Rightarrow (E',\sigma',pc') \in [\![\varphi]\!]$ iff $(E,\sigma,pc) \in [\![\widehat{wlp}(\alpha,\varphi)]\!]$

Let us consider the case $\varphi = X{\lhd}w{\rhd}_K S$ (respectively $\varphi = (X,x){\lhd}w{\rhd}_K S$) all others cases are direct consequences of semantics definition 2.8 and the $\widehat{wlp}$ definition 3.2.

We have $\widehat{wlp}(\alpha, X{\lhd}w{\rhd}_K S) = pc_p^i = \ell \Rightarrow X{\lhd}w{\rhd}_K S \wedge t{\lhd}w{\rhd}_K S$

To prove the equivalence we use the induction stability property of the protected modality (proposition 2.3) and the implication $m\langle w\rangle_K S \Rightarrow m\langle\varepsilon\rangle_K S$.

$(E,\sigma,pc) \xrightarrow{\alpha} (E',\sigma',pc') \Rightarrow (E',\sigma',pc') \in [\![X{\lhd}w{\rhd}_K S]\!]$ iff

$pc_p^i = \ell \wedge (E',\sigma',pc') = (E \cup \{\!|t\sigma|\!\}, \sigma, pc[pc_p^i \leftarrow \ell']) \Rightarrow (E',\sigma',pc') \in [\![X{\lhd}w{\rhd}_K S]\!]$ iff

$pc_p^i = \ell \Rightarrow (E \cup \{\!|t\sigma|\!\}, \sigma, pc[pc_p^i \leftarrow \ell']) \in [\![X{\lhd}w{\rhd}_K S]\!]$ iff

$pc_p^i = \ell \Rightarrow E \cup \{\!|t\sigma|\!\}\langle w\sigma\rangle_K S\sigma$ iff

$pc_p^i = \ell \Rightarrow E\langle w\sigma\rangle_K S\sigma \wedge t\sigma\langle w\sigma\rangle_K S\sigma$ iff

$(E,\sigma,pc) \in [\![pc_p^i = \ell \Rightarrow X{\lhd}w{\rhd}_K S \wedge t{\lhd}w{\rhd}_K S]\!]$ iff

$(E,\sigma,pc) \in [\![\widehat{wlp}(\alpha, X{\lhd}w{\rhd}_K S)]\!]$

■

Let us give a short example of a *wlp*-calculus for an output action:

**Example 3.1**

*Let $t = (\{y,M\}_k, \{N,z\}_k)$ be a term, $\alpha = [pc_p^i = \ell] \rightarrow add(X,t); pc_p^i := \ell'$ an output action and $\varphi = X{\lhd}(\{N,x\}_k, 01){\rhd}_K N_2$ be a postcondition. Then,*

$$\widehat{wlp}(\alpha,\varphi) \;=\; pc_p^i = \ell \Rightarrow X{\lhd}(\{N,x\}_k,01){\rhd}_K N_2 \wedge t{\lhd}(\{N,x\}_k,01){\rhd}_K N_2$$

$$\begin{aligned}
t{\lhd}(\{N,x\}_k,01){\rhd}_K N_2 &= \mathcal{J}(t,(\{N,x\}_k,01),N_2)\\
&= (\{y,M\}_k,\{N,z\}_k){\lhd}(\{N,x\}_k,01){\rhd}_K N_2\\
&= \{y,M\}_k{\lhd}(\{N,x\}_k,01){\rhd}_K N_2 \wedge \{N,z\}_k{\lhd}(\{N,x\}_k,01){\rhd}_K N_2\\
&\quad \wedge(\{y,M\}_k,\{N,z\}_k) \neq N_2\\
&= \{y,M\}_k \neq N_2 \wedge [(y = N \wedge x = M) \Rightarrow x{\lhd}\varepsilon{\rhd}_K N_2]\\
&\quad \wedge\{N,z\}_k \neq N_2 \wedge [z = x \Rightarrow x{\lhd}\varepsilon{\rhd}_K N_2]\\
&= [(y = N \wedge x = M) \Rightarrow x{\lhd}\varepsilon{\rhd}_K N_2] \wedge [(z = x) \Rightarrow x{\lhd}\varepsilon{\rhd}_K N_2]
\end{aligned}$$

*Therefore,*

$$\begin{aligned}
\widehat{wlp}(\alpha,\varphi) \;=\; & pc_p^i = \ell \Rightarrow X{\lhd}(\{N,x\}_k,01){\rhd}_K N_2 \wedge\\
& [(y = N \wedge x = M) \Rightarrow x{\lhd}\varepsilon{\rhd}_K N_2] \wedge [(z = x) \Rightarrow x{\lhd}\varepsilon{\rhd}_K N_2]
\end{aligned}$$

■

## 3.2 Input actions

Let $\alpha = [pc_p^i = \ell \wedge in(X, t(\tilde{x}))] \rightarrow pc_p^i := \ell'$ be an input action. We show that we can express $wlp(\alpha, \varphi)$ for any formula $\varphi \in \text{TTL}_\forall$. To do so, we need to give a few definitions and prove several intermediate results.

The definition 2.7 has introduced well-formed modalities i.e, $(w_i, S_i)_{i\in I}$ is well-formed. As now we are dealing with formulae, we extend this notion to well-formed formula, in the same sense.

**Definition 3.3 (well-formed formula)**

*A formula $\Phi$ is* well-formed*, if for any term transducer $w$ and closed set of extended terms $S$, whenever $\Phi \Rightarrow X{\lhd}w{\rhd}_K S$, there exist $(w_i, S_i)_{i\in I}$ well-formed, such that $\Phi \Rightarrow \bigwedge_{i\in I} X{\lhd}w_i{\rhd}_K S_i$ and $(w, S) \in (w_i, S_i)_{i\in I}$.*

■

The main property satisfied by well-formed formulae is a stability property given by the following corollary, which is a direct consequence of definitions 2.7 and 3.3 and theorem 2.9.

**Corollary 3.2**

*Let $\Phi$ be a well-formed formula such that $\Phi \Rightarrow X{\lhd}w{\rhd}_K S$ and let $(E, \sigma, pc) \in [\![\Phi]\!]$. If $m$ is a message such that $E\sigma \vdash m$, then $\Phi \Rightarrow m\langle w\sigma\rangle_K S\sigma$.*

■

The property of this corollary turns out to be crucial for developing a complete weakest precondition calculus and where well-formedness is preserved. Therefore, we introduce the function $\mathcal{H}$ which takes as arguments a formula $X{\lhd}(t,p)\cdot w{\rhd}_K S$ and computes the weakest (the largest w.r.t. set inclusion) well-formed formula $\mathcal{H}(X{\lhd}(t,p)\cdot w{\rhd}_K S)$, such that $\mathcal{H}(X{\lhd}(t,p)\cdot w{\rhd}_K S) \Rightarrow X{\lhd}(t,p)\cdot w{\rhd}_K S$. The intuition follows from the Definition 2.7: in the case that the term transducer $(t,p)$ contains an inner term transducer $(t_1,p_1)$, either $t$ cannot be built or it doesn't help; moreover, the formula $\mathcal{H}(X{\lhd}(t,p)\cdot w{\rhd}_K S)$ is closed with respect to suffixes of $w$. The inductive definition of $\mathcal{H}$ is given below:

$$\mathcal{H}(X{\lhd}(t,p)\cdot w{\rhd}_K S) = \begin{cases} X{\lhd}(t,p)\cdot w{\rhd}_K S \wedge \mathcal{H}(X{\lhd}w{\rhd}_K S) \\ \qquad \text{– if } \mathrm{NT}((t,p)) \text{ is undefined} \\ \\ X{\lhd}(t,p)\cdot w{\rhd}_K S \wedge \mathcal{H}(X{\lhd}w{\rhd}_K S) \\ \wedge\big(\mathcal{H}(X{\lhd}(b_1,p_1)\cdot w{\rhd}_K S) \vee \bigvee_{S'\in tc(b)} X{\lhd}\epsilon{\rhd}_K S'\big) \\ \qquad \text{– if } (b_1,p_1) = \mathrm{NT}((b,p)) \end{cases}$$

A direct consequence of $\mathcal{H}$ definition and the well-formed formula definition 3.3 is the following proposition:

**Proposition 3.3**

*Let $\Phi$ be a well-formed formula. Let $w$ be a nonempty term transducer and $S$ a closed set of terms such that $\Phi \Rightarrow X{\lhd}w{\rhd}_K S$. Then $\Phi \Rightarrow \mathcal{H}(X{\lhd}w{\rhd}_K S)$.*

■

We remind from section 2.2, that given a term $t$, $\mathtt{Secret}(t) ::= \bigvee_{S' \in tc(t)} X{\lhd}\epsilon{\rhd}_K S'$. Therefore, being in a state $(E, \sigma, pc)$, in order to be able to make an input $t(\tilde{x})$, such that $\tilde{x}$ are instantiated by $\rho$, it must be that $(E, \sigma, pc) \notin [\![\mathtt{Secret}(t\rho)]\!]$, namely $t(\sigma \oplus \rho)$ must be derivable from $E$.

The following lemma gives the weakest condition that has to be satisfied in a configuration $c$, such that if in the next step $x$ is instantiated by an input action of the form $[pc_p^i = \ell \wedge in(X, t(\tilde{x}))] \rightarrow pc_p^i := \ell'$, where $x \in \tilde{x}$, the reached configuration $c'$ satisfies $(X, x){\lhd}w{\rhd}_K S$.

The key idea can be explained by considering the sequence of actions:

$$\begin{array}{ll} [pc = \ell_0 \wedge in(X, \{x\}_k)] & \rightarrow pc := \ell_1 \\ [pc = \ell_1] & \rightarrow add(X, x); pc := \ell_2 \end{array}$$

If a secret $s$ that appears in $x$, it has to be protected, then it has to appear in $x$ under an encryption. Thus, before executing the sequence of actions given above, it should be the case that even if we provide the intruder with the term transducer that takes as input $\{x\}_k$ and yields $x$, it is not possible to derive $s$.

**Lemma 3.4**

*Let $t$ be a term, $S$ a closed set of extended terms, $w$ a term transducer, $x$ a variable and $P_{x,t}$ the set of critical positions of $x$ in $t$. Let*

$$\mathcal{K}(t, x, w, S) = X{\lhd}w{\rhd}_K S \wedge \bigwedge_{p = FP(t, p_x), p_x \in P_{x,t}} \mathcal{H}(X{\lhd}(t_{|_p}, p^{-1}p_x) \cdot w{\rhd}_K S).$$

*Let $E$ be a set of terms, $pc$ and $pc'$ control point vectors, and $\rho$, $\sigma$ ground substitutions such that $dom(\rho) = \tilde{x}$, $x \in \tilde{x}$, $dom(\sigma) \cap \tilde{x} = \emptyset$. Let $\Phi$ a well-formed formula such that whenever $E \vdash t(\sigma \oplus \rho)$, it holds*

*$(E, \sigma \oplus \rho, pc') \in [\![(X, x){\lhd}w{\rhd}_K S]\!]$ iff $(E, \sigma, pc) \in [\![\Phi]\!]$*

*Then $[\![\Phi]\!] = [\![\rho(\mathcal{K}(t, x, w, S))]\!]$.*

■

**Proof:**

"⇒": $(E, \sigma \oplus \rho, pc') \in [\![(X, x){\lhd}w{\rhd}_K S]\!] \Rightarrow (E, \sigma, pc) \in [\![\rho(\mathcal{K}(t, x, w, S))]\!]$

We have that $(E, \sigma \oplus \rho, pc') \in [\![(X, x){\lhd}w{\rhd}_K S]\!]$ iff

$E\langle w(\sigma \oplus \rho)\rangle_K S(\sigma \oplus \rho) \wedge x\rho\langle w(\sigma \oplus \rho)\rangle_K S(\sigma \oplus \rho) \qquad (*)$

We have to prove that

$$(E,\sigma,pc) \in [\![X{\lhd}w\rho{\rhd}_K S\rho \wedge \bigwedge_{\substack{p = \mathrm{FP}(t,p_x)\\ p_x \in P_{x,t}}} \mathcal{H}(X{\lhd}(t\rho_{|_p}, p^{-1}p_x)\cdot w\rho{\rhd}_K S\rho)]\!].$$

From $(*)$ we have $(E,\sigma,pc) \in [\![X{\lhd}w\rho{\rhd}_K S\rho]\!]$.

Let $t_1$ denote $t_{|_p}$ and $p_1$ denote $p^{-1}p_x$, as $p = \mathrm{FP}(t,p_x)$ we have $t_{1_{|_{p_1}}} = x \qquad (**)$

We prove that

$$(E,\sigma,pc) \in [\![ \bigwedge_{\substack{p = \mathrm{FP}(t,p_x)\\ p_x \in P_{x,t}}} \mathcal{H}(X{\lhd}(t_1\rho, p_1)\cdot w\rho{\rhd}_K S\rho)]\!].$$

Let suppose that $\forall p_x \in P_{x,t} \exists \mathrm{FP}(t,p_x)$. If $\exists p_x \in P_{x,t}$ s. t. $\neg\exists \mathrm{FP}(t,p_x)$ then that $p_x$ is not take into account in the conjunction.

Using $(**)$, we obtain the formula $(*)$ is equivalent to

$E\langle w(\sigma \oplus \rho)\rangle_K S(\sigma \oplus \rho) \wedge t_1(\sigma \oplus \rho)_{|_{p_1}} \langle w(\sigma \oplus \rho)\rangle_K S(\sigma \oplus \rho)$

which imply:

$E\langle (t_1(\sigma \oplus \rho), p_1)\cdot w(\sigma \oplus \rho)\rangle_K S(\sigma \oplus \rho)$

Now the assertion follows from the proposition 3.3, so we have:

$(E,\sigma,pc) \in [\![\mathcal{H}(X{\lhd}(t_1\rho,p_1)\cdot w\rho{\rhd}_K S\rho))]\!]$

hence, $(E,\sigma,pc) \in [\![\rho(\mathcal{K}(t,x,w,S))]\!]$

"$\Leftarrow$": $(E,\sigma,pc) \in [\![\rho(\mathcal{K}(t,U,x,w,s))]\!] \Rightarrow (E,\sigma\oplus\rho,pc') \in [\![(X,x){\lhd}w{\rhd}_K s]\!]$

$(E,\sigma,pc) \in [\![\rho(\mathcal{K}(t,x,w,s))]\!]$ iff

$$(E,\sigma,pc) \in [\![X{\lhd}w\rho{\rhd}_K S\rho \wedge \bigwedge_{\substack{p = \mathrm{FP}(t,p_x)\\ p_x \in P_{x,t}}} \mathcal{H}(X{\lhd}(t\rho_{|_p}, p^{-1}p_x)\cdot w\rho{\rhd}_K S\rho)]\!] \text{ iff}$$

$$E\langle w(\sigma{\oplus}\rho)\rangle_K S(\sigma{\oplus}\rho){\wedge} \bigwedge_{\substack{p = \mathrm{FP}(t,p_x)\\ p_x \in P_{x,t}}} (E,\sigma,pc) \in [\![\mathcal{H}(X{\lhd}(t\rho_{|_p}, p^{-1}p_x){\cdot}w\rho{\rhd}_K S\rho)]\!] \quad (***)$$

From $(E,\sigma,pc) \in [\![\mathcal{H}(X{\lhd}(t\rho_{|_p}, p^{-1}p_x)\cdot w\rho{\rhd}_K S\rho)]\!]$ it follows that

$E\langle (t\rho_{|_p}, p^{-1}p_x)\cdot w(\sigma \oplus \rho)\rangle_K S(\sigma \oplus \rho)$.

Also we know that $E \vdash\ t(\sigma \oplus \rho)$

and by construction, the formula $\mathcal{H}(X{\lhd}(t\rho_{|_p}, p^{-1}p_x)\cdot w\rho{\rhd}_K S)$ is well-formed.

Hence, for $m = t(\sigma \oplus \rho)$ using the corollary 3.2 we have

$t(\sigma \oplus \rho)\langle (t\rho_{|_p}, p^{-1}p_x)\cdot w(\sigma \oplus \rho)\rangle_K S(\sigma \oplus \rho)$

and from the first equivalence of the proposition 2.4 we obtain

$t(\sigma \oplus \rho)_{|_{p_x}} \langle w(\sigma \oplus \rho)\rangle_K S(\sigma \oplus \rho)$
but $t_{|_{p_x}} = x$ hence $x(\sigma \oplus \rho)\langle w(\sigma \oplus \rho)\rangle_K S(\sigma \oplus \rho)$ and from $(***)$ we obtain
$E\langle w(\sigma \oplus \rho)\rangle_K S(\sigma \oplus \rho) \wedge x(\sigma \oplus \rho)\langle w(\sigma \oplus \rho)\rangle_K S(\sigma \oplus \rho)$
$(E, \sigma \oplus \rho, pc') \in [\![(X,x)\lessdot w\gtrdot_K S]\!]$.

■

Now we are ready to introduce for input actions the weakest liberal preconditions for all formulae in TTL$_\forall$.

**Definition 3.4**

*We define the function* $\widehat{wlp}$*, which gives the weakest liberal preconditions for an input action* $\alpha = [pc_p^i = \ell \wedge in(X, t(\tilde{x}))] \rightarrow pc_p^i := \ell'$ *in* TTL$_\forall$*, as follows:*

$$
\begin{array}{lcll}
\widehat{wlp}(\alpha, (X,x)\lessdot w\gtrdot_K S) & \stackrel{def}{=} & pc_p^i = \ell \Rightarrow (\mathit{Secret}(t) \vee \mathcal{K}(t,x,w,S)) & \\
 & & & \text{if } x \in \tilde{x} \\
\widehat{wlp}(\alpha, \varphi) & \stackrel{def}{=} & pc_p^i = \ell \Rightarrow (\mathit{Secret}(t) \vee \varphi) & \\
 & & & \text{if } \varphi \in \{X\lessdot w\gtrdot_K S, (X,y)\lessdot w\gtrdot_K S, \top, \bot, \\
 & & & \quad t\lessdot \varepsilon \gtrdot_K x.f, x \neq t', x = t', pc_q^j = \ell''\} \\
 & & & \quad \text{and } y \notin \tilde{x} \text{ and } q \neq p \text{ or } j \neq i \\
\widehat{wlp}(\alpha, pc_p^i = \ell'') & \stackrel{def}{=} & pc_p^i = \ell \Rightarrow (\mathit{Secret}(t) \vee \ell' = \ell'') & \\
\widehat{wlp}(\alpha, \varphi \vee \psi) & \stackrel{def}{=} & \widehat{wlp}(\alpha, \varphi) \vee \widehat{wlp}(\alpha, \psi) & \\
\widehat{wlp}(\alpha, \varphi \wedge \psi) & \stackrel{def}{=} & \widehat{wlp}(\alpha, \varphi) \wedge \widehat{wlp}(\alpha, \psi) & \\
\widehat{wlp}(\alpha, \forall \tilde{y} \varphi) & \stackrel{def}{=} & \forall \tilde{y} \widehat{wlp}(\alpha, \varphi) & \\
 & & & \text{if } var\,(\alpha) \cap \tilde{y} = \emptyset \\
\widehat{wlp}(\alpha, \exists f \varphi) & \stackrel{def}{=} & \exists f \widehat{wlp}(\alpha, \varphi) &
\end{array}
$$

■

**Lemma 3.5**

*Let* $\alpha = [pc_p^i = \ell \wedge in(X, t(\tilde{x}))] \rightarrow pc_p^i := \ell'$ *be an input action,* $\varphi$ *a* TTL$_\forall$ *formula and* $\rho \in \Gamma(\tilde{x})$ *a ground substitution with* $\mathrm{dom}(\rho) = \tilde{x}$*. Then*

$$\rho(\widehat{wlp}(\alpha, \varphi)) \;\equiv\; \big((pc_p^i = \ell) \Rightarrow (\mathit{Secret}(t\rho) \vee \rho(\varphi))\big)$$

■

**Proof:**

By induction on the structure of $\varphi$. In the sequel, we use implicitly that $\sigma$ is a ground substitution such that $dom(\sigma) \cap dom(\rho) = \emptyset$.

- $\varphi = X{\lhd}w{\rhd}_K S$. Then

$$\begin{array}{rcl}\rho(\widehat{wlp}(\alpha,\varphi)) & = & \rho(\widehat{wlp}(\alpha, X{\lhd}w{\rhd}_K S)) \\ & = & \rho((pc_p^i = \ell) \Rightarrow (\mathtt{Secret}(t) \vee X{\lhd}w{\rhd}_K S)) \\ & = & (pc_p^i = \ell) \Rightarrow (\mathtt{Secret}(t\rho) \vee \rho(X{\lhd}w{\rhd}_K S)) \\ & = & (pc_p^i = \ell) \Rightarrow (\mathtt{Secret}(t\rho) \vee \rho(\varphi))\end{array}$$

- $\varphi = (X, x){\lhd}w{\rhd}_K S$ and $x \in \tilde{x}$. Obvious using Lemma 3.4.

- $\varphi = x \neq t'$ and $\varphi = x = t'$. Obvious.

- $\varphi = \top$ and $\varphi = \bot$. Obvious. Using that for any formulas $\psi$, it holds $\psi \vee \top \equiv \top$ and $\psi \vee \bot \equiv \psi$.

- $\varphi = \varphi_1 \vee \varphi_2$. Then using the induction hypothesis we obtain

$$\begin{array}{rcl}\rho(\widehat{wlp}(\alpha,\varphi)) & = & \rho(\widehat{wlp}(\alpha, \varphi_1 \vee \varphi_2)) \\ & = & \rho(\widehat{wlp}(\alpha, \varphi_1)) \vee \rho(\widehat{wlp}(\alpha, \varphi_2)) \\ & = & ((pc_p^i = \ell) \Rightarrow (\mathtt{Secret}(t\rho) \vee \rho(\varphi_1)))\vee \\ & & ((pc_p^i = \ell) \Rightarrow (\mathtt{Secret}(t\rho) \vee \rho(\varphi_2))) \\ & = & (pc_p^i = \ell) \Rightarrow ((\mathtt{Secret}(t\rho) \vee \rho(\varphi_1)) \vee (\mathtt{Secret}(t\rho) \vee \rho(\varphi_2))) \\ & = & (pc_p^i = \ell) \Rightarrow (\mathtt{Secret}(t\rho) \vee (\rho(\varphi_1) \vee \rho(\varphi_2))) \\ & = & (pc_p^i = \ell) \Rightarrow (\mathtt{Secret}(t\rho) \vee \rho(\varphi))\end{array}$$

- $\varphi = \varphi_1 \wedge \varphi_2$. Then using the induction hypothesis we obtain

$$\begin{array}{rcl}\rho(\widehat{wlp}(\alpha,\varphi)) & = & \rho(\widehat{wlp}(\alpha, \varphi_1 \wedge \varphi_2)) \\ & = & \rho(\widehat{wlp}(\alpha, \varphi_1)) \wedge \rho(\widehat{wlp}(\alpha, \varphi_2)) \\ & = & ((pc_p^i = \ell) \Rightarrow (\mathtt{Secret}(t\rho) \vee \rho(\varphi_1)))\wedge \\ & & ((pc_p^i = \ell) \Rightarrow (\mathtt{Secret}(t\rho) \vee \rho(\varphi_2))) \\ & = & (pc_p^i = \ell) \Rightarrow ((\mathtt{Secret}(t\rho) \vee \rho(\varphi_1)) \wedge (\mathtt{Secret}(t\rho) \vee \rho(\varphi_2))) \\ & = & (pc_p^i = \ell) \Rightarrow (\mathtt{Secret}(t\rho) \vee (\rho(\varphi_1) \wedge \rho(\varphi_2))) \\ & = & (pc_p^i = \ell) \Rightarrow (\mathtt{Secret}(t\rho) \vee \rho(\varphi))\end{array}$$

- $\varphi = \forall \tilde{y}\varphi_1$. We can suppose that $var\,(t) \cap \tilde{y} = \emptyset$ . Then, using that $\rho$ and $\rho_1$ are ground substitutions such that $dom(\rho) \cap dom(\rho_1) = \emptyset$, we obtain

$$\begin{array}{rcl}[\![\rho(\widehat{wlp}(\alpha,\varphi))]\!] & = & [\![\rho(\widehat{wlp}(\alpha, \forall\tilde{y}\varphi_1)]\!] \\ & = & [\![\rho(\forall\tilde{y}(\widehat{wlp}(\alpha, \varphi_1)))]\!] \\ & = & \rho(\bigcap_{\rho_1 \in \Gamma(\tilde{y})} [\![\rho_1(\widehat{wlp}(\alpha, \varphi_1))]\!]) \\ & = & \bigcap_{\rho_1 \in \Gamma(\tilde{y})} [\![\rho(\rho_1(\widehat{wlp}(\alpha, \varphi_1)))]\!] \\ & = & \bigcap_{\rho_1 \in \Gamma(\tilde{y})} [\![\rho_1(\rho(\widehat{wlp}(\alpha, \varphi_1)))]\!] \\ & = & \bigcap_{\rho_1 \in \Gamma(\tilde{y})} [\![\rho_1((pc_p^i = \ell) \Rightarrow (\mathtt{Secret}(t\rho) \vee \rho(\varphi_1)))]\!]\end{array}$$

$$
\begin{aligned}
&= \bigcap_{\rho_1 \in\Gamma(\tilde{y})} ([\![\rho_1(pc_p^i \neq \ell)]\!] \cup ([\![\rho_1(\mathtt{Secret}(t\rho))]\!] \cup [\![\rho_1(\rho(\varphi_1))]\!])) \\
&= \bigcap_{\rho_1 \in\Gamma(\tilde{y})} ([\![pc_p^i \neq \ell]\!] \cup ([\![\mathtt{Secret}(t\rho)]\!] \cup [\![\rho_1(\rho(\varphi_1))]\!])) \\
&= [\![pc_p^i \neq \ell]\!] \cup ([\![\mathtt{Secret}(t\rho)]\!] \cup \bigcap_{\rho_1 \in\Gamma(\tilde{y})} [\![\rho(\rho_1(\varphi_1))]\!]) \\
&= [\![pc_p^i \neq \ell]\!] \cup ([\![\mathtt{Secret}(t\rho)]\!] \cup \rho(\bigcap_{\rho_1 \in\Gamma(\tilde{y})} [\![\rho_1(\varphi_1)]\!])) \\
&= [\![pc_p^i \neq \ell]\!] \cup ([\![\mathtt{Secret}(t\rho)]\!] \cup \rho([\![\forall\tilde{y}\varphi_1]\!])) \\
&= [\![pc_p^i \neq \ell]\!] \cup ([\![\mathtt{Secret}(t\rho)]\!] \cup \rho([\![\varphi]\!])) \\
&= [\![pc_p^i \neq \ell]\!] \cup ([\![\mathtt{Secret}(t\rho)]\!] \cup [\![\rho(\varphi)]\!]) \\
&= [\![(pc_p^i = \ell) \Rightarrow (\mathtt{Secret}(t\rho) \vee \rho(\varphi))]\!]
\end{aligned}
$$

and hence we obtain that $\rho(\widehat{wlp}(\alpha, \varphi)) \equiv (pc_p^i = \ell) \Rightarrow (\mathtt{Secret}(t\rho) \vee \rho(\varphi))$.

- $\varphi = \exists f \varphi_1$. Using that $f_1$ is a fresh set of function symbols in $\mathcal{B}$ such that $f \cap f_1 = \emptyset$, we obtain

$$
\begin{aligned}
\rho(\widehat{wlp}(\alpha, \varphi)) &= \rho(\widehat{wlp}(\alpha, \exists f \varphi_1) \\
&= \rho(\exists f \widehat{wlp}(\alpha, \varphi_1) \\
&= \exists f \rho(\widehat{wlp}(\alpha, \varphi_1) \\
&= \exists f (pc_p^i = \ell) \Rightarrow (\mathtt{Secret}(t\rho) \vee \rho(\varphi_1)) \\
&= (pc_p^i = \ell) \Rightarrow \exists f (\mathtt{Secret}(t\rho) \vee \exists f \rho(\varphi_1))
\end{aligned}
$$

Since $\mathtt{Secret}(t\rho) = \exists f_1 \bigvee_{S' \in \mathrm{tc}(t)} X \lessdot \epsilon \gtrdot_K S'$ where $f_1$ is the set of all fresh variables of $\mathcal{B}$ that occur in $\mathrm{tc}(t)$ ($f \cap f_1 = \emptyset$) and $t$ is not an extended term we obtain $\exists f (\mathtt{Secret}(t\rho) = \mathtt{Secret}(t\rho)$. Then,

$$
\rho(\widehat{wlp}(\alpha, \varphi)) = (pc_p^i = \ell) \Rightarrow (\mathtt{Secret}(t\rho) \vee \rho(\exists f \varphi_1))
$$

and hence we obtain that $\rho(\widehat{wlp}(\alpha, \varphi)) \equiv (pc_p^i = \ell) \Rightarrow (\mathtt{Secret}(t\rho) \vee \rho(\varphi))$.

■

We can see that for any formula $\varphi \in \mathrm{TTL}_\forall$, $\widehat{wlp}(\alpha, \varphi) \in \mathrm{TTL}_\forall$. We have the following theorem:

**Theorem 3.6**

*Let $\alpha = [pc_p^i = \ell \wedge in(X, t(\tilde{x}))] \rightarrow pc_p^i := \ell'$ an input action and $\varphi \in \mathrm{TTL}_\forall$ an universal* TTL *formula. The wlp-calculus defined by* $\mathrm{WLP}(\alpha, \varphi) = \forall\tilde{x} \cdot \widehat{wlp}(\alpha, \varphi)$ *is sound and complete.*

$$
wlp(\alpha, [\![\varphi]\!]) = [\![\mathrm{WLP}(\alpha, \varphi)]\!].
$$

■

**Proof:**

In the sequel $pc$ is a control points vector, $E$ is a set of messages and $\sigma$ is a ground substitution such that $\tilde{x} \cap dom(\sigma) = \emptyset$.

$(E, \sigma, pc) \in wlp(\alpha, [\![\varphi]\!])$ iff

$\forall \rho \in \Gamma(\tilde{x}).\ (pc_p^i = l) \Rightarrow \big(E \vdash t(\sigma \oplus \rho) \Rightarrow (E, \sigma \oplus \rho, pc) \in [\![\varphi]\!]\big)$ iff

$\forall \rho \in \Gamma(\tilde{x}).\ (pc_p^i = l) \Rightarrow \big(E \nvdash t(\sigma \oplus \rho) \vee (E, \sigma \oplus \rho, pc) \in [\![\varphi]\!]\big)$ iff

$\forall \rho \in \Gamma(\tilde{x}).\ \big((E, \sigma, pc) \in [\![pc_p^i = l]\!]\big) \Rightarrow \big((E, \sigma, pc) \in [\![\mathtt{Secret}(t\rho)]\!] \cup [\![\rho(\varphi)]\!]\big)$ iff

$(E, \sigma, pc) \in \bigcap_{\rho \in \Gamma(\tilde{x})} [\![(pc_p^i = l) \Rightarrow (\mathtt{Secret}(t\rho) \vee \rho(\varphi))]\!]$ iff

$(E, \sigma, pc) \in \bigcap_{\rho \in \Gamma(\tilde{x})} [\![\rho(\hat{wlp}(\alpha, \varphi)]\!]$ iff

$(E, \sigma, pc) \in [\![\forall \tilde{x}(\hat{wlp}(\alpha, \varphi))]\!]$.

■

Let us give a short example of a wlp-calculus for an input action:

**Example 3.2**

*Let $K = \{k\}$, $inv(k) = k$ and $t = (\{z, \{y\}_k\}_k, \{I, \{z, A\}_k\}_{k_I})$ be a term, $\alpha = [pc_p^i = \ell \wedge in(X, t)] \rightarrow pc_p^i := \ell'$ be an input action, where $z$ is a variable instantiated by the action, and $\varphi = (X, z)\prec\varepsilon\succ_K N_2$ a postcondition. Then,*

$$\begin{aligned}
\widehat{wlp}(\alpha, \varphi) &= pc_p^i = \ell \Rightarrow (\mathcal{S}ecret(t) \vee \mathcal{K}(t, z, \varepsilon, N_2)) \\
&= pc_p^i = \ell \Rightarrow \big(\exists f_1 \exists f_2 \exists f_3 \bigvee_{S' \in tc(t)} X\prec\epsilon\succ_K S' \vee \\
&\qquad (X\prec(\{z, \{y\}_k\}_k, 00)\succ_K N_2 \wedge X\prec(\{z, A\}_k, 00)\succ_K N_2)\big)
\end{aligned}$$

*where $f_1, f_2, f_3$ are fresh symbols of $\mathcal{B}$ and*

$$\begin{aligned}
tc(t) = \{\{&t; \{z, \{y\}_k\}_k; (z, \{y\}_k); z.f_1; k\}; \\
&\{t; \{z, \{y\}_k\}_k; (z, \{y\}_k); \{y\}_k; y.f_2; k\}; \\
&\{t; \{z, \{y\}_k\}_k; (z, \{y\}_k); \{y\}_k; k\}; \\
&\{t; \{z, \{y\}_k\}_k; k\}; \\
&\{t; \{I, \{z, A\}_k\}_{k_I}); (I, \{z, A\}_k); I; k\}; \\
&\{t; \{I, \{z, A\}_k\}_{k_I}); (I, \{z, A\}_k); \{z, A\}_k; (z, A); z.f_3; k\}; \\
&\{t; \{I, \{z, A\}_k\}_{k_I}); (I, \{z, A\}_k); \{z, A\}_k; (z, A); A; k\}; \\
&\{t; \{I, \{z, A\}_k\}_{k_I}); (I, \{z, A\}_k); \{z, A\}_k; k\}; \\
&\{t; \{I, \{z, A\}_k\}_{k_I}); k_I; k\}\}
\end{aligned}$$

■

## 3.3 Collecting the results together

**Definition 3.5**

*We define the formula* $\text{WLP}(\alpha, \varphi)$ *as follows:*

$$\text{WLP}(\alpha, \varphi) = \begin{cases} \widehat{wlp}(\alpha, \varphi) & \text{if } \alpha = [pc_p^i = \ell] \rightarrow add(X, t); pc_p^i := \ell' \\ \forall \tilde{x} \cdot \widehat{wlp}(\alpha, \varphi) & \text{if } \alpha = [pc_p^i = \ell \wedge in(X, t(\tilde{x}))] \rightarrow pc_p^i := \ell' \end{cases}$$

■

The following theorem collects the results of theorems 3.1 and 3.6. It establish the correctness and completeness of our wlp-calculus for bounded cryptographic protocols and $\text{TTL}_\forall$.

**Theorem 3.7**

*The wlp-calculus of Definition 3.5 is sound and complete. Formally, let* $\alpha$ *be any action and* $\varphi$ *any formula in* $\text{TTL}_\forall$*. Then,*

$$wlp(\alpha, [\![\varphi]\!]) = [\![\text{WLP}(\alpha, \varphi)]\!].$$

■

The Hoare logic consisting of the inference rules for composition and the axiom schema $\{wlp(\alpha, \varphi)\}\alpha\{\varphi\}$, for each action, is sound and complete.

# Chapter 4

# Decidability of TTL

In this chapter we study the decidability of the satisfiability problem of a TTL formula.

First we prove decidability for the existential fragment of TTL (i.e., formulae in $\text{TTL}_\exists$). Since we showed in the previous chapter that given a formula $\varphi$ in $\text{TTL}_\forall$ and a bounded cryptographic protocol $\Pi$, one can compute $\text{WLP}(\Pi, \varphi)$, decidability of the satisfiability of existential formulae yields a decision procedure. Indeed, assume that we are given an existential formula $\psi$ and a property $\varphi$ in $\text{TTL}_\forall$, assume also that we are given a bounded cryptographic protocol $\Pi$ then $\{\psi\}\Pi\{\varphi\}$ is true iff $\psi \wedge \neg\text{WLP}(\Pi, \varphi)$ is not satisfiable.

To prove decidability for existential formulae we follow a rule based approach (see [JK91, Com91] for two surveys) i.e.:

1. We introduce a set of formulae in *solved form*. For these formulae it is "easy" to decide whether a model exists.
2. We introduce a set of rewriting rules that transform any formula $\varphi$ of $\text{TTL}_\exists$, into a set of solved formulae, such that $\varphi$ is satisfiable iff one the formulae in solved form is satisfiable.
3. We prove soundness and completeness of these rules.
4. We also prove their termination for a given control i.e, that normal forms are reached and that normal forms are indeed in solved form.

The reduction of a formula of $\text{TTL}_\exists$ into a set of solved formulae is done in three phases.

1. We define a *preliminary form* and we introduce a set of rewriting rules to transform any formula in the fragment that interest us, into a preliminary form.
2. We define an *intermediate form* and we introduce a set of rewriting rules to transform any formula in preliminary form into an intermediate form.
3. For each formula in intermediate form, we show how to reduce its satisfiability to the satisfiability of a set of *saturated formulae* in intermediate form; moreover, for

each saturated formula in intermediate form, we can extract a formula in solved form such that a model exists for the formula in intermediate form if and only if the extracted formula in solved form is satisfiable.

Second, we show that $\text{TTL}_\exists$ satisfiability problem is $NP$-complete, this is an alternative prove for the results of [AL00, RT01, MS01, Che03, Tur03] for Dolev-Yao model with atomic keys. First we prove that our satisfiability problem is in $NP$, given that the size of the solution space is polynomially bounded in the size of the problem. The proof is based on a result concerning the polynomial bound of the DAG-size of a substitution in a normal attack (Theorem 1 of [RT01]). Then, we show that the problem of deciding satisfiability for a $\text{TTL}_\exists$ formula is $NP$-complete. We prove the $NP$-hardness by defining a polynomial reduction of 3-SAT problem to $\text{TTL}_\exists$ satisfiability problem, i.e. we construct a $\text{TTL}_\exists$-formula such that it is satisfiable if and only if the 3-SAT problem has solution.

In the general case the satisfiability problem of a TTL formula is undecidable. We show this result by encoding the Post's correspondence problem as a satisfiability problem in TTL.

## 4.1 Decidability of $\text{TTL}_\exists$

The formulae of $\text{TTL}_\exists$ are of the form:

$$\begin{aligned}\varphi,\psi \quad ::= \quad & X{\lessdot}w{\gtrdot}_K S \mid x{\lessdot}w{\gtrdot}_K S \mid t{\lessdot}\varepsilon{\gtrdot}_K x.f \mid X{\lessdot}\not{w}{\gtrdot}_K S \mid x{\lessdot}\not{w}{\gtrdot}_K S \mid t{\lessdot}\not{\varepsilon}{\gtrdot}_K x.f \\ & \mid x = t \mid pc_p^i = \ell \mid x \neq t \mid pc_p^i \neq \ell \mid \top \mid \bot \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \exists \tilde{x}\varphi \mid \forall f\varphi\end{aligned}$$

where $X$ is a fixed second-order variable that ranges over sets of messages, $S$ is a second-order variable that ranges over closed sets of extended terms, $f$ is a meta-variable that ranges over $\mathcal{B}$, and $x$ is a meta-variable that ranges over the set $\mathcal{X}$ of first-order variables. First-order variables range over messages; $t$ is a meta-variable over terms and $w \in (\mathcal{T} \times \mathcal{P}os)^*$ is a term transducer.

### 4.1.1 A decidable fragment $\text{TTL}_\exists$

We do not consider formulae of the form $pc_p^i = \ell$. It will be clear that adding these formulae does not add any technical difficulty; it is only cumbersome to consider them here. Also, we do not consider formulae of the form $X{\lessdot}w{\gtrdot}_K s$ or $x{\lessdot}w{\gtrdot}_K s$ with $s$ a variable; first, positive formulae appears only from initial conditions, and clearly, it does not make much sense to consider positive formulae with $s$ a variable; second, such formulae add some technical difficulties that make harder the presentation of our results.

Hence, we deal with formulae of the form $\Phi = \exists x_1 \ldots \exists x_m \forall f_1 \ldots \forall f_p \varphi$ where $\{x_i \mid i = 1 \ldots m\} \cup \{f_k \mid k = 1 \ldots p\}$ is the set of all variables that appear in $\varphi$, and $\varphi$ is a

quantifier free formula built using the connectives $\wedge$ and $\vee$ and the following literals:

$$X{\lhd}w{\rhd}_K t \mid t{\lhd}w{\rhd}_K t' \mid x = t \mid \top \mid X{\lhd}\not{w}{\rhd}_K s \mid t{\lhd}\not{w}{\rhd}_K t' \mid t{\lhd}\not\in{\rhd}_K x.f \mid x \neq t \mid \bot$$

where $X$ is a fixed second-order variable that ranges over sets of messages, $x$ is a meta-variable that ranges over the set $\mathcal{X}$ of first-order variables, $f$ is a meta-variable that ranges over $\mathcal{B}$, $s$ ranges over extended terms, $t, t'$ range over terms and $w$ is a term transducer that can contain free variables.

Also, in order to study the decidability of a formula $\Phi$ we shall add to TTL two new kinds of formulae, $X, U{\lhd}\not\in{\rhd}_K x$ and $U{\lhd}\not\in{\rhd}_K x$ with $x \in \mathcal{X}$ and $U$ a meta-variable that ranges over sets of terms, and which have the following semantics:

$$[\![X, U{\lhd}\not\in{\rhd}_K x]\!] = \left\{(E, \sigma, pc) \mid \forall A \in \mathrm{mc}(x\sigma)\ (E\sigma \cup U\sigma)\langle\not\in\rangle_K A\right\}$$

$$[\![U{\lhd}\not\in{\rhd}_K x]\!] = \left\{(E, \sigma, pc) \mid \forall A \in \mathrm{mc}(x\sigma)\ U\sigma\langle\not\in\rangle_K A\right\}.$$

Thus, given a formulae $\forall f_1 \ldots \forall f_p\ \varphi$ with $x_1, \cdots, x_n$ as free variables in $\varphi$, a model of this formula is a pair $(E, \sigma)$ consisting of a set $E$ of messages and a ground substitution $\sigma$ over $x_1, \cdots, x_n$.

### 4.1.2 Solved form

In this section we define the solved form formula and we show how one can check whether a formula in solved form has a model.

**Definition 4.1**

*A formula is called in* solved form *if is syntactically equal to* $\top$, $\bot$ *or* $\exists x_1, \cdots, x_n \cdot \varphi$ *and* $\varphi$ *is of the form:*

$$\bigwedge_{i=1}^{n_1} X{\lhd}\varepsilon{\rhd}_K w_i \wedge \bigwedge_{i=1}^{n_2} X{\lhd}\not\in{\rhd}_K w_i' \wedge \bigwedge_{i=1}^{n} [\bigwedge_{j=1}^{m_i} x_i{\lhd}\varepsilon{\rhd}_K t_i^j \wedge \bigwedge_{j=1}^{l_i} x_i{\lhd}\not\in{\rhd}_K u_i^j \wedge \bigwedge_{j=1}^{o_i} x_i \neq v_i^j]$$

*such that:*

- *For any* $i = 1, \cdots, n$, $x_i \in \mathcal{X}$.
- *For any* $i = 1, \cdots, n$, $x_i \notin \bigcup_{j=1}^{m_i} var\,(t_i^j) \cup \bigcup_{j=1}^{l_i} var\,(u_i^j) \cup \bigcup_{j=1}^{o_i} var\,(v_i^j)$.
- *There is an ordering* $x_{i_1}, \cdots, x_{i_n}$ *of* $x_1, \cdots, x_n$ *such that* $\bigcup_{j=1}^{l_{i_k}} var\ (u_{i_k}^j) \cap \{x_{i_{k+1}}, \cdots, x_{i_n}\} = \emptyset$.

*where by* var $(t)$ *we denote the set of variables that appear in the term* $t$.

■

We now show how one can "easily" check whether a formula in solved form has a model. We only consider the third type of solved formulae. So, let $\varphi$ a conjunction as above. We define a particular substitution $\sigma$ and a set of messages $E$ such that $\varphi$ has a model iff it is satisfied by $(E, \sigma)$.

Let $k \in K$ be a fixed key.

Let $F(n)$, for $n \geq 1$, denote $n$ concatenations of $k$, i.e.,

- $F(1) = k$
- $F(n+1) = \mathbf{pair}(k, F(n))$.

Let $|\varphi|_t$ be the cardinality of the set $St(\varphi)$ consisting of the sub-terms occurring in $\varphi$.

We define the substitution $\sigma$ recursively as follows:

- If $n = 1$, i.e., there is only one variable then

$$\sigma(x_{i_1}) = (u^1_{i_1}, (\cdots, (u^{l_{i_1}}_{i_1}, \{F(|\varphi|_t + i_1)\}_k) \cdots)).$$

  In case $l_{i_1} = 0$ this term is understood as $\{F(|\varphi|_t + i_1)\}_k$.

- If $n > 1$ then replace $x_{i_1}$ by $\sigma(x_{i_1})$ in $\varphi$. This yields a new formula $\varphi'$ and the ordering $x_{i_2}, \cdots, x_{i_n}$, and by recursion, a substitution $\sigma'$. Then, let

$$\sigma = [x_{i_1} \mapsto (u^1_{i_1}, (\cdots, (u^{l_{i_1}}_{i_1}, \{F(|\varphi|_t + i_1))\}_k) \cdots))] \oplus \sigma'.$$

Intuitively, $\sigma(x_i)$ is a "pair" combination of all terms that must not be protected in $x_i$ (all $u^j_i, j = 1, l_i$) plus an unique term $\{F(|\varphi|_t + i)\}_k$ which ensures that $\sigma(x_i)$ is different from $\sigma(v^j_i)$ for all $v^j_i \neq x_i$ when $j = 1, o_i$.

**Theorem 4.1**

*Let $\varphi$ be a formula in solved form syntactically different from $\top$ and $\bot$. Let $\sigma$ be the substitution as defined above and $E = \{w'_1\sigma, \ldots, w'_{n_2}\sigma, k\}$. Then, $\varphi$ has a model iff $(E, \sigma)$ satisfies $\varphi$.*

■

**Proof:**

The interesting implication to prove is the following: If $(E, \sigma)$ does not satisfy $\varphi$ then $\varphi$ has no model.

Now, since $\sigma$ has been defined such that $x\sigma$ is not a sub-term of $\varphi$, for any $x$ in $\{x_1, \cdots, x_n\}$, we have:

1. If $u\sigma \langle \not\in \rangle_K t\sigma$ then $u \lessdot \not\in \gtrdot_K t$ and

2. If $u\sigma \neq t\sigma$ then $u \neq t$.

On the other hand, we can prove that if $\sigma$ is not a model of $\varphi$ then $u_i^j\sigma\langle \not\in \rangle_K t_i^q\sigma$, for some $i \in \{1, \cdots, n\}$, $j \in \{1, \cdots, l_i\}$ and $q \in \{1, \cdots, m_i\}$. Therefore, $u_i^j \lessdot \not\in \gtrdot_K t_i^q$, and hence, $\varphi$ has no model.

■

### 4.1.3 Rewriting rules

In this section, we present a set of rewriting rules that transform any formula $\varphi$ as considered in subsection 4.1.1, into a set of solved formulae, such that $\varphi$ is satisfiable if and only if one of the formulae in solved form is satisfiable.

For the rules of the form $\varphi \longrightarrow \psi$, where $\varphi$ is an atomic formula, we tacitly assume a rule $\neg\varphi \longrightarrow \neg\psi$. Obvious rules (as distributivity of $\vee$ (respectively $\wedge$) with respect to $\wedge$ (respectively $\vee$)) are not mentioned explicitly.

We will encounter two sorts of rewriting rules:

- Deterministic rules are of the form $\varphi \to \varphi'$. They transform a given problem into a single problem. A deterministic rule is sound, if $[\![\varphi]\!] = [\![\varphi']\!]$.
- Non-deterministic rules of the form $\varphi \to \varphi_1, \cdots, \varphi_n$. They transform a given problem into a set of problems. A non-deterministic rule is sound, if $[\![\varphi]\!] = \bigcup_{i=1}^{n} [\![\varphi_i]\!]$.

#### Transducer elimination

The rule **T** decreases the length of $w$ in sub-formulae of the form $u\lessdot w\gtrdot_K s$, if $w \neq \varepsilon$ and $u$ is $X$ or any term; it allows to reduce such formulae to the case $w = \varepsilon$.

$$u\lessdot(b,p).w\gtrdot_K s \quad \mapsto \quad u\lessdot\varepsilon\gtrdot_K s \wedge (u\lessdot\varepsilon\gtrdot_K b \vee b_{|_p}\lessdot w\gtrdot_K s) \quad \text{if } u \in \mathcal{T} \cup \{X\} \qquad \textbf{(T)}$$

Table 4.1: Transducer elimination

#### Preliminary rules

The following rules are useful to eliminate the universal quantifiers and variables of $\mathcal{BX}$. We distribute universal quantification over conjunctions and for disjunctions we use the formulae introduced in the section 4.1.1.

$$\begin{array}{lcll}
\forall f(\phi \wedge \psi) & \mapsto & \forall f\phi \wedge \forall f\psi & \textbf{(P1)} \\
\forall f(\phi \vee \psi) & \mapsto & (\forall f\phi) \vee \psi \text{ if } f \notin var\,(\psi) & \textbf{(P2)} \\
\forall f(X{\prec}\not{\in}{\succ}_K x.f \vee \bigvee_{t\in U} t{\prec}\not{\in}{\succ}_K x.f) & \mapsto & X, U{\lhd}\not{\in}{\rhd}_K\, x & \textbf{(P3.1)} \\
\forall f(\bigvee_{t\in U} t{\prec}\not{\in}{\succ}_K x.f) & \mapsto & U{\lhd}\not{\in}{\rhd}_K\, x & \textbf{(P3.2)}
\end{array}$$

Table 4.2: Preliminary

### Elimination of trivial sub-formulae

The formulae of table 4.3 eliminate trivially satisfied or unsatisfied sub-formulae.

$$\begin{array}{llll}
t = t \mapsto \top & t{\prec}\varepsilon{\succ}_K t \mapsto \bot & \bot \wedge \Phi \mapsto \bot & \top \wedge \Phi \mapsto \Phi \\
x = t \mapsto \bot & x{\prec}\varepsilon{\succ}_K t \mapsto \top & \text{if } x \in \mathcal{X} \cap var\,(t) \wedge t \not\equiv_s x &
\end{array}$$

Table 4.3: Eliminate trivial sub-formulae

### Replacement

The rule **R** is the usual rule for substituting a term for a variable:

$$x = t \wedge \Phi \quad \mapsto \quad \Phi[t/x], \quad \text{if } x \notin var\,(t) \quad \textbf{(R)}$$

Table 4.4: Replacement

### Decomposition rules

The rules of table 4.5 deal with equalities and formulae of the form $t{\prec}\varepsilon{\succ}_K s$ which are transformed using the function $\mathcal{J}$ defined in the proposition 2.10 of the chapter 2.

$$t \lessdot \varepsilon \gtrdot_K s \quad \mapsto \quad \mathcal{J}(t, \varepsilon, s), \quad \text{if } t \notin \mathcal{X} \qquad \textbf{(D1)}$$
$$s = t \quad \mapsto \quad \mu(s, t), \quad \text{if } s, t \notin \mathcal{X} \qquad \textbf{(D2)}$$

Table 4.5: Decompose

**Occur-check**

The main idea behind this rule is that $y \lessdot \not\varepsilon \gtrdot_K t$ induces an ordering on the variables in $t$ and $y$. Indeed, if $x$ is a variables in $t$ then, in any model of this formula, the term assigned to $x$ is a sub-term of the term assigned to $y$.

$$\varphi \mapsto \varphi[y/x] \qquad \textbf{(OC)}$$

if $x$ and $y$ are syntactically different and $x \leq y$ and $y \leq x$, where $\leq$ is the reflexive transitive closure of $<$ with "$x < y$ iff there is a sub-formula of $\varphi$ of the form $y \lessdot \not\varepsilon \gtrdot_K t$ with $x \in \mathit{var}\,(t)$".

Table 4.6: Occur check.

**Simplification rules**

The following rules deal with formulae of the form $U \lhd \varepsilon \rhd_K s$ and $X, U \lhd \varepsilon \rhd_K s$ in the case that $s$ is not a variable (such formulae can be introduced by the elimination of equalities). In order to express this formulae we define for a term $t$ the weakly closed sets of terms $\mathrm{wc}(t)$ as follows:

$$\mathrm{wc}(t) = t \uplus \begin{cases} \mathrm{wc}(t1) \cup \mathrm{wc}(t2) & \text{if } t = (t1, t2) \\ \mathrm{wc}(t') \cup \mathrm{wc}(k) & \text{if } t = \{t'\}_k \\ \{K^{-1}\} & \text{if } t \text{ is atom or variable} \end{cases}$$

### 4.1.4 Rewriting process

The reduction of a formula of TTL$_\exists$ into a set of solved formulae is done in three phases.

1. We define a *preliminary form* and the reduction of a formula in the fragment that interest us, into a preliminary form.
2. We define an *intermediate form* and we show how we use the rules of section 4.1.3 in order to transform any formula in preliminary form into an intermediate form.

If $s \notin \mathcal{X}$ then:

$$U\lhd \not\in \rhd_K s \quad \mapsto \quad \bigwedge_{A\in \mathrm{WC}(s)} \Big[ \bigvee_{t\in A\setminus\mathcal{X}} \bigvee_{u\in U} u\lessdot \not\in \gtrdot_K t \vee \bigvee_{t\in A\cap\mathcal{X}} U\lhd \not\in \rhd_K t \Big] \quad \textbf{(Si1)}$$

$$X, U\lhd \not\in \rhd_K s \quad \mapsto \quad \bigwedge_{A\in \mathrm{WC}(s)} \Big[ \bigvee_{t\in A\setminus\mathcal{X}} \big(X\lessdot \not\in \gtrdot_K t \vee \bigvee_{u\in U} u\lessdot \not\in \gtrdot_K t\big) \vee \bigvee_{t\in A\cap\mathcal{X}} X, U\lhd \not\in \rhd_K t \Big] \quad \textbf{(Si2)}$$

Table 4.7: Simplification

3. For each formula in intermediate form, we show how to reduce its satisfiability to the satisfiability of a set of *saturated formulae* in intermediate form; moreover, for each saturated formula in intermediate form, we can extract a formula in solved form such that a model exists for the formula in intermediate form if and only if the extracted formula in solved form is satisfiable.

### Preliminary form

**Definition 4.2**

*A formula $\Phi$ is called in* preliminary form *if it is of the form $\exists x_1 \ldots \exists x_m \phi$ where $\{x_i \mid i = 1 \ldots m\}$ is the set of all variables that appear in $\phi$, and $\phi$ is a quantifier free formula builded using the connectives $\wedge$ and $\vee$ and the following literals:*

$$X\lessdot\varepsilon\gtrdot_K t \mid t\lessdot\varepsilon\gtrdot_K t' \mid x = t \mid \top \mid X\lessdot \not\in \gtrdot_K t \mid t\lessdot \not\in \gtrdot_K t' \mid X, U\lhd \not\in \rhd_K x \mid U\lhd \not\in \rhd_K x \mid x \neq t \mid \bot$$

*where $X$ is a fixed second-order variable that ranges over sets of messages, $x$ is a meta-variable that ranges over the set $\mathcal{X}$ of first-order variables, $t, t'$ range over terms and $w$ is a term transducer that can contain free variables.*

■

It is easy to see that repeated application as much as possible of *Transducer elimination* and *Preliminary rules* transform any formula as considered in subsection 4.1.1, into an equivalent formula in preliminary form. From now on, it is obvious that as we consider satisfiability of formulae in preliminary form, we can restrict ourselves to conjunctions of literals.

**Intermediate form**

**Definition 4.3**

*A formula is called in* intermediate form *if it is syntactically equal to* $\top$, $\bot$ *or to a conjunction* $\varphi_1 \wedge \varphi_2$ *where* $\varphi_1$ *is of the form:*

$$\bigwedge_{i=1}^{n_1} X \lessdot \varepsilon \gtrdot_K w_i \wedge \bigwedge_{i=1}^{n_2} X \lessdot \not\varepsilon \gtrdot_K w'_i \wedge \bigwedge_{i=1}^{n} [\bigwedge_{j=1}^{m_i} x_i \lessdot \varepsilon \gtrdot_K t_i^j \wedge \bigwedge_{j=1}^{l_i} x_i \lessdot \not\varepsilon \gtrdot_K u_i^j \wedge \bigwedge_{j=1}^{o_i} x_i \neq v_i^j]$$

*and* $\varphi_2$ *is of the form:*

$$\bigwedge_{i=1}^{n} [\bigwedge_{j=1}^{k_i} X, U_i^j \lhd \not\varepsilon \rhd_K x_i \wedge \bigwedge_{j=1}^{h_i} V_i^j \lhd \not\varepsilon \rhd_K x_i]$$

*such that:*

- *For any* $i = 1, \cdots, n$, $x_i \in \mathcal{X}$.
- *For any* $i = 1, \cdots, n$, $x_i \notin \bigcup_{j=1}^{m_i} var\,(t_i^j) \cup \bigcup_{j=1}^{l_i} var\,(u_i^j) \cup \bigcup_{j=1}^{o_i} var\,(v_i^j)$.
- *There is an ordering* $x_{i_1}, \cdots, x_{i_n}$ *of* $x_1, \cdots, x_n$ *such that* $\bigcup_{j=1}^{l_{i_k}} var\,(u_{i_k}^j) \cap \{x_{i_{k+1}}, \cdots, x_{i_n}\} = \emptyset$.

■

A formula in intermediate form defined as above, is called *saturated*, if $\varphi_1 \wedge \varphi_2$ is satisfiable if and only if $\varphi_1$ is satisfiable.

**From preliminary form to intermediate form**

**Theorem 4.2**

*Application of the rules of section 4.1.3 terminates in an intermediate form.*

■

**Proof:**

Let us first briefly mention how each rule contributes in reaching a normal form:

1. Rule **D1** decrease the number of sub-formulae of the form $t \lessdot \varepsilon \gtrdot_K s$ or $t \lessdot \not\varepsilon \gtrdot_K s$ but may introduce equalities and disequalities.
2. Rule **D2** decreases the number of equalities (disequalities) where the two members are not variables.

3. Rules **Si1** and **Si2** decrease the number of sub-formulae of the form $X, U\lhd \not\in \rhd_K s$ and $U\lhd \not\in \rhd_K s$, where $s$ is not a variable, but may introduce new formulae of the form $t\lessdot \not\in \gtrdot_K s$.

4. Rules **Occur-check** and **Replacement** eliminate a variable.

Now, to prove termination we need to introduce interpretation functions which are intended to decrease by applications of the rules:

- $f_1(\varphi)$ is the cardinality of $var\,(\varphi)$.
- $f_2(\varphi)$ is the number of formulae of the form $X, U\lhd \not\in \rhd_K s$ and $U\lhd \not\in \rhd_K s$ with $s$ not a variable.
- $f_3(\varphi)$ is the number of formulae of the form $t\lessdot\varepsilon\gtrdot_K s$ with $t$ not a variable.
- $f_4(\varphi)$ is the number of equalities (disequalities) where both members are not variables.
- $f_5(\varphi)$ is the size of $\varphi$.

Figure 4.1 summarizes the variation of each function by the transformation rules:

| | $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ |
|---|---|---|---|---|---|
| **E** | $=$ | $=$ | $=$ | $=$ | $<$ |
| **D2** | $=$ | $=$ | $=$ | $<$ | |
| **D1** | $=$ | $=$ | $<$ | | |
| **Si1** + **Si2** | $=$ | $<$ | | | |
| **R** + **OC** | $<$ | | | | |

Figure 4.1: Variation of the ranking functions

Thus, if we define $F(\varphi) = (f_1(\varphi), \cdots, f_5(\varphi))$ then $F(\varphi)$ decreases with respect to lexicographic ordering by each rule. Hence, termination of the rules.

It remains now to show that if no rule can be applied then the obtained formula is in intermediate form. This proof is obvious since, for any literal which is not in the intermediate form there exists some rule which can be applied.

■

**Saturating formulae in intermediate form**

We prove that for any formula $\varphi$ in intermediate form, we can construct a set of saturated formulae $\varphi_1; \ldots; \varphi_n$ in intermediate form, such that $[\![\varphi]\!] = \bigcup_{i=1}^{n} [\![\varphi_i]\!]$.

**Saturate rules**

The following rules allow us to saturate a formula in intermediate form.

$$\phi \wedge x \prec \not\rhd_K t \wedge X, U \lhd \not\rhd_K x \;\mapsto\; \phi \wedge x \prec \not\rhd_K t \wedge X, U \lhd \not\rhd_K x \wedge X, U \lhd \not\rhd_K t \quad \textbf{(Sa1)}$$
$$\phi \wedge x \prec \not\rhd_K t \wedge U \lhd \not\rhd_K x \;\mapsto\; \phi \wedge x \prec \not\rhd_K t \wedge U \lhd \not\rhd_K x \wedge U \lhd \not\rhd_K t \quad \textbf{(Sa2)}$$

Table 4.8: Saturate

**Theorem 4.3**

*There exists a strategy to apply the rules of section 4.1.3 and the above ones that terminates in a saturated intermediate form.*

■

**Proof:**

To ensure the termination, we apply the *Saturate rules* **Sa1** and **Sa2** only for the pairs $(x \prec \not\rhd_K t;\ X, U \lhd \not\rhd_K x)$ or $(x \prec \not\rhd_K t;\ V \lhd \not\rhd_K x)$ that are not marked, and after the application of such a rule, we mark the corresponding pair.

On the other hand, any time we apply the **Replacement** or the **Occur-check** rules, we unmark all the pairs of constraints which were marked before. Then, the termination follows from the remark that the number of variables is finite, all the rules but Replacement or Occur-check introduce only subformulae of the formulae we already have, and no rule does not introduce any new variable.

To prove that any formula obtained after the termination of the above algorithm is saturated, we make an induction on the position of variables w.r.t. the order $\leq$. Indeed let $\varphi_1 \wedge \varphi_2$ be such a formula and let $\sigma$ be the substitution and $E$ be the set of messages as defined in the section 4.1.2 corresponding to the formula $\varphi_1$. Then, $(\sigma, E)$ satisfies $\varphi_1 \wedge \varphi_2$.

We prove by induction on the position of the variable $x$ w.r.t. the order $\leq$ that for any constraints $X, U \lhd \not\rhd_K x \in \varphi_2$, and $V \prec \not\rhd_K x \in \varphi_2$ and for any $A \in \mathrm{mc}(x\sigma)$ it holds $(E \cup U)\sigma\langle\not\rangle_K A$, and $V\sigma\langle\not\rangle_K A$.

The idea of the proof, is that when the algorithm terminates, we already applied the Saturate rules to any of the pairs $(x \prec \not\rhd_K t;\ X, U \lhd \not\rhd_K x)$ and $(x \prec \not\rhd_K t;\ V \lhd \not\rhd_K x)$, and such rules introduce only formulae of the kind $\phi_1$, or of the form $X, U \lhd \not\rhd_K z$ or $V \lhd \not\rhd_K z$ with $z \leq x$.

■

## 4.2 Complexity

In this section we prove that the problem of deciding the existence of a model for a $\text{TTL}_\exists$ formula is $NP$-complete. We define the size of a formula $\varphi$ to be the size of its DAG representation. Roughly speaking, it is the cardinality of the set of its sub-formulae and sub-terms. We denote the size of $\varphi$ by $|\varphi|$.

### 4.2.1 $\text{TTL}_\exists$ satisfiability problem is in $NP$

Let $\varphi$ be a conjunction of literals in preliminary form. And let $|\varphi|_t$ be the cardinality of the set $St(\varphi)$ consisting of the sub-terms of $\varphi$. Clearly, we have $|\varphi|_t \leq |\varphi|$. Then, we show that if $\varphi$ is satisfiable then it has a model $(E, \sigma)$ such that the size of $\sigma(x)$ is polynomially bounded by $|\varphi|_t$, for each variable $x$. To do so, we first introduce a special kind of substitutions.

**Definition 4.4**

*Let $Tr$ be a set of terms, and let $\rho$ be a substitution defined on the set of variables of $Tr$. Then $\rho$ is called a $Tr$-substitution, if:*

- *for any variable $x \in dom(\rho)$, there is a term $v_x \in St(Tr)$ which is not a variable (i.e. $v_x \notin \mathcal{X}$), and such that $\rho(x) = \rho(v_x)$;*
- *$\rho$ is idempotent.*

■

Then, we will use the following proposition, which is proved as theorem 1 in [RT01].

**Proposition 4.4**

*Let $Tr$ a set of terms and let $\rho$ a $Tr$-substitution. Then for any variable $x$, $|\rho(x)| \leq |St(Tr)|$.*

■

Let us assume that $\varphi$ is satisfiable. Then, there exists a formula $\psi$ in solved form such that $\psi$ is obtained from $\varphi$ using the rewriting rules from section 4.1.3, and such that $\psi$ is satisfiable. Let $(E, \sigma_0)$ be the model of $\psi$ defined in section 4.1.2.

Let $\{x_1, \cdots, x_n\}$ be the variables of $\varphi$. For each $x_i$, we introduce $|\varphi|_t$ new variables $z^i_1, \cdots, z^i_{|\varphi|_t}$ and consider the term $t_{x_i}$ defined as follows:

$$t_{x_i} = (z^i_1, \cdots (z^i_{|\varphi|_t}, F(|\varphi|_t) + i)) \cdots).$$

Now, the following facts hold:

1. there exists a substitution $\sigma$ such that $\sigma_0 \subset \sigma$ and $\sigma$ is a $Tr$-substitution, where $Tr = St(\psi) \cup \{t \mid t \leq t_{x_i}, i = 1, \cdots, n\}$

   This is an immediate consequence of the construction of $\sigma_0$ and the definition of $t_x$ terms.

2. $St(\psi) \subseteq St(\varphi)$

   This can be proved by induction on the rewriting length of $\varphi$ into $\psi$ and using that, if $\varphi_1, \varphi_2$ are two TTL$_\exists$ formulae such that $\varphi_1 \rightarrow \varphi_2$ by applying some rewriting rule defined in section 4.1.3 then we have $St(\varphi_2) \subseteq (St(\varphi_1))$.

From the first fact, using the proposition 4.4 we obtain that the size of $\sigma_0$ is polynomial in the size of $St(\psi) \cup \{t_{x_i} \mid i = 1, \cdots, n\}$. From the second fact, we obtain that $|\psi|_t$ is polynomial in $|\varphi|_t$.

Moreover, since the size of the set $\{t \mid t \leq t_{x_i}, i = 1, \cdots, n\}$ is also polynomial in the size of $St(\varphi)$ we conclude that the size of $\sigma_0$ is polynomial in the size of $St(\varphi)$.

Verifying that $(E, \sigma_0)$ is a solution of $\varphi$ has a polynomial time complexity too. Each atomic equality $x = t$ amounts to a syntactic equality of messages $x\sigma_0$ and $t\sigma_0$. Each atomic formula $X \lessdot \varepsilon \gtrdot_K t$ (or $x \lessdot \varepsilon \gtrdot_K t$) amounts to check if the intersection $I_d(E) \cap \{t\sigma_0\}$ (respective $I_d(x\sigma_0) \cap \{t\sigma_0\}$) is empty, where $I_d(E), I_d(x\sigma_0)$ are bounded by the size of messages in $E$ and $\sigma_0$.

Hence, the following theorem holds.

**Proposition 4.5**

TTL$_\exists$ *satisfiability problem is in* $NP$.

■

### 4.2.2 $NP$-hardness

First, we prove the $NP$-hardness, using a polynomial reduction of 3-SAT to TTL$_\exists$ satisfiability problem. Let $x_1, \ldots, x_n$ be Boolean variables, and let $f = \bigwedge_{i=1}^{m} l_i^1 \vee l_i^2 \vee l_i^3$ a formula in 3-conjunctive normal form, where $l_i^j \in \{x_1, \ldots, x_n, \neg x_1, \ldots \neg x_n\}$. It is well known that deciding the existence of a model for such a formula is $NP$-complete. We shall construct a TTL$_\exists$-formula $\varphi_f$ such that $\varphi_f$ is satisfiable if and only if $f$ is satisfiable.

Let $c, T, F \in \mathcal{A}$ be three distinct atoms and let $k_1, k_2 \in K$ be two distinct keys in $K$. For any literal $l$, we denote

$$t(l) = \begin{cases} \{c, x_j\}_{k_1} & \text{if } l = x_j \\ \{c, x_j\}_{k_2} & \text{if } l = \neg x_j \end{cases}$$

and for any clause $C = l^1 \vee l^2 \vee l^3$ we denote $t(C) = ((t(l^1), t(l^2)), t(l^3))$. Then, for any clause $C_i = l_i^1 \vee l_i^2 \vee l_i^3$, we consider the formula

$$\varphi_{C_i} = \neg(t(C_i), \{t(C_i), T\}_{k_1}) \langle (\{x, T\}_{k_1}, 00).(\{y, F\}_{k_2}, 00) \rangle_K c.$$

and finally, for $f = \bigwedge_{i=1}^{m} C_i$, we take $\varphi_f = \bigwedge_{i=1}^{m} \varphi_{C_i}$.

Now we prove that models of $f$ coincide with models of $\varphi_f$. More precisely, given a substitution $\sigma : \{x_1, \ldots, x_n\} \to \{T, F\}$, let $\hat{\sigma}$ denote the boolean function such that $\hat{\sigma}(x_i) = \top$, if $\sigma(x_i) = T$ and $\hat{\sigma}(x_i) = \bot$, otherwise. Then, we prove

$$\sigma \models \varphi_f \text{ iff } \hat{\sigma} \models f.$$

For any variable $x_i$, if a substitution $\sigma$ satisfies $\neg t(x_i)\langle(\{x, T\}_{k_1}, 00)\rangle_K c$, then $\sigma(x_i) = T$ and similarly, if $\sigma$ satisfies $\neg t(\neg x_i)\langle(\{y, F\}_{k_2}, 00)\rangle_K c$, then $\sigma(x_i) = F$. Moreover, the formulae $\neg t(\neg x_i)\langle(\{x, T\}_{k_1}, 00)\rangle_K c$ and $\neg t(x_i)\langle(\{y, F\}_{k_2}, 00)\rangle_K c$ are not satisfiable. Therefore, $\sigma$ is a model of

$$\neg t(C_i)\langle(\{x, T\}_{k_1}, 00)\rangle_K c \vee \neg t(C_i)\langle(\{y, F\}_{k_2}, 00)\rangle_K c$$

iff $\hat{\sigma}$ is a model of $C_i = l_i^1 \vee l_i^2 \vee l_i^3$. And hence,

$$\sigma \models \varphi_f \text{ iff } \hat{\sigma} \models \varphi.$$

Now, since if $\varphi$ has a model, then it has a model that maps each variable $x_i$ into $\{T, F\}$, and since $\varphi_f$ is polynomial in the size of $f$, we have the following:

**Proposition 4.6**

TTL$_\exists$ *satisfiability problem is NP-hard.*

■

Hence, from the propositions 4.5 and 4.6 we obtain that the problem of deciding the existence of a model for a TTL$_\exists$ formula is $NP$-complete.

## 4.3 Undecidability of TTL

In this section, we study the decidability of the satisfiability of a general TTL formula. In section 4.1 we have proved the decidability of this problem for existential formulae TTL$_\exists$. Here, we prove that if we allow both existential and universal quantifiers, then the satisfiability problem becomes undecidable. Indeed, we can show that Post's correspondence problem is reducible to the decision problem in our logic.

**Theorem 4.7**

*The satisfiability problem for the* TTL *logic is undecidable.*

■

**Proof:**

We show that the Post's correspondence problem is reducible to the decision problem for the TTL logic.

The proof is inspired from [Ven87], where it is shown the undecidability of a certain fragment in the theory of free term algebras.

Let $P = \{(p_i, q_i) \mid i = 1, \ldots, n\}$ be an instance of Post's correspondence problem, where $p_i, q_i \in D^*$, with $D = \{d_1, \ldots, d_k\}$. We use $d_1, \ldots, d_k$ as constants, and also

let $c$ be another particular constant. One is asked to determine whether there exists a non-empty sequence of indices $i_1, \dots i_m$ such that $p_{i_1} \dots p_{i_m} = q_{i_1} \dots q_{i_m}$.

The idea is to construct a TTL formula such that if this formula is satisfiable, then from its solution a solution to Post's problem can be recovered.

For $i = 1, \dots k$, we denote, $g_i(x) = \mathbf{pair}(d_i, x)$. The monadic functions $g_i$ represent the alphabet as follows: the string $d_{i_1} \dots d_{i_j}$ is represented by the term $g_{i_1}(\dots(g_{i_j}(c))\dots)$. Moreover, for any string $d = d_{i_1} \dots d_{i_j}$ we denote by $d(x)$ the term $g_{i_1}(\dots(g_{i_j}(x))\dots)$. In particular, we use this notation to encode the strings $p_i, q_i$ occurring in the problem definition.

We denote $f(x_1, x_2, x_3) = \mathbf{pair}(\mathbf{pair}(x_1, x_2), x_3)$. The use of this function will be clear later.

Suppose that $P$ has a solution $i_1, \dots i_m$, that is $m > 0$ and $1 \leq i_j \leq n$ for each $j$ and $p_{i_1} \dots p_{i_m} = q_{i_1} \dots q_{i_m}$. For each $j = 1, \dots, m+1$, let $r_j = p_{i_j} \dots p_{i_m}$ and $s_j = q_{i_j} \dots q_{i_m}$. Thus $r_1 = s_1$ and $r_{m+1} = s_{m+1} = \varepsilon$. Then the formula $\Phi_P$ given below is satisfiable, with the following value for $x$:

$$x = f(r_1, s_1, f(r_2, s_2, f(\dots f(r_{m+1}, s_{m+1}, c)\dots))).$$

The formula $\Phi_P$ is

$\exists x, x_1, x_2 \forall y_0, \dots, y_6$

$$[Is_{fgc}(x) \wedge \bigwedge_{i=0}^{6} x \prec \not\rhd_\emptyset y_i] \tag{4.1}$$

$$\wedge [x = f(x_1, x_1, x_2) \wedge x_1 \neq c] \tag{4.2}$$

$$\wedge [y_0 \neq f(y_1, y_2, y_3) \vee [y_1 \neq f(y_4, y_5, y_6) \wedge y_2 \neq f(y_4, y_5, y_6) \wedge \bigwedge_{i=1}^{k} y_3 \neq g_i(y_4)]] \tag{4.3}$$

$$\wedge [y_0 \neq f(y_1, y_2, c) \vee y_1 = y_2 = c] \tag{4.4}$$

$$\wedge [y_0 \neq f(y_1, y_2, f(y_3, y_4, y_5)) \vee \bigvee_{i=1}^{n} [y_1 = p_i(y_3) \wedge y_2 = q_i(y_4)]] \tag{4.5}$$

where

$$Is_{fgc}(x) ::= [Is_{gc}(x) \vee P_3(x)] \wedge \forall y [x \prec\!\varepsilon\!\rhd_\emptyset y \vee Is_{gc}(y) \vee P_3(y)]$$

$$P_3(x) ::= \exists x_1, x_2, x_3 [x = f(x_1, x_2, x_3)]$$

$$Is_{gc}(x) ::= M(x) \wedge \forall y [x \prec\!\varepsilon\!\rhd_\emptyset y \vee M(y)]$$

$$M(x) ::= x = c \vee \exists y[\bigvee_{i=1}^{k} x = g_i(y)]$$

The meaning of each sub-formula is given below:

1. $Is_{gc}(t)$ means that $t$ is either $c$ or has the form $g_{i_1}(\ldots(g_{i_p}(c)))$.
2. $Is_{fgc}(t)$ means that $t$ is either $c$ or has the form $g_{i_1}(\ldots(g_{i_p}(c)))$ or the form $f(t_1, t_2, t_3)$ and for the last case, the same property holds for $t_1$, $t_2$ and $t_3$ too.
3. the sub-formula (4.1) ensures that $y_0, \ldots, y_6$ are sub-terms of $x$. $x$ and also any sub-term of $x$ are built using only the constant $c$ and the "function symbols" $g_i$ and $f$; moreover, any sub-term that has a $g_i$ as the outermost function symbol, has the form $g_{i_1}(\ldots(g_{i_p}(c)))$.
4. the sub-formula (4.2) forces $r_1 = s_1$.
5. the sub-formula (4.3) ensures for any sub-term $f(y_1, y_2, y_3)$ of $x$ that $y_1$ and $y_2$ must be $c$ or have one of the $g_i$ as the outermost "function symbol", and $y_3$ must be $c$ or have $f$ as the outermost symbol.
6. the sub-formula (4.4) forces $r_{m+1} = s_{m+1} = \varepsilon$.
7. the sub-formula (4.5) ensures for each $j$ that there is $i$ such that $r_j = p_i r_{j+1}$ and $s_j = q_i s_{j+1}$.

Intuitively, the formula $\Phi_P$ encodes terms of the form depicted in figure 4.2. At each level, $r_i$ and $s_i$ represent strings. The whole term encode a solution of the Post correspondence problem.

■

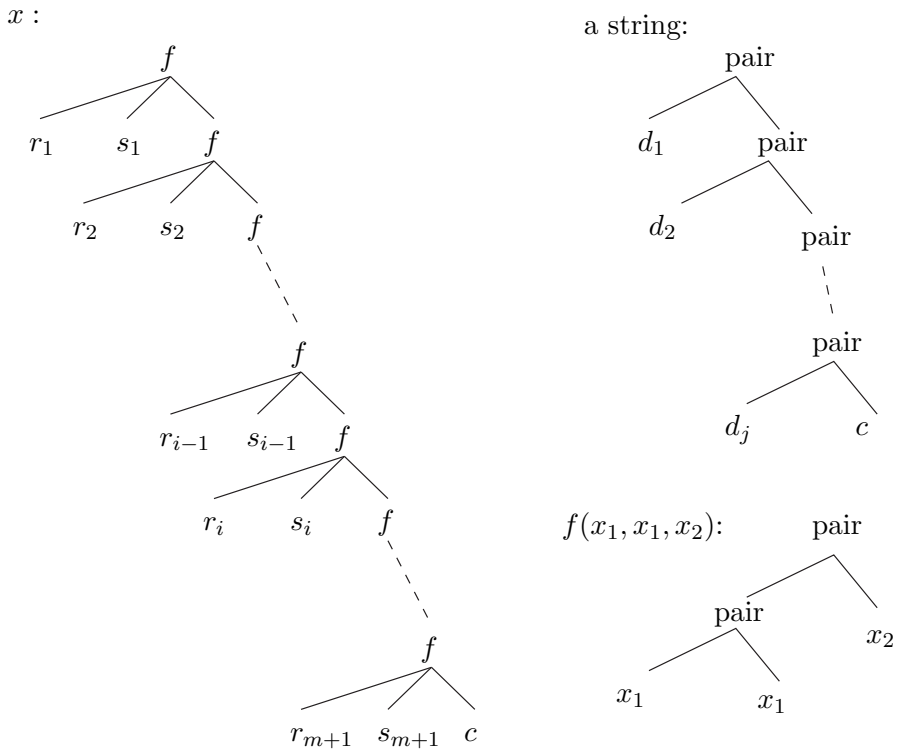


Figure 4.2: Encoding of the Post correspondence problem

# Chapter 5

# Timed TTL

Most of the verification methods and decidability results for cryptographic protocols consider time-independent protocols. The timestamps are replaced by nonce and, as consequence, temporal properties of timestamps are not taken into account. The existing work on timed cryptographic protocols uses theorem-provers [BP98, Coh00, ES00] or finite-state model-checking [Low96]. While the first needs human help, the second relies on typing assumptions and assumption on the time window to bound the search space. Recently, a verification approach based on data structures for symbolically representing sets of configurations of unbounded time sensitive cryptographic protocols has been proposed in [DP04].

In this chapter we extend the TTL logic to Timed TTL, short TTTL, in order to apply our verification method for bounded time sensitive cryptographic protocols. The TTTL logic combines constraints on the knowledge of the intruder with time constraints on clocks and time variables.

First, we give a formal definition for the TTTL and we show how the semantics of TTL is extended to TTTL. Timed cryptographic protocols have been introduced in section 1.5. Then, we extend the weakest precondition calculus for TTL logic and untimed cryptographic protocols to TTTL logic and timed cryptographic protocols.

Finally, we show that our results concerning the decidability of the TTL logic are also valid for the TTTL logic.

Our timed model is clearly inspired by timed automata and our verification method influenced by the work on symbolic verification of timed automata and temporal logics for real-time systems (e.g.[AD94, HNSY92, AFH91, BL95]).

## 5.1 Time-sensitive Logic

We denote by $\Psi$ the time constraints (see the definition 1.1) and by $\Phi$ the formulae of the TTL, where the terms involved range over $\mathcal{T}(\mathcal{X} \cup \mathcal{Y} \cup \mathcal{C}, \mathcal{F})$. The extended terms are terms of $\mathcal{T}_c$, except that we allow function symbols in $\mathcal{B}$ to occur applied to variables of

$\mathcal{X}$. Then, the TTTL logic is defined as follows:

$$\Phi ::= X\lhd w\rhd_K S \mid x\lhd w\rhd_K S \mid t\lhd \varepsilon\rhd_K x.f \mid x = t \mid pc_p^i = \ell \mid \top \mid \Phi \wedge \Phi \mid \forall x\Phi \mid \exists f\Phi \mid \neg\Phi \quad \text{TTL formulae}$$

$$\Psi ::= \top \mid \sum_{i=1}^{n} a_i c_i + \sum_{j=1}^{m} b_j T_j \bowtie d \mid \Psi_1 \wedge \Psi_2 \mid \Psi_1 \vee \Psi_2 \quad \text{time constraints}$$

$$\Gamma ::= \Phi \mid \Psi \mid \Gamma \vee \Gamma \mid \Gamma \wedge \Gamma \quad \text{TTTL formulae}$$

The formulae are interpreted over the restricted set of configurations

$$\mathcal{C}onf_0 = \{(E, \sigma, \nu, pc) \mid (E, \sigma, \nu, pc) \in \mathcal{C}onf \wedge \mathcal{K} \setminus K^{-1} \subseteq E\}.$$

**Definition 5.1 (TTTL semantics)**

*The semantics of* TTTL *is defined inductively on the structure of formula by the following clauses:*

- $[\![\Phi]\!]_t = \{(E, \sigma, \nu, pc) \mid (E, \sigma, pc) \in [\![\Phi]\!]\}$
- $[\![\Psi]\!]_t = \{(E, \sigma, \nu, pc) \mid [\![\Psi]\!]_{\nu,\sigma} = 1\}$
- $[\![\Gamma_1 \wedge \Gamma_2]\!]_t = [\![\Gamma_1]\!]_t \cap [\![\Gamma_2]\!]_t$
- $[\![\Gamma_1 \vee \Gamma_2]\!]_t = [\![\Gamma_1]\!]_t \cup [\![\Gamma_2]\!]_t$

■

## 5.2 Weakest liberal precondition

We define the TTTL$_\forall$ fragment of TTTL by restricting the TTL formulae to TTL$_\forall$ formulae. The purpose of this section is to show that the weakest liberal precondition $wlp(\alpha, \Gamma)$ is effectively expressible in TTTL$_\forall$, when $\Gamma$ is a TTTL$_\forall$ formula and $\alpha$ is a timed cryptographic protocol action, i.e. timed input action, timed output action or time passing action. We recall that omitting the negation for time constraints is not essential as any negation of a time constraint can be express in terms of positive form.

For the sake of clarity, we denote the formulae of TTTL logic by $\Phi$ when they are formulae corresponding of the TTL, by $\Psi$ when they are time constraints and by $\Gamma$ when we do not distinguish between them.

First, it is easy to see that $wlp(\alpha, \Phi) = \Phi$, if $\alpha$ is a time passing action and $\Phi$ is a TTL$_\forall$ formula.

Second, for any output action $\alpha = [pc_p^i = \ell_1 \wedge g] \rightarrow reset(\mathcal{R}); add(X, t_c); pc_p^i := \ell_2$ and any time constraint $\Psi$, we have $wlp(\alpha, \Psi) = pc_p^i = \ell_1 \wedge g \Rightarrow \Psi[R]$. Respectively for

any input action $\alpha = [pc_p^i = \ell_1 \wedge in(X, t(\tilde{x})) \wedge g] \rightarrow reset(\mathcal{R}); pc_p^i := \ell_2$, and any time constraint $\Psi$, we have $wlp(\alpha, \Psi) = pc_p^i = \ell_1 \wedge g \Rightarrow (\mathtt{Secret}(t) \vee \Psi[R])$.

Let us now concentrate our attention on time passing action and time constraints formulae.

### 5.2.1 Time passing and time constraints

We show that the weakest liberal precondition of $[\![\Psi]\!]$, where $\Psi$ is a time constraint, can be described by a TTTL$_\forall$ formula. Here we consider the action $\xrightarrow{\tau}$, i.e. time passing. The case of input and output actions has been seen above.

We need to define three kinds of normal forms for time constraints. Let $\Psi$ be the atomic time constraint $\sum_{i=1}^{n} a_i c_i + \sum_{j=1}^{m} b_j T_j \bowtie d$. We denote by $\mathscr{C}(\Psi)$ the sum of the coefficients of clocks, i.e. $\sum_{i=1}^{n} a_i$. Then, an atomic time constraint $\Psi \equiv \sum_{i=1}^{n} a_i c_i + \sum_{j=1}^{m} b_j T_j \bowtie d$ is in

- *positive normal form* (PNF for short), if $\mathscr{C}(\Psi) > 0$;
- *negative normal form* (NNF for short), if $\mathscr{C}(\Psi) < 0$;
- *0-normal form*, if $\mathscr{C}(\Psi) = 0$.

Clearly any time constraint can be put in the form of a conjunction of disjunctions of the form $\Psi_1 \vee \Psi_2 \vee \Psi_3$, where $\Psi_1$ is a disjunction of formulae in PNF, $\Psi_2$ is a disjunction of formulae in NNF and $\Psi_3$ is a disjunction of formulae in 0-NF.

Thus, let us consider a time constraint of the form $\Psi_1 \vee \Psi_2 \vee \Psi_3$ as above. Then, $wlp(\xrightarrow{\tau}, \Psi_1 \vee \Psi_2 \vee \Psi_3)$ can be described by the formula $\forall \delta \geq 0 \cdot \Psi_1 + \delta \vee \Psi_2 + \delta \vee \Psi_3 + \delta$. We have then to show that we can eliminate the quantification on $\delta$ and obtain a time constraint.

First, notice that $\Psi_3 + \delta$ is logically equivalent to $\Psi_3$, since it is in 0-NF. Therefore, we can rewrite our formula as $\forall \delta \geq 0 \cdot (\Psi_1 + \delta \vee \Psi_2 + \delta) \vee \Psi_3$ and focus on how to transform $\forall \delta \geq 0 \cdot (\Psi_1 + \delta \vee \Psi_2 + \delta)$ into an equivalent time constraint.

To do that, we replace the universal quantification by an existential one, using a double negation:

$$\forall \delta \geq 0 \cdot (\Psi_1 + \delta \vee \Psi_2 + \delta) \;\equiv\; \neg(\exists \delta \geq 0 \cdot (\neg(\Psi_1 + \delta) \wedge \neg(\Psi_2 + \delta)))$$

and then, we eliminate the existential quantification using the Fourier-Motzkin variable elimination procedure.

Let us start with a formula of the form: $\exists \delta \geq 0 \cdot (\Theta_1 + \delta \wedge \Theta_2 + \delta)$, where $\Theta_1$ is a conjunction of formulae in PNF corresponding to $\neg\Psi_2$ and $\Theta_2$ is a conjunction of formulae in NNF corresponding to $\neg\Psi_1$.

We explain the main idea by considering the simple case where $\Theta_1 = \theta_1$ and $\Theta_2 = \theta_2$ are atomic time constraints.

**The simple case**

Consider a PNF time constraint $\theta_1 \equiv \sum_{i=1}^{n} a_i c_i + \sum_{j=1}^{m} b_j T_j \bowtie_1 d$ and a NNF $\theta_2 \equiv \sum_{i=1}^{n} a'_i c_i + \sum_{j=1}^{m} b'_j T_j \bowtie_2 d'$. Then, we have:

$$\theta_1 + \delta \equiv \sum_{i=1}^{n} a_i c_i + \sum_{j=1}^{m} b_j T_j + \delta \sum_{i=1}^{n} a_i \bowtie_1 d$$
$$\theta_2 + \delta \equiv \sum_{i=1}^{n} a'_i c_i + \sum_{j=1}^{m} b'_j T_j + \delta \sum_{i=1}^{n} a'_i \bowtie_2 d'$$

By multiplying with $\mathscr{C}(\theta_1)$ and $|\mathscr{C}(\theta_2)|$ we have:

$$\theta_1 + \delta \equiv \sum_{i=1}^{n} |\mathscr{C}(\theta_2)| a_i c_i + \sum_{j=1}^{m} |\mathscr{C}(\theta_2)| b_j T_j + \delta |\mathscr{C}(\theta_2)| \sum_{i=1}^{n} a_i \bowtie_1 |\mathscr{C}(\theta_2)| d$$
$$\theta_2 + \delta \equiv \sum_{i=1}^{n} \mathscr{C}(\theta_1) a'_i c_i + \mathscr{C}(\theta_1) \sum_{j=1}^{m} b'_j T_j + \delta \mathscr{C}(\theta_1) \sum_{i=1}^{n} a'_i \bowtie_2 \mathscr{C}(\theta_1) d'$$

Adding the right-hands of the equivalences yields the time constraint:

$$\sum_{i=1}^{n} a''_i c_i + \sum_{j=1}^{m} b''_j T_j \bowtie' |\mathscr{C}(\theta_2)| d + \mathscr{C}(\theta_1) d'$$

with $a''_i = |\mathscr{C}(\theta_2)| a_i + \mathscr{C}(\theta_1) a'_i$, $b''_j = |\mathscr{C}(\theta_2)| b_i + \mathscr{C}(\theta_1) b'_i$ and if $\bowtie_1 \equiv \bowtie_2$ then $\bowtie' \equiv \bowtie_1$ else $\bowtie' \equiv <$.

Let us denote this formula by $\Delta(\theta_1, \theta_2)$. Notice that $\Delta(\theta_1, \theta_2)$ is independent of $\delta$. One can prove that $\exists \delta \geq 0 \cdot (\theta_1 + \delta \wedge \theta_2 + \delta)$ is equivalent to the time constraint $\theta_1 \wedge \Delta(\theta_1, \theta_2)$. The conjunct $\theta_1$ has to be kept as we are interested in the predecessors, thus the upper bound on the clocks must be satisfied as time only increases.

**The general case**

Let us now return to the general case, where $\Theta_1$ and $\Theta_2$ are arbitrary conjunctions of formulae in PNF, respectively, NNF. To handle this case we generalize $\Delta$ to sets (conjunctions of formulae as follows):

- $\Delta(\emptyset, \Theta) = \top$.
- $\Delta(\Theta, \emptyset) = \Theta$.
- $\Delta(\Theta_1, \Theta_2) = \bigwedge_{\theta_1 \in \Theta_1, \theta_2 \in \Theta_2} \Delta(\theta_1, \theta_2)$

Then we can prove that $\exists \delta \geq 0 \cdot (\Theta_1 + \delta \wedge \Theta_2 + \delta)$ is equivalent to $\Delta(\Theta_1, \Theta_2) \wedge \Theta_1$.

Summarizing, we can transform $\forall \delta \geq 0 \cdot \Psi_1 + \delta \vee \Psi_2 + \delta \vee \Psi_3 + \delta$ into the equivalent time constraint $\neg \Delta(\neg \Psi_2, \neg \Psi_1) \vee \Psi_2 \vee \Psi_3$. Hence, if we define

$$\text{WLP}(\xrightarrow{\tau}, \Psi_1 \wedge \Psi_2 \wedge \Psi_3) \stackrel{def}{=} \neg \Delta(\neg \Psi_2, \neg \Psi_1) \vee \Psi_2 \vee \Psi_3,$$

we obtain the following result:

**Proposition 5.1**

*For any time constraint* $\Psi$*,*

$$wlp(\xrightarrow{\tau}, [\![\Psi]\!]) = [\![\text{WLP}(\xrightarrow{\tau}, \Psi)]\!].$$

■

### 5.2.2 Output action and atomic TTL formulae

Throughout this section let $\alpha = [pc_p^i = \ell_1 \wedge g] \rightarrow reset(\mathcal{R}); add(X,t); pc_p^i := \ell_2$ and let $\tilde{c}$ be all the clocks that occur in $t$ and do not occur in $\mathcal{R}$. We show that we can express $wlp(\alpha,\varphi)$, for any atomic TTL formula $\varphi$. The core point here is how we deal with the clock occurrences in the sent message. Since the values of clocks change with time, we have to freeze these values in the message added to the intruder knowledge; we do this by replacing in $t$, all occurrences of clocks that are not reset $\tilde{c}$ with fresh time variables $\tilde{T}_c$ and by introducing the constraints $\tilde{T}_c = \tilde{c}$. For details, see also the example presented in the section 5.2.5.

Following the previous remarks, we define $\widehat{wlp}(\alpha,\varphi)$ in a similar way to the untimed case:

$$\begin{array}{lll}
\widehat{wlp}(\alpha,\varphi) & \stackrel{def}{=} & pc_p^i = \ell_1 \wedge g \Rightarrow (\tilde{T}_c = \tilde{c} \Rightarrow \varphi \wedge \mathcal{J}(t[0/\mathcal{R}, \tilde{T}_c/\tilde{c}], w, S)) \\
 & & \qquad \text{if } \varphi \in \{X \lessdot w \gtrdot_K S, (X,x) \lessdot w \gtrdot_K S\} \\
\widehat{wlp}(\alpha,\varphi) & \stackrel{def}{=} & pc_p^i = \ell_1 \wedge g \Rightarrow \varphi \\
 & & \qquad \text{if } \varphi \in \{x \neq t', x = t', t \lessdot \varepsilon \gtrdot_K x.f, \top, \bot, \\
 & & \qquad\quad pc_q^j = \ell''\} \text{ and } q \neq p \text{ or } j \neq i \\
\widehat{wlp}(\alpha, pc_p^i = \ell'') & \stackrel{def}{=} & pc_p^i = \ell_1 \wedge g \Rightarrow \ell_2 = \ell''
\end{array}$$

Then, the following theorem can be proved in the same way as the theorem 3.1.

**Theorem 5.2**

*For any output action* $\alpha$ *and atomic* TTL *formula* $\varphi$*,*

$$wlp(\alpha, [\![\varphi]\!]) = [\![\widehat{wlp}(\alpha,\varphi)]\!].$$

■

### 5.2.3 Input action and atomic TTL formulae

Throughout this section let $\alpha = [pc_p^i = \ell_1 \wedge in(X, t(\tilde{x})) \wedge g] \rightarrow reset(\mathcal{R}); pc_p^i := \ell_2$ Then we define $\widehat{wlp}(\alpha,\varphi)$ in the same way as for the untimed input action but adding the $g$ guard.

$$\begin{array}{lll}
\widehat{wlp}(\alpha,(X,x)\lessdot w\gtrdot_K S) & \stackrel{def}{=} & pc_p^i = \ell_1 \wedge g \Rightarrow (\mathtt{Secret}(t) \vee \mathcal{K}(t,x,w,S)) \\
 & & \qquad \text{if } x \in \tilde{x} \\
\widehat{wlp}(\alpha,\varphi) & \stackrel{def}{=} & pc_p^i = \ell_1 \wedge g \Rightarrow (\mathtt{Secret}(t) \vee \varphi) \\
 & & \qquad \text{if } \varphi \in \{X\lessdot w\gtrdot_K S, (X,y)\lessdot w\gtrdot_K S, t\lessdot \varepsilon\gtrdot_K x.f, \\
 & & \qquad\quad x \neq t', x = t', pc_q^j = \ell'', \top, \bot\} \\
 & & \qquad\quad \text{and } y \notin \tilde{x} \text{ and } q \neq p \text{ or } j \neq i \\
\widehat{wlp}(\alpha, pc_p^i = \ell'') & \stackrel{def}{=} & pc_p^i = \ell_1 \wedge g \Rightarrow \ell_2 = \ell''
\end{array}$$

We can prove in the same way as the theorem 3.6 the following theorem:

**Theorem 5.3**

*For any input action $\alpha$ and term formula $\varphi$,*

$$wlp(\alpha, [\![\varphi]\!]) = [\![\forall \tilde{x} \cdot \widehat{wlp}(\alpha,\varphi)]\!].$$

■

### 5.2.4 Collecting the results together

It is easy to see that for any formula $\varphi \in \text{TTTL}_\forall$ and any action $\alpha$, $\widehat{wlp}(\alpha,\varphi) \in \text{TTTL}_\forall$.

**Definition 5.2**

*We define the formula* $\text{WLP}(\alpha,\varphi)$ *as follows:*

$$\text{WLP}(\alpha,\varphi) = \begin{cases} \widehat{wlp}(\alpha,\varphi) & \text{if } \alpha = [pc_p^i = \ell_1 \wedge g] \rightarrow reset(\mathcal{R}); add(X,t); pc_p^i := \ell_2 \\ \forall \tilde{x} \cdot \widehat{wlp}(\alpha,\varphi) & \text{if } [pc_p^i = \ell_1 \wedge in(X,t(\tilde{x})) \wedge g] \rightarrow reset(\mathcal{R}); pc_p^i := \ell_2 \end{cases}$$

■

Then, we have the following theorem:

**Theorem 5.4**

*Let $\alpha$ be any action and $\varphi$ any formula in* $\text{TTTL}_\forall$. *Then,*

$$wlp(\alpha, [\![\varphi]\!]) = [\![\text{WLP}(\alpha,\varphi)]\!].$$

■

### 5.2.5 Computation of *wlp* for a sequence of actions

In this section, we give an example that shows how we compute weakest precondition with respect to a sequence of actions. Let

$$\alpha_0 = [pc_a^1 = \ell_0] \rightarrow add(X, \{c\}_k); pc_a^1 := \ell_1,$$

$$\alpha_1 = [pc_a^1 = \ell_1] \rightarrow reset(d); add(X, \{c\}_k); pc_a^1 := \ell_2$$

$$\alpha_2 = [pc_a^1 = \ell_2 \wedge in(X, \{T\}_k) \wedge g_2] \rightarrow pc_a^1 := \ell_3$$

$$\alpha_3 = [pc_a^1 = \ell_3] \rightarrow add(X, Secret); pc_a^1 := \ell_4,$$

where $c$ and $d$ are clocks, $k \in K$ is a symmetric key (intended to remain secret for the intruder), $Secret$ is a message (the secret) and $g_2 \equiv d = 1 \wedge -c + T < -1$. Let $\Gamma \stackrel{def}{=} X \lessdot \varepsilon \gtrdot_K Secret$ be the formula for which we want to compute the weakest liberal precondition (the secret $Secret$ isn't known by the intruder).

Now let $\Pi_0 = \alpha_0\alpha_1\alpha_2\alpha_3$ and $\Pi_1 = \alpha_1\alpha_2\alpha_3$ be two sequences. We show that $\Pi_1$ is secure with respect to formula $\Gamma$, while $\Pi_0$ is not. For sake of simplicity, we work modulo $\equiv$.

### $1^{st}$ *step*

$wlp(\xrightarrow{\tau}, X \lessdot \varepsilon \gtrdot_K Secret) = X \lessdot \varepsilon \gtrdot_K Secret.$

### $2^{nd}$ *step*

$$\begin{array}{rcl} wlp(\alpha_3, X \lessdot \varepsilon \gtrdot_K Secret) & = & pc_a^1 = \ell_3 \Rightarrow (X, Secret) \lessdot \varepsilon \gtrdot_K Secret \\ & \equiv & pc_a^1 \neq \ell_3 \end{array}$$

### $3^{rd}$ *step*

$wlp(\xrightarrow{\tau}, pc_a^1 \neq \ell_3) = pc_a^1 \neq \ell_3$

### $4^{th}$ *step*

$wlp(\alpha_2, pc_a^1 \neq \ell_3) = \Gamma_1$ where

$$\begin{array}{rcl} \Gamma_1 & \equiv & d = 1 \wedge -c + T < -1 \wedge pc_a^1 = \ell_2 \Rightarrow \\ & & X \lessdot \varepsilon \gtrdot_K \{\{T\}_k, T\} \vee X \lessdot \varepsilon \gtrdot_K \{\{T\}_k, k\} \vee \ell_3 \neq \ell_3 \\ & \equiv & -d < -1 \vee d < 1 \vee c - T \leq 1 \vee pc_a^1 \neq \ell_2 \vee X \lessdot \varepsilon \gtrdot_K \{\{T\}_k, T\} \vee X \lessdot \varepsilon \gtrdot_K \{\{T\}_k, k\} \end{array}$$

### $5^{th}$ *step*

$wlp(\xrightarrow{\tau}, \Gamma_1) = \Gamma_2$ where $\Gamma_2 \equiv \Psi_2 \vee \Phi_2$

$$\begin{array}{rcl} \Psi_2 & \equiv & \neg\Delta(d \leq 1, -d \leq 1) \vee \neg\Delta(d \leq 1, -c + T < -1) \vee -d < -1 \\ & \equiv & \bot \vee -d + c - T \leq 0 \vee -d < -1 \end{array}$$

$$\Phi_2 \equiv pc_a^1 \neq \ell_2 \vee X \lessdot \varepsilon \gtrdot_K \{\{T\}_k, T\} \vee X \lessdot \varepsilon \gtrdot_K \{\{T\}_k, k\}$$

hence,

$$\Gamma 2 \equiv -d + c - T \leq 0 \vee -d < -1 \vee pc_a^1 \neq \ell_2 \vee X \lessdot \varepsilon \gtrdot_K \{\{T\}_k, T\} \vee X \lessdot \varepsilon \gtrdot_K \{\{T\}_k, k\}$$

$6^{th}$ *step*

$wlp(\alpha_1, \Gamma_2) = \Gamma_3$ where

$$\begin{array}{rcl}
\Gamma_3 & \equiv & pc_a^1 = \ell_1 \Rightarrow 0 + c - T \leq 0 \vee 0 < -1 \vee \ell_2 \neq \ell_2 \\
& & \vee\big(T_c = c \Rightarrow X{\lhd\varepsilon\rhd}_K\{\{T\}_k, T\} \wedge \{T_c\}_k{\lhd\varepsilon\rhd}_K\{\{T\}_k, T\}\big) \\
& & \vee\big(T_c = c \Rightarrow X{\lhd\varepsilon\rhd}_K\{\{T\}_k, k\} \wedge \{T_c\}_k{\lhd\varepsilon\rhd}_K\{\{T\}_k, k\}\big) \\
& \equiv & pc_a^1 = \ell_1 \Rightarrow c - T \leq 0 \vee T_c \neq c \vee (T_c \neq T \wedge (X{\lhd\varepsilon\rhd}_K\{\{T\}_k, T\} \vee X{\lhd\varepsilon\rhd}_K\{\{T\}_k, k\})) \\
& \equiv & pc_a^1 = \ell_1 \Rightarrow (c - T \leq 0 \vee T_c \neq c \vee T_c \neq T) \\
& & \wedge(c - T \leq 0 \vee T_c \neq c \vee X{\lhd\varepsilon\rhd}_K\{\{T\}_k, T\} \vee X{\lhd\varepsilon\rhd}_K\{\{T\}_k, k\})
\end{array}$$

we observe that

$(c - T \leq 0 \vee T_c \neq c \vee T_c \neq T) \equiv ((T_c = c \wedge T_c = T) \Rightarrow c \leq T) \equiv \top$

hence

$$\Gamma_3 \equiv pc_a^1 \neq \ell_1 \vee c - T \leq 0 \vee T_c \neq c \vee X{\lhd\varepsilon\rhd}_K\{\{T\}_k, T\} \vee X{\lhd\varepsilon\rhd}_K\{\{T\}_k, k\}$$

$7^{th}$ *step*

$wlp(\xrightarrow{\tau}, \Gamma_3) = \Gamma_4$ where $\Gamma_4 \equiv \Psi_4 \vee \Phi_4$

$$\begin{array}{rcl}
\Psi_4 & \equiv & \neg\Delta(-T_c + c \leq 0, -c + T < 0) \vee \neg\Delta(-T_c + c \leq 0, T_c - c \leq 0) \vee T_c - c < 0 \\
& \equiv & T_c - T < 0 \vee \bot \vee T_c - c < 0
\end{array}$$

$$\Phi_4 \equiv pc_a^1 \neq \ell_1 \vee X{\lhd\varepsilon\rhd}_K\{\{T\}_k, T\} \vee X{\lhd\varepsilon\rhd}_K\{\{T\}_k, k\}$$

hence

$$\Gamma_4 \equiv T_c - T < 0 \vee T_c - c < 0 \vee pc_a^1 \neq \ell_1 \vee X{\lhd\varepsilon\rhd}_K\{\{T\}_k, T\} \vee X{\lhd\varepsilon\rhd}_K\{\{T\}_k, k\}$$

$8^{th}$ *step*

$wlp(\alpha_0, \Gamma_4) = \Gamma_5$ where

$$\begin{array}{rcl}
\Gamma_5 & \equiv & pc_a^1 = \ell_0 \Rightarrow T_c - T < 0 \vee T_c - c < 0 \vee \ell_1 \neq \ell_1 \\
& & \vee\big(T'_c = c \Rightarrow (X{\lhd\varepsilon\rhd}_K\{\{T\}_k, T\} \wedge \{T'_c\}_k{\lhd\varepsilon\rhd}_K\{\{T\}_k, T\})\big) \\
& & \vee\big(T'_c = c \Rightarrow (X{\lhd\varepsilon\rhd}_K\{\{T\}_k, k\} \wedge \{T'_c\}_k{\lhd\varepsilon\rhd}_K\{\{T\}_k, k\})\big) \\
& \equiv & pc_a^1 = \ell_0 \Rightarrow T_c - T < 0 \vee T_c - c < 0 \vee T'_c \neq c \\
& & \vee(T'_c \neq T \wedge (X{\lhd\varepsilon\rhd}_K\{\{T\}_k, T\} \vee X{\lhd\varepsilon\rhd}_K\{\{T\}_k, k\})) \\
& \equiv & (pc_a^1 \neq \ell_0 \vee T_c - T < 0 \vee T_c - c < 0 \vee T'_c \neq c \vee T'_c \neq T) \wedge (pc_a^1 \neq \ell_0 \\
& & \vee T_c - T < 0 \vee T_c - c < 0 \vee T'_c \neq c \vee X{\lhd\varepsilon\rhd}_K\{\{T\}_k, T\} \vee X{\lhd\varepsilon\rhd}_K\{\{T\}_k, k\})
\end{array}$$

$9^{th}$ *step*

$wlp(\xrightarrow{\tau}, \Gamma_5) = \Gamma_6$ where $\Gamma_6 \equiv (\Psi_6 \vee \Phi_6) \wedge (\Psi_7 \vee \Phi_7)$

$$\begin{array}{rcl}
\Psi_6 & \equiv & \neg\Delta(c - T_c \leq 0, -c + T'_c \leq 0) \vee \neg\Delta(c - T'_c, -c + T'_c \leq 0) \\
& & \vee T_c - c < 0 \vee -c + T'_c < 0 \vee T_c - T < 0 \vee T'_c \neq T \\
& \equiv & T_c - T'_c < 0 \vee \bot \vee T_c - c < 0 \vee -c + T'_c < 0 \vee T_c - T < 0 \vee T'_c \neq T
\end{array}$$

$$\begin{aligned}
\Psi_7 &\equiv \neg\Delta(c - T_c \leq 0, -c + T'_c \leq 0) \vee \neg\Delta(c - T'_c, -c + T'_c \leq 0) \\
&\quad \vee T_c - c < 0 \vee -c + T'_c < 0 \vee T_c - T < 0 \\
&\equiv T_c - T'_c < 0 \vee \bot \vee T_c - c < 0 \vee -c + T'_c < 0 \vee T_c - T < 0 \\
\Phi_6 &\equiv pc_a^1 \neq \ell_0 \\
\Phi_7 &\equiv pc_a^1 \neq \ell_0 \vee X \lessdot \varepsilon \gtrdot_K \{\{T\}_k, T\} \vee X \lessdot \varepsilon \gtrdot_K \{\{T\}_k, k\}
\end{aligned}$$

hence

$$\begin{aligned}
\Gamma_6 &\equiv pc_a^1 \neq \ell_0 \vee T_c - T'_c < 0 \vee \bot \vee T_c - c < 0 \vee -c + T'_c < 0 \vee T_c - T < 0 \\
&\quad \vee (T'_c \neq T \wedge (X \lessdot \varepsilon \gtrdot_K \{\{T\}_k, T\} \vee X \lessdot \varepsilon \gtrdot_K \{\{T\}_k, k\}))
\end{aligned}$$

Hence, we obtain $wlp(\Pi_0, X \lessdot \varepsilon \gtrdot_K Secret) \equiv \Gamma_6$ and $wlp(\Pi_1, X \lessdot \varepsilon \gtrdot_K Secret) \equiv \Gamma_4$

Since we supposed that $k$ is a secret symmetric key (i.e. $X \lessdot \varepsilon \gtrdot_K k$) and moreover, if there are no messages of the form $\{t\}_k$, initially known by the intruder with $t \in \mathbb{R}_{\geq 0}$, the protocol $\Pi_1$ is secure with respect to the secrecy of $Secret$. Contrarily, the protocol $\Pi_0$ is unsecure. If we pick $T_c$ and $T'_c$ such that $T < T_c \wedge T'_c < T_c \wedge T'_c = T$, then we obtain an attack which corresponds to the fact that the first message sent can be replayed successfully by the intruder (it satisfies the time constraints).

## 5.3 Satisfiability of TTTL$_\exists$

We define the TTTL$_\exists$ fragment of TTTL by restricting the TTL formulae to TTL$_\exists$ formulae. In this section we show how we can extend the satisfiability of TTL$_\exists$ formulae to the satisfiability of TTTL$_\exists$ formulae.

Let us consider a general formula of the form:

$$\Gamma = \exists T_1 \ldots \exists T_m \exists x_1 \ldots \exists x_m \forall f_1 \ldots \forall f_p \varphi$$

where $\{T_i \in \mathcal{Y} \mid i = 1 \ldots m\} \cup \{x_i \in \mathcal{X} \mid i = 1 \ldots m\} \cup \{f_k \in \mathcal{B} \mid k = 1 \ldots p\}$ is the set of all variables that appear in $\varphi$, and $\varphi$ is a quantifier free formula built using the connectives $\wedge$ and $\vee$ and literals of TTTL$_\exists$.

As the time constraints do not contain meta variables of $\mathcal{B}$ and the function symbols of $\mathcal{B}$ are applied only to variables of $\mathcal{X}$ we eliminate the universal quantifiers in the same way as for the TTL$_\exists$ (section 4.1.4). Moreover, as we consider the satisfiability problem we can restrict ourselves to conjunctions of literals.

Hence, we have to deal with restricted formula of the following form:

$$\exists T_1 \ldots \exists T_m \exists x_1 \ldots \exists x_m (\psi \wedge \varphi)$$

where $\psi$ is a conjunction of time constraints and $\varphi$ is a conjunction of TTL$_\exists$ formulae.

$\varphi$ can be transformed into a set of solved form formulae as shown in the section 4.1.3. But, the terms may contain clocks and time variables and therefore, some rules are needed to handle them. These rules are presented in table 5.1.

$$x = \{t\}_k \mapsto \bot \quad x = (t_1, t_2) \mapsto \bot \quad x = N \mapsto \bot \quad x = P \mapsto \bot$$

$$x = k \mapsto \bot \qquad \text{if } x \in \mathcal{C} \cup \mathcal{Y} \cup \mathbb{R},\ N \in \mathcal{N},\ P \in \mathcal{P} \text{ and } k \in \mathcal{K}$$

Table 5.1: Eliminate trivial sub-formulae for TTTL

$$t{\lessdot}\varepsilon{\gtrdot}_K s \mapsto \mathcal{J}(t, \varepsilon, s), \quad \text{if } t \notin \mathcal{X} \cup \mathcal{C} \cup \mathcal{Y} \cup \mathbb{R}_{\geq 0} \qquad \textbf{(D1.1)}$$

$$t{\lessdot}\varepsilon{\gtrdot}_K s \mapsto t \neq s, \quad \text{if } t \in \mathcal{C} \cup \mathcal{Y} \cup \mathbb{R}_{\geq 0} \qquad \textbf{(D1.2)}$$

$$s = t \mapsto \mu(s, t), \quad \text{if } s, t \notin \mathcal{X} \cup \mathcal{C} \cup \mathcal{Y} \cup \mathbb{R}_{\geq 0} \qquad \textbf{(D2)}$$

Table 5.2: Decompose

Also, the decomposition rules have to be modified as shown in table 5.2.

It turns out that only the equalities between the variables in $\mathcal{C} \cup \mathcal{Y}$ that are implied by $\psi$ might rule out some of the models of $\varphi$. That is, we need only to take into account such equalities.

Let us illustrate this by an example. Consider the formula

$$\varphi' \equiv x{\lessdot}\epsilon{\gtrdot}_K(A, c) \wedge x{\lessdot}\not{\epsilon}{\gtrdot}_K(A, T).$$

Then, $\varphi' \wedge 0 \leq c \leq 1 \wedge 0 \leq T \leq 1$ is satisfiable; while $\varphi' \wedge c - T = 0$ is not. Indeed, in the first case the time constraint does not imply any equality; while in the second case it implies $c = T$.

Therefore, we proceed as follows. We compute the strongest time constraint $\psi'$ of the form $\bot$ or

$$\bigwedge_{i=1}^{m} z_i = z_i'$$

where $z_i, z_i' \in \mathcal{C} \cup \mathcal{Y}$ and such that $\psi$ implies $\psi'$. If $\psi'$ is $\bot$ then $\psi \wedge \varphi$ is not satisfiable, and we are done.

Therefore, let us suppose that $\psi'$ is satisfiable. Then, $\psi'$ induces an equivalence relation on the variables in $\mathcal{C} \cup \mathcal{Y}$ as follows: $z \sim_{\psi'} z'$ iff $z = z'$ is a consequence of $\psi'$. Using this equivalence relation we can define an idempotent substitution $\sigma_{\psi'}$ that associates to the members of an equivalence class a designated representative. Now, we apply the substitution $\sigma_{\psi'}$ to $\varphi$ and check that the obtained formula is satisfiable.

We only consider term formulae as the complexity of time constraints is well-known [Sch86]. Moreover, as seen above time constraints can be eliminated leading to a formula $\psi'$. This can be done in $NP$-time and the size of $\psi'$ is polynomial in the size of $\psi$. Hence, the satisfiability problem of TTTL formula is also $NP$-complete.

# Part II

# Unbounded Protocols

# Chapter 6

# Abstraction

We introduce in this chapter our underlying model for the verification by abstraction of unbounded cryptographic protocols.

We begin by defining an operational semantics for protocols in an unbounded setting. Our starting point are syntactic descriptions of untimed cryptographic protocols by parameterized session, as defined in section 1.3. Regarding the semantics, there is little difference between the bounded case and the unbounded one. Roughly speaking, configurations includes now varying sets of *active* session instances and, moreover, an action to create a new session instance is added. That is, contrarily to the bounded case where we instantiate and run a fixed number of protocol sessions, we give here the possibility to create and run new sessions of the protocol, at any time.

In the unbounded case, we will restrict our attention to the verification of secrecy properties. In fact, as you will see later, the use of the control abstraction will prevent us to express authentication properties, which requires precise information about control points of participants. Nevertheless, even restricted to secrecy properties, the verification problem is not easy. In fact, in addition to the unbounded nature, now in two dimensions (size of messages and number of sessions) the verification problem is implicitly quantified universally over the set of session instances. That is, we require that some secret is preserved in each session instance, and despite the number of session instances created.

In order to tackle the complexity of this problem we propose an abstraction of the protocol with respect to the secrecy property and a witness session, that is, a session taking place between honest participants. Our abstraction combines *data abstraction* and *control* abstraction. Data abstraction allows to map the infinite domains of principal names and nonces to some finite domains, more precisely, we will distinguish only the honest participants occurring in the witness session while the others are becoming the same as the intruder. Control abstraction allows to map any unknown number of concrete parallel sessions to a finite number of abstract sessions. The keypoint is to loose precise information about control points of participants, more precisely, to replace their concrete actions with abstract actions (later called abstract rules) where the order

of occurrence is just partially preserved.

Finally, we show that our abstraction is sound with respect to secrecy properties. That is, if the secrecy property is satisfied on the abstract protocol, then so it is on the concrete protocol too. Roughly speaking, if the secret is preserved on the witness session, it will be preserved on any session of the protocol taking place between honest participants. Nevertheless, the converse does not hold, our abstraction not being complete. It is indeed possible that the secrecy property does not hold on the abstract protocol and still holds on the concrete protocol.

## 6.1 Unbounded Semantics

Let us consider $\mathcal{S} = (P, act, fresh)$ a parameterized session description. We want to represent the behavior of the protocol described by $\mathcal{S}$ without any restriction on the numbers of sessions and principals.

We recall that a session instance $S_i$ is characterized by a pair $(i, \pi)$. We defined the *states* of a session instance $S_i$ by the pairs $(\sigma, al)$, where $\sigma$ is a substitution, and $al$ is a function. The function $al$ associates for each role of the protocol $p \in P$ the list of its instantiated actions, which are left to be executed, initially in the session instance $S_i$, $al(p) = act^i(\pi(p))$.

The configurations set of unbounded protocol is given by pairs $(E, \xi)$ where $E$ is a set of messages and $\xi$ is a function, $dom(\xi) = \mathbb{N}$ which represents identifiers of session instances and $\xi(i)$ describes the state of the session instance $S_i$. The *operational semantics* is defined as a labelled transition system over the set of configurations. There are two sets of transitions:

1. *transitions that create new sessions:*

$$\frac{i \notin dom(\xi)}{(E, \xi) \xrightarrow{\tau} (E, \xi[i \mapsto (\sigma, al)])}$$

   where $\sigma$ is the empty substitution and $al$ is a function that associates for each role of the protocol $p \in P$ the list of instantiated actions $al(p) = act^i(\pi(p))$ and $\pi$ is an arbitrary assigment of principals to parameters. That corresponds to creating a new session $(i, \pi)$.

2. *transitions that correspond to protocol actions:*

$$(E, \xi) \xrightarrow{\alpha} (E', \xi')$$

   are defined by the following two rules:

   – *output actions:* $\alpha = [pc_p^i = \ell_1] \rightarrow add(X, t); pc_p^i := \ell_2$

$$\frac{i \in dom(\xi) \quad \xi(i) = (\sigma, al) \quad al(p) = \alpha \cdot list}{(E, \xi) \xrightarrow{\alpha} (E \cup \{\!|t\sigma|\!\}, \xi[i \mapsto (\sigma, al')])}$$

where $al' = al[p \mapsto list]$. That is, sending the message $t\sigma$ amounts to adding $t\sigma$ to the knowledge of the intruder.

– *input actions:* $\alpha = [pc_p^i = \ell_1 \wedge in(X, t(\tilde{x}))] \rightarrow pc_p^i := \ell_2$

$$\frac{i \in dom(\xi) \quad \xi(i) = (\sigma, al) \quad al(p) = \alpha \cdot list \quad \rho \in \Gamma(\tilde{x}) \quad E \vdash t(\sigma \oplus \rho)}{(E, \xi) \xrightarrow{\alpha} (E, \xi[i \mapsto (\sigma \oplus \rho, al')])}$$

where $al' = al[p \mapsto list]$. That corresponds to receiving any message that matches with $t\sigma$ and is known by the intruder.

**Example 6.1**

*Consider again our running example 1.3, the Needham-Schroeder protocol, and a session, identified by 0, between A and B, $\pi(p_1) = A, \pi(p_2) = B$, with principals A being at the second step of the protocol and B being in the last step of the protocol. The state $\xi(0)$ of the session is $(\sigma_0, al_0)$ where $\sigma_0 = [y \rightarrow A; z \rightarrow N_1]$, and*

$$\begin{aligned} al_0(p_1) &= [pc_{p_1}^0 = \ell_1 \wedge in(\{N_1, x\}_{\mathrm{pbk}(A)}, X)] \rightarrow pc_{p_1}^0 := \ell_2 \cdot \\ &\quad\ [pc_{p_1}^0 = \ell_2] \rightarrow add(\{x\}_{\mathrm{pbk}(B)}, X); pc_{p_1}^0 := \ell_3 \\ al_0(p_2) &= [pc_{p_2}^0 = \ell_2 \wedge in(\{N_2\}_{\mathrm{pbk}(B)}, X)] \rightarrow pc_{p_2}^0 := \ell_3 \end{aligned}$$

*Moreover, let $\{N_1, N_2\}_{K_A} \in E$ then A can fire its first remaining action which modifies the configuration as follows: $\sigma_0 = [y \rightarrow A; z \rightarrow N_1; x \rightarrow N_2]$, and*

$$\begin{aligned} al_0(p_1) &= [pc_{p_1}^0 = \ell_2] \rightarrow add(\{x\}_{\mathrm{pbk}(B)}, X); pc_{p_1}^0 := \ell_3 \\ al_0(p_2) &= [pc_{p_2}^0 = \ell_2 \wedge in(\{N_2\}_{\mathrm{pbk}(B)}, X)] \rightarrow pc_{p_2}^0 := \ell_3 \end{aligned}$$

■

## 6.2 Secrecy modeling

A *secrecy* goal states that a designated message should not be made *public*. A secret is public when it is deductible from the set of messages intercepted by the intruder. In our setting, a secret is defined by a role term. More precisely, to each session instance is associated a secret that we want to prove with respect to some hypothesis. Hence, we are interested in session instances where all participants are honest. If one of participants who are supposed to know the secret is dishonest it does not make sense to require that the secret associated to this session instance should not be made public.

Hence, the secret property for unbounded protocols is expressed in accordance with the honesty of protocol participants. A participant is *honest* if initially his private keys and his shared keys with other honest participants are not deductible from the intruder

knowledge and later it will execute only actions specified by the protocol. Also, for all instance values that are associated to the fresh parameters we have to express that they have not appeared before the session creation.

Let $\mathcal{S} = (P, act, fresh)$ be a given parameterized session description. Given $S_i = (i, \pi)$ a session instance, we denote by $C(E, \pi, i)$ the constraint which asserts for every nonce variable $n \in \bigcup_{p \in P} fresh(p)$, and for every message $m$ such that $E \vdash m$ we have $N^i$ (the instance of $n$ in the session $S_i$) does not appear in $m$.

Formally,

$$C(E, \pi, i) = \{E \not\vdash^{\in} N^i \mid n \in fresh(P)\}$$

where by $E \not\vdash^{\in} m$ we denote that $m$ does not appear in any message derivable from $E$.

Intuitively, this means that initially the intruder cannot know messages that contain fresh nonces.

Moreover, let $C(E)$ denote the condition:

$$\forall (i, \pi) \in \text{Inst} \; . \; C(E, \pi, i).$$

We model the honesty of a participant $p$ by a predicate honest($p$). All participants of the session instance $(i, \pi)$ are honest, denoted honest($\pi$), if for every $p$ of $P$ we have honest($\pi(p)$).

We are now ready to define our secrecy property formally. The protocol $P$ described by $\mathcal{S}$ satisfies the secrecy property defined by the secret template $t$ in the initial set $E_0$ of intruder's knowledge, denoted by $Secret(\mathcal{S}, t, E_0)$, if $C(E_0)$ and for every $E$ such that $(E_0, \xi_0) \rightarrow^* (E, \xi)$ and honest($\pi$) we have $E \not\vdash t^i_\pi$ where $(i, \pi)$ are session instances of Inst and $dom(\xi_0) = \emptyset$.

Formally, in $C(E_0)$ hypothesis we have

$$Secret(\mathcal{S}, t, E_0) \Leftrightarrow \forall (i, \pi) \in \text{Inst} \; . \; \forall E \in \mathcal{M} \; . \; [\text{honest}(\pi) \wedge (E_0, \xi_0) \rightarrow^* (E, \xi)] \Rightarrow E \not\vdash t^i_\pi$$

The definition of secrecy can be easily extended to a set $T$ of secret templates by: $Secret(\mathcal{S}, T, E_0)$ iff $Secret(\mathcal{S}, t, E_0)$, for all $t \in T$.

## 6.3 Abstraction Definition

In this section we fix an arbitrary cryptographic protocol given by a session description $\mathcal{S} = (P, act, fresh)$. To prove any property of this protocol in the case of unbounded number of sessions, we are faced with the following problems:

1. The definition of the verification problem is a problem quantified universally over all session instances $(i, \pi) \in \text{Inst}$.

2. There is no bound on the number of sessions instances that can be created.

3. There is no bound on the size of the messages that occur during execution of the protocol.

We present here an abstraction that is parameterized by an arbitrary session instance $(i_0, \pi_0) \in \text{Inst}$, then we argue that the abstract system we obtain does not depend on the choice of the session instance $(i_0, \pi_0)$. The main idea of the abstraction is as follows. Clearly, the behavior of a honest participant does not depend on its identity. This is simply a consequence of defining protocol sessions in a parameterized manner as we did. Also, the behavior of a participant does not depend on the identifier associated to the session.

Therefore, we fix an arbitrary session where the participants are honest, say we have two participants, $A$ and $B$. We assume there is such a session, otherwise the hypothesis of the *Secret* property is never satisfied and we have nothing to prove. Then, we identify with the intruder $I$ all participants other than $A$ and $B$. Moreover, we identify all sessions in which neither $A$ nor $B$ are involved. Concerning the other sessions, that is, those where $A$ or $B$ are involved, we identify:

- all sessions where $A$ plays the role of $p_1$, $B$ plays the role of $p_2$ and the session is different from the fixed session,
- all sessions where $B$ plays the role of $p_1$ and $A$ plays the role of $p_2$,
- all sessions where $A$ (respectively $B$) plays the both roles
- all sessions where $A$ (respectively $B$) plays the role of $p_1$ and the role of $p_2$ is played by a participant different from $A$ or $B$,
- all sessions where $A$ (respectively $B$) plays the role of $p_2$ and the role of $p_1$ is played by a participant different from $A$ or $B$,

Identifying sessions means also identifying the nonces and keys used in these sessions. We now present this idea formally.

### 6.3.1 Data abstraction

Let the session instance $(i_0, \pi_0) \in \text{Inst}$ be fixed such that honest$(\pi_0)$ holds. We suppose that there is at least one honest participant. For a concrete semantic object $x$, we use the notation $x^{(i_0,\pi_0)}$ to denote its abstraction, and in case $(i_0, \pi_0)$ is known from the context, we use $x^\sharp$.

We start by defining the abstract domains $\mathbb{N}^\sharp = \{\top, \bot\}$ and $\mathcal{P}^\sharp = \{A_1, \cdots A_r, I\}$, where $r$ is the cardinal of $P$ and the abstractions:

- $i^\sharp = \begin{cases} \top & \text{if } (i, \pi) = (i_0, \pi_0) \\ \bot & \text{otherwise} \end{cases}$
- $p^\sharp = \begin{cases} A_j & \text{if } p = \pi_0(p_j) \\ I & \text{otherwise} \end{cases}$

We extend the abstraction of participants to vectors of participants by taking the abstractions of the components.

The abstraction of the nonce $N^i$, of a session instance $S_i = (i, \pi)$, denoted by $(N^i)^\sharp$, is given by:

- $N_I$, if $n \in fresh(p)$ and $\pi(p)^\sharp = I$,
- $N$, if $i^\sharp = \top$, and
- $N^{\pi^\sharp}$, otherwise.

where $N_I$ is a fresh constant.

Thus, as abstract sets of nonce, we have $\mathcal{N}^\sharp(I) = \{\!\!\{N_I\}\!\!\}$ and $\mathcal{N}^\sharp(A_j) = \{\!\!\{N, N^{\pi^\sharp} \mid n \in fresh(p_j), \pi(p_j) = A_j\}\!\!\}$.

**Example 6.2**

*For Needham-Schroeder, we have the following set of abstract nonces:*

$$\mathcal{N}^\sharp = \{\!\!\{N_I, N_1, N_2, {N_1}^{x,y}, {N_2}^{y,x} \mid x \in \{A, B\}\ y \in \{A, B, I\}\}\!\!\}.$$

■

We denote $\mathcal{N}^\sharp = \mathcal{N}^\sharp(I) \cup \bigcup_{j \in \mathbb{N}_r} \mathcal{N}^\sharp(A_j)$.

It remains to define the abstraction of keys. We take the abstract set $\mathcal{K}^\sharp$ that consists of a distinguished key $K_I$ and the keys in $\mathcal{AK}(i, [n^\sharp], p_1^\sharp, \cdots, p_l^\sharp)$ with $n^\sharp \in \mathcal{N}^\sharp$ $p_1^\sharp, \cdots, p_l^\sharp \in \mathcal{P}^\sharp$ and $p_j^\sharp \neq I$, for all $j \in \mathbb{N}_l$. The abstraction of a key $k(i, n, p_1, \cdots, p_n)$ is defined by:

$$k^\sharp([i,][n,]p_1, \ .. \ , p_n) = \begin{cases} k([i,][n^\sharp,]p_1^\sharp, \cdots, p_n^\sharp) & \text{if } p_i^\sharp \neq I,\ i = 1, \cdots, n \\ & \text{and } n^\sharp \neq N_I \\ K_I & \text{otherwise} \end{cases}$$

**Example 6.3**

*For Needham-Schroeder, we have the following set of abstract keys:*

$$\mathcal{K}^\sharp = \{\!\!\{K_I, \mathrm{pbk}(A), \mathrm{pvk}(A), \mathrm{pbk}(B), \mathrm{pvk}(B)\}\!\!\}.$$

■

We denote $\mathcal{A}^\sharp = \mathcal{P}^\sharp \cup \mathcal{N}^\sharp \cup \mathcal{K}^\sharp$.

The abstraction of a term $t$, denoted by $t^\sharp$, is obtained as the homomorphic extension of the abstractions on participants, nonces and keys. For a set $T$ of terms, let $T^\sharp = \{t^\sharp \mid t \in T\}$.

The set $\mathcal{M}^\sharp$ of abstract messages is the set of ground terms over $\mathcal{A}^\sharp$ and the constructors **encr** and **pair** as for $\mathcal{M}$. Similarly, we can define the set of abstract terms by allowing variables in $\mathcal{X}$.

### 6.3.2 Control abstraction

Now we have a system with a finite number of participants, of nonces and of keys but an unbounded number of sessions. Further, we apply an abstraction that removes the control, for all session instances.

To do so, we remove from the actions the program counters and in order to keep the value assigned to the variables we add to each guard of an output action all guards from the previously input actions of the same participant. Thus, an implicit control is given by these new guards. An output action is executed only if all previously input actions have been able to be executed. It is apparent that the input actions are redundant. We call the obtained set of actions, the set of *abstract actions* or *transitions*.

Let us give an example to show how to change a parameterized protocol description into a set of parametrized transitions:

**Example 6.4**

*We take again the Needham-Schroeder protocol. The first action of* $p_1$ *remains as it is, except that the explicit manipulation of the control point* $pc^i_{p_1}$ *is removed. The last two actions of* $p_1$ *are merged together. In fact, since the second action is an input, it becomes a guard of the third action. The situation on* $p_2$ *is similar. The first two actions are merged together into a transition. Moreover, the third action is simply discarded.*

| *actions* | *transitions* |
|---|---|
| $[pc_{p_1} = 0] \rightarrow add(X, \{p_1, n_1\}_{\mathrm{pbk}(p_2)}); pc_{p_1} := 1$ | $[] \rightarrow add(X, \{p_1, n_1\}_{\mathrm{pbk}(p_2)})$ |
| $[pc_{p_1} = 1 \wedge in(X, \{n_1, \tilde{x}\}_{\mathrm{pbk}(p_1)})] \rightarrow pc_{p_1} := 2$<br>$[pc_{p_1} = 2] \rightarrow add(X, \{x\}_{\mathrm{pbk}(p_2)}); pc_{p_1} := 3$ | $[in(X, \{n_1, x\}_{\mathrm{pbk}(p_1)})] \rightarrow add(X, \{x\}_{\mathrm{pbk}(p_2)})$ |
| $[pc_{p_2} = 0 \wedge in(X, \{\tilde{y}, \tilde{z}\}_{\mathrm{pbk}(p_2)})] \rightarrow pc_{p_2} := 1$<br>$[pc_{p_2} = 1] \rightarrow add(X, \{z, n_2\}_{\mathrm{pbk}(y)}); pc_{p_2} := 2$ | $[in(X, \{y, z\}_{\mathrm{pbk}(p_2)})] \rightarrow add(X, \{z, n_2\}_{\mathrm{pbk}(y)})$ |
| $[pc_{p_2} = 2 \wedge in(X, \{n_2\}_{\mathrm{pbk}(p_2)})] \rightarrow pc_{p_2} := 3$ | |

■

To simplify, we denote a transition of the form $[\bigwedge_{i=1}^{n} in(X, t_i)] \rightarrow add(X, t')$ by $t \rightarrow t'$ where $t$ is constructed by pairing from $t_1, \cdots t_n$. If $t \rightarrow t'$ is a $(i, \pi)$ instantiated transition then an abstract transition is $t^\sharp \rightarrow t'^\sharp$. Where $t^\sharp, t'^\sharp$ are the abstractions of the terms $t$ respectively $t'$.

To summarize, we have a system with a finite number of participants, of nonces and of keys but an unbounded number of sessions. We model the protocol as a set of abstract

transitions that can be taken in any order and any number of times. The number of messages as their size are left unbounded.

### 6.3.3 Protocol abstraction

We are now ready to define the abstraction of a cryptographic protocol with respect to a fixed session $(i_0, \pi_0)$ and a secrecy goal $s$ for this session.

#### Abstract protocol

The abstraction of a protocol will be given as a pair $(C^\sharp, R^\sharp)$ of constraints $C^\sharp$ of the form $E \not\vdash^\in m$, where $m \in \mathcal{M}^\sharp$ and a set $R^\sharp$ of abstract transitions. We call $(C^\sharp, R^\sharp)$ an *abstract protocol*. The abstract protocol configurations set are given by a sets of abstract messages $E^\sharp \subseteq \mathcal{M}^\sharp$. The *operational semantics* is defined by the following semantics rules:

$$\frac{}{E^\sharp \xrightarrow{\tau}_R E^\sharp}$$

$$\frac{t \to t' \in R^\sharp \quad E^\sharp \vdash t\sigma}{E^\sharp \xrightarrow{t \to t'}_R E^\sharp \cup \{t'\sigma\}}$$

#### Abstraction

The abstraction $\mathcal{S}^\sharp$ of the cryptographic protocol defined by $\mathcal{S} = (P, act, fresh)$ is defined by:

- the set $C$ of abstract constraints

$$C^\sharp = \{E^\sharp \not\vdash^\in m^\sharp \mid E \not\vdash^\in m \text{ in } C(E, \pi_0, i_0)\} \text{ and}$$

- the set $R^\sharp$ of abstract transitions (or abstract rules)

$$R^\sharp = \{t_1^\sharp \to t_2^\sharp \mid t_1 \to t_2 \text{ is a transition in some session instance } (i, \pi)\}$$

The abstraction of the freshness hypothesis $C^\sharp$ means that, in the abstract model the intruder cannot know messages that contain fresh nonces that will be generated by the fixed session. As we identify the others sessions we have to consider that the others nonces are known by the intruder from previous executions. A finer abstraction can be defined by distinguishing between sessions which finished (resp. starts) before (resp. after) the fixed session.

**Example 6.5**

*Let $\pi_0 = (A, B)$. In our model which yields an over-approximation of the possible runs of the protocol, we can describe the Needham-Schroeder protocol by the constraint $C^\sharp = \{E^\sharp \not\vdash^\in N_1, E^\sharp \not\vdash^\in N_2\}$ and the rules $R^\sharp$ of the figure 6.1. Where $X, Y$ range over the set $\pi_0$ of participants $A, B$.*

| Sessions | Transitions | | |
|---|---|---|---|
| the fixed session $(A, B)$ | $- \rightarrow \{A, N_1\}_{pbk(B)}$; | $\{A, y\}_{pbk(B)} \rightarrow \{y, N_2\}_{pbk(A)}$; | $\{N_1, z\}_{pbk(A)} \rightarrow \{z\}_{pbk(B)}$; |
| other sessions $(X, Y)$ | $- \rightarrow \{X, N_1^{XY}\}_{pbk(Y)}$; | $\{X, y\}_{pbk(Y)} \rightarrow \{y, N_2^{XY}\}_{pbk(X)}$; | $\{N_1^{XY}, z\}_{pbk(X)} \rightarrow \{z\}_{pbk(Y)}$; |
| the sessions $(I, Y)$ | $- \rightarrow \{I, N_I\}_{pbk(Y)}$; | $\{I, y\}_{pbk(Y)} \rightarrow \{y, N_2^{IY}\}_{pbk(I)}$; | $\{N_I, z\}_{pbk(I)} \rightarrow \{z\}_{pbk(Y)}$; |
| the session $(X, I)$ | $- \rightarrow \{X, N_1^{XI}\}_{pbk(I)}$; | $\{X, y\}_{pbk(I)} \rightarrow \{y, N_I\}_{pbk(X)}$; | $\{N_1^{XI}, z\}_{pbk(X)} \rightarrow \{z\}_{pbk(I)}$; |

Figure 6.1: The abstract rules of Needham-Schroeder Protocol

■

## 6.4 Soundness

Let $\mathcal{S}^\sharp = (C^\sharp, R^\sharp)$ be an abstract protocol and $E_0^\sharp \subseteq \mathcal{M}^\sharp$. We say that $\mathcal{S}^\sharp$ *preserves the secret* $s^\sharp$ *in* $E_0^\sharp$, denoted by $Secret^\sharp(\mathcal{S}^\sharp, s^\sharp, E_0^\sharp)$, if for all $E^\sharp \subseteq \mathcal{M}^\sharp$, if $C^\sharp$ and $E_0^\sharp \rightarrow_R^* E^\sharp$ then $E^\sharp \not\vdash s^\sharp$.

**Example 6.6**

*In our running example, the abstract hypothesis are*

$$E_0^\sharp \not\vdash^{\in_c} N_1$$
$$E_0^\sharp \not\vdash^{\in_c} N_2$$
$$honest(A), E_0^\sharp \not\vdash \mathrm{pvk}(A)$$
$$honest(B), E_0^\sharp \not\vdash \mathrm{pvk}(B)$$

*and the secret property is:*

$$E_0^\sharp \rightarrow_R^* E^\sharp \quad \Rightarrow \quad E^\sharp \not\vdash N_2$$

■

To relate a cryptographic protocol and its abstraction, we need to relate derivation by the intruder on the concrete and abstract messages. We can prove the following:

**Lemma 6.1**

*Let $E$ be a set of messages and $E^\sharp = \{m^\sharp \mid m \in E\}$. Then, $E \vdash m$ implies $E^\sharp \vdash m^\sharp$, for any message $m \in \mathcal{M}$.*

■

**Proof:**

By induction on the tree derivation

1. $E \vdash m$ in one step: Hence, $m \in E$. By the definition of $E^\sharp = \{m^\sharp \mid m \in E\}$ then $m^\sharp \in E^\sharp$ and then $E^\sharp \vdash m^\sharp$.

2. Induction step. $E \vdash m$ in $k+1$ steps. We make a case analysis on the last derivation step:

   - Case of pairing, $m = (m_1, m_2)$. We have $E \vdash m_1$ and $E \vdash m_2$ in $k$ steps, then by induction hypothesis $E^\sharp \vdash m_1^\sharp$ and $E^\sharp \vdash m_2^\sharp$ and by **pair** rule we have $E^\sharp \vdash (m_1^\sharp, m_2^\sharp)$ but $(m_1^\sharp, m_2^\sharp) = (m_1, m_2)^\sharp$ so $E^\sharp \vdash (m_1, m_2)^\sharp$.
   - Case of encryption, $m = \{t\}_k$. Similarly to the previous case.
   - Case of left projection. We have $E \vdash (m, m')$ in $k$ steps, then by induction hypothesis $E^\sharp \vdash (m, m')^\sharp$ but $(m, m')^\sharp = (m^\sharp, m'^\sharp)$ so $E^\sharp \vdash (m^\sharp, m'^\sharp)$ and by left projection rule we have $E^\sharp \vdash m^\sharp$. Similarly for right projection.
   - Case of decryption. We have $E \vdash \{m\}_k$ and $E \vdash inv(k)$ in $k$ steps, then by induction hypothesis $E^\sharp \vdash \{m\}_k^\sharp$ and $E^\sharp \vdash inv(k)^\sharp$ which is equivalent with $E^\sharp \vdash \{m^\sharp\}_{k^\sharp}$ and $E^\sharp \vdash inv(k^\sharp)$ then by decryption rule we have $E^\sharp \vdash m^\sharp$.

■

We can also prove the following lemma to relate concrete and abstract term instantiations:

**Lemma 6.2**

*Let $t_1$ and $t_2$ be terms and let $\rho : \mathcal{X} \to \mathcal{M}$. Then, $\rho(t_1) = \rho(t_2)$ implies $\rho^\sharp(t_1^\sharp) = \rho^\sharp(t_2^\sharp)$, where $\rho^\sharp$ is defined by $\rho^\sharp(x) = (\rho(x))^\sharp$ for any $x \in dom(\rho)$.*

■

**Proof:**

We have $\rho(t_1) = \rho(t_2)$ implies $\rho(t_1)^\sharp = \rho(t_2)^\sharp$ and we prove by structural induction on the term $t$ that $\rho(t)^\sharp = \rho^\sharp(t^\sharp)$:

1. case $t$ atomic - by definition

2. case $t = f(t_1, t_2)$, where $f \in \mathbf{\{pair, encr\}}$:

$$\begin{aligned}
\rho(t)^\sharp &= \rho(f(t_1, t_2))^\sharp \\
&= f(\rho(t_1), \rho(t_2))^\sharp \\
&= f(\rho(t_1)^\sharp, \rho(t_2)^\sharp) \\
&= f(\rho^\sharp(t_1^\sharp), \rho^\sharp(t_2^\sharp)) \text{ by induction hypothesis} \\
&= \rho^\sharp(f(t_1^\sharp, t_2^\sharp)) \\
&= \rho^\sharp(f(t_1, t_2)^\sharp) \\
&= \rho^\sharp(t^\sharp)
\end{aligned}$$

■

Using lemma 6.1, we can prove that $(C^\sharp, R^\sharp)$ is indeed an abstraction of $\mathcal{S}$ where the abstraction of a configuration $(E, \xi)$ is $E^\sharp$:

**Proposition 6.3**

*Let* $\mathcal{S} = (P, \text{act}, \text{fresh})$ *be a protocol and* $\mathcal{S}^\sharp = (C^\sharp, R^\sharp)$ *its abstraction. Let* $(E_k, \xi_k)$ *and* $(E_{k+1}, \xi_{k+1})$ *be reachable concrete configurations and* $\alpha$ *a protocol action Then,*

$$(E_k, \xi_k) \xrightarrow{\alpha} (E_{k+1}, \xi_{k+1}) \text{ implies } \exists \gamma \in R^\sharp \cup \{\tau\} \text{ such that } E_k^\sharp \xrightarrow{\gamma}_R E_{k+1}^\sharp.$$

*Moreover, if* $C(E)$ *is true then also* $C^\sharp$.

■

**Proof:**

Following the protocol actions we have two cases:

1. $\alpha$ corresponds to the creation of a new session $(i, \pi)$ or to an input action:

   We have $E_k = E_{k+1}$ and then $E_k^\sharp \xrightarrow{\tau}_R E_{k+1}^\sharp$ by definition.

2. $\alpha = [pc_p^i = \ell_1] \rightarrow add(X, t'); pc_p^i := \ell_2$ is an output action:

   We have $(E_k, \xi_k) \xrightarrow{\alpha} (E_{k+1}, \xi_{k+1})$ where $\xi_k = (\sigma_k, al_k)$ and $\xi_{k+1} = (\sigma_k, al_{k+1})$ and $E_{k+1} = E_k \cup (t'\sigma_k)$. We will prove that there is an abstract transition $\gamma$ in $R^\sharp$ such that $E_k^\sharp \xrightarrow{\gamma}_R E_k^\sharp \cup \{(t'\sigma_k)^\sharp\}$.

   As $(E_k, \xi_k)$ is a reachable configuration, we know that between initial configuration $(E_0, \xi_0)$ and $(E_k, \xi_k)$ all input actions of the participant $\pi(p)$ of session instance $(i, \pi)$ that are before the local control point $\ell_1$ has been executed. I.e. for every such input action $\alpha_j = [pc_p^i = \ell_1^j \wedge in(X, t^j(\tilde{x}))] \rightarrow pc_p^i := \ell_2^j$, $j \in J$ there are two configurations $(E_j, \xi_j), (E_{j+1}, \xi_{j+1})$ such that $(E_j, \xi_j) \xrightarrow{\alpha_j} (E_{j+1}, \xi_{j+1})$, $0 \leq j \leq k$. For every $j$ smaller than $k$, we have by definition $E_j$ subset of $E_k$ and $\sigma_k$ include $\sigma_j$ where $\xi_j(i) = (\sigma_j, _)$ respectively $\xi_k(i) = (\sigma_k, _)$.

   Hence, for all $j \in J$ we have $E_k \vdash t^j\sigma_k$ and by lemma 6.1 we have

   $$\forall j \in J \quad E_k^\sharp \vdash (t^j\sigma_k)^\sharp \tag{6.1}$$

   $\alpha_j, j \in J$ are all input actions of the participant $\pi(p)$ in the session instance $(i, \pi)$ before the output action $\alpha$. Hence, for this session instance we have the transition $t \rightarrow t'$ where $t$ is construct by pairing from $(t^j)_{j \in J}$. Therefore, in the abstract transition $R^\sharp$ of the protocol we have $t^\sharp \rightarrow t'^\sharp$. Then, using the equation 6.1 we have $E_k^\sharp \xrightarrow{t^\sharp \rightarrow t'^\sharp}_R E_k^\sharp \cup \{(t'\sigma_k)^\sharp\}$.

■

Exploiting Proposition 6.3 and the fact that $(C^\sharp, R^\sharp)$ does not depend on $(i_0, \pi_0)$, that is, we have the same constraints and transitions for all $(i, \pi) \in \text{Inst}$, we can prove:

**Theorem 6.4**

*The protocol defined by $\mathcal{S}$ satisfies the secrecy property defined by $Secret(\mathcal{S}, s, E_0)$, if its abstraction $(C^\sharp, R^\sharp)$ preserves $s^\sharp$ in $E_0^\sharp$, i.e.,*

$$Secret^\sharp(\mathcal{S}^\sharp, s^\sharp, E_0^\sharp) \text{ implies } Secret(\mathcal{S}, s, E_0).$$

■

**Proof:**

From the proposition 6.3 we have $C(E_0) \Rightarrow C^\sharp$, hence the hypothesis are preserved by the abstraction.

$Secret^\sharp(\mathcal{S}^\sharp, s^\sharp, E_0^\sharp) \quad \Leftrightarrow$

$\forall E^\sharp \in \mathcal{M}^\sharp \cdot E_0^\sharp \rightarrow_R^* E^\sharp \Rightarrow E^\sharp \not\vdash s^\sharp \quad \Leftrightarrow$

$\forall \pi_0 \in \text{Inst} \cdot \text{honest}(\pi_0)$ we have $\forall E^\sharp \in \mathcal{M}^\sharp \cdot E_0^\sharp \rightarrow_R^* E^\sharp \Rightarrow E^\sharp \not\vdash s^\sharp$

We want to prove $Secret(\mathcal{S}, s, E_0)$, that is,$\forall (i, \pi) \in Inst$ such that honest$(\pi)$ and $\forall E \subseteq \mathcal{M}$ such that $E_0 \rightarrow^* E$ we have $E \not\vdash s_\pi^i$.

If we suppose $\neg Secret(\mathcal{S}, s, E_0)$. That is $\exists (i', \pi') \in Inst$ such that honest$(\pi')$ and $\exists E' \subseteq \mathcal{M}$ such that $E_0 \rightarrow^* E'$ and $E' \vdash s_{\pi'}^{i'}$.

Let $\pi_0 = \pi'$. Using the proposition 6.3 we have $C^\sharp$ and $E_0^\sharp \rightarrow^* E'^\sharp$. Also, from $E' \vdash s_{\pi'}^{i'}$ using the lemma 6.1 we obtain $E'^\sharp \vdash (s_{\pi'}^{i'})^\sharp$, that is $E'^\sharp \vdash s^\sharp$.

Hence, there is $\pi'$, honest$(\pi')$ such that $C^\sharp$ and $\exists E'^\sharp \in \mathcal{M}^\sharp \cdot E_0^\sharp \rightarrow_R^* E'^\sharp$ and $E'^\sharp \vdash s^\sharp$, contradiction.

■

# Chapter 7

# Symbolic Verification

This chapter presents our symbolic verification algorithm for secrecy properties on unbounded cryptographic protocols. The algorithm takes as inputs (1) an abstract protocol $(C, R)$ consisting of an hypothesis on the initial intruder knowledge $C$ and a set of abstract rules $R$ and (2) the set of secrets $S$ to be preserved by the protocol. At termination, the algorithm answers *yes* or *no*, depending if indeed, the set of secrets is or not preserved by the protocol. In particular, if not, an abstract attack defined in terms of abstract rules applied, that are abstract protocol steps, is also provided as a counter-example.

The algorithm relies on the syntactic characterization of secrecy by message transducers as defined in section 2.2. In few words, the algorithm attempts to compute a minimal extended set of secrets $S'$ together with a minimal set of message transducers $MT$ such that:

1. the initial set of secrets $S$ is included in the extended set $S'$
2. the pair $(MT, S')$ is *well formed*, that is, if the secrets are protected from the intruder, despite of the use of message transducers, they remain also protected in any messages derivable by the intruder,
3. the pair $(MT, S')$ is *stable* with respect to the protocol rules, that is, no more useful knowledge can be obtained by the intruder through the application of any of the rules of the protocol.

Once the pair $(MT, S')$ satisfying the properties above is computed, the verification problem is reduced to check if the intersection between the set of messages accessible by $MT$ from initial knowledge of the intruder $C$ and the extended set of secrets $S'$ is or not empty.

Therefore, the main computational work of our algorithm consists in finding the appropriate sets $MT$ and $S'$. In order to compute them, we propose a least fixpoint technique. We start with an empty set of message transducers together with the initial set of secrets. Then, we progressively add message transducers and/or secrets until the pair

become stable and well-formed. In particular, non-stability with respect to a protocol rule gives an effective way to compute new secrets and new message transducers. Much simpler, well-formedness reduces to have a closed set of extended secrets and another structural property on the set of message transducers.

If we want to implement the above algorithm we are faced with two major problems. First, the sets of secrets and respectively message transducers can growth infinitely, hence, we need finite, symbolic representations for potentially infinite sets of such elements. For sets of messages, we propose to use *patterns*, that are, terms extended with the *Sup* operator, where $Sup(t)$ denotes the set of all terms containing as subterm $t$. Similarly, for message transducers, we will use now pattern transducers.

The second problem concerns the termination of the above algorithm. It is rather easy to provide examples where the fixpoint computation does not terminate, despite of the use of symbolic representations based on patterns. Therefore, in order to enforce termination, we propose a widening technique which detects and approximates potentially infinitely growing sequences of messages (or in general, patterns) by only a finite set of patterns. Clearly, the precision of the algorithm is lost here, the sets computed being over-approximations of the exact results. But, on all practical examples we tested, our widening scheme ensure termination and, it is precise enough to prove interesting secrecy properties.

In this chapter we assume that we are given a protocol $\Pi = (C, R)$ and a set of secrets defined by a set *Secret* of messages.

## 7.1 Secret characterization

In this section we deal with a particular case of the protected modality which use sets of one length message transducers. We recall or reformulate some definitions from the first part and we prove the same properties on this modality.

As before, we consider $K \subseteq \mathcal{K}$ be a fixed but arbitrary set of keys, such that $\emptyset \neq K \neq \mathcal{K}$, i.e. the set of keys for which their inverses are supposed to be unknown for the intruder. Also, $K^{-1} = \{k \mid k' \in K \wedge \mathrm{inv}(k, k')\}$ is the set of keys for which their inverse are in $K$.

By abstraction we removed the control, so the rules of $R$ can be taken any number of times and in any order. Therefore, we have to relax the protected modality, i.e. the order of message transducers as well as the fact that they are consumed by a message transducer application are removed. Hence, we define an abstract protected modality called set-protected modality as follows:

**Definition 7.1 (set-protected modality)**

*Let $m, s \in \mathcal{M}$ be two messages and* MT *be a set of message transducers of the form $(\{m'\}_k, r)$, where $m' \in \mathcal{M}$ is a message, $k \in K$ is a key and $r$ is a position in $\{m'\}_k$. We say that $s$ is* **set-protected** *in $m$ despite* MT*, denoted by $m \ll \mathrm{MT} \gg_K s$ if the message $s$ is not accessible by the message transducer $\left(Id + \sum_{(b,r)\in \mathrm{MT}} (b, r)\right)^*$ from the message $m$.*

$$m{\ll}\mathrm{MT}{\gg}_K s \Leftrightarrow s \notin \left( Id + \sum_{(b,r)\in \mathrm{MT}} (b,r) \right)^*_{\preceq} (m)$$

*For $M$ and $S$ sets of messages, we say that the secrets $S$ are set-protected in $M$ despite* MT*, denoted by $M{\ll}\mathrm{MT}{\gg}_K S$, if it holds* $\bigwedge_{m\in M, s\in S} m{\ll}\mathrm{MT}{\gg}_K s$.

■

The following equivalences are direct consequences of set-protected definition.

**Proposition 7.1**

*When the set of message transducers is empty the set-protected modality is equivalent with the protected modality where the sequence of message transducers is $\varepsilon$.*

$$m{\ll}\emptyset{\gg}_K s \Leftrightarrow m\langle\varepsilon\rangle_K s.$$

*Also, if* MT *is a set of one length message transducers it holds:*

$$m{\ll}\mathrm{MT}{\gg}_K s \Leftrightarrow \bigwedge_{w\in\left(\sum_{(b,r)\in\mathrm{MT}}(b,r)\right)^*} m\langle w\rangle_K s.$$

■

From now on we use sets of message transducers described by sets of pairs of the form $(\{m\}_k, r)$, where $m \in \mathcal{M}$ is a message, $k \in K$ is a key and $r$ is a position in $\{m\}_k$.

Similar to the proposition 2.3 we have:

**Proposition 7.2**

*Let $M, S$ be sets of messages, $m$ be a message and* MT *be a set of message transducers. We have the following equivalence:*

$$M{\ll}\mathrm{MT}{\gg}_K S \wedge m{\ll}\mathrm{MT}{\gg}_K S \Leftrightarrow M \cup \{m\}{\ll}\mathrm{MT}{\gg}_K S$$

■

From theorem 2.8 and first equivalence of proposition 7.1 we have the following theorem:

**Theorem 7.3**

*Let $m$ be a message and $E$ a set of messages such that $\mathcal{K} \setminus K^{-1} \subseteq E$. Then, $E \not\vdash m$ iff there exists a set of messages $A \in mc(m)$ s.t. $E{\ll}\emptyset{\gg}_K A$.*

■

Where, mc($m$) is the set of closure messages, defined in the first part of this thesis as follows:

**Definition 2.4 (closed set)** *Let $m \in \mathcal{M}$ a message. We define recursively the set of message sets $mc(m)$, where each message set of $mc(m)$ is a closure for $m$ as follows:*

$$mc(m) = m \uplus \begin{cases} mc(m1) \cup mc(m2) & \text{if } m = (m1, m2) \\ mc(m') \cup mc(k) & \text{if } m = \{m'\}_k \\ \{K^{-1}\} & \text{if } m \text{ is atomic} \end{cases}$$

*where $m \uplus M = \{M_i \cup \{m\} \mid M_i \in M\}$, $M$ set of messages.*

*A set $M$ of messages is called* **closed***, if for any $m \in M$ there exists $M' \in mc(m)$ such that $M' \subseteq M$. In particular $\forall A \in mc(m)$, $A$ is closed.*

**Definition 7.2 (well-formed)**

*Let* MT *be a set of message transducers and $S$ a set of messages. The pair* $(\mathrm{MT}, S)$ *is called* **well-formed***, if the following conditions are satisfied:*

- *$S$ is closed,*
- *for any $(b, r)$ in* MT *if there exists a message transducer $(b_1, r_1) = NT(b, r)$, then $(b_1, r_1)$ is in* MT *or $b$ is in $S$.*

■

We recall that $\mathrm{NT}(b, r)$ is the next (from the top) message transducer in $b$ that dominates $b_{|r}$, if it exists (see definition 2.6).

Intuitively, the first condition ensures that the intruder will always miss at least one part of a composed secret preventing him from deducing it by composition and the second takes into account the ability of the intruder to use encryption in order to obtain a message that can be broken using a message transducer.

It is easy to see that the well-formedness of the pair $(MT, S)$ implies the well-formedness of the set of pairs $(w, S)$ (see definition 2.7), where $w \in \big(\sum_{(b,r)\in MT} (b, r)\big)^*$.

The main property of the predicate $E{\ll}MT{\gg}_K S$ is that it is stable under the intruder's deduction rules. The following theorem is a direct consequence of theorem 2.9 and proposition 7.1.

**Theorem 7.4**

*Let $E$ be a set of messages and* $(\mathrm{MT}, S)$ *be a pair of message transducers and messages. If* $(\mathrm{MT}, S)$ *is well-formed and $E{\ll}\mathrm{MT}{\gg}_K S$ then the messages of $S$ are set-protected despite* MT *in any message $m$ derivable from $E$, that is, $E \vdash m \Rightarrow m{\ll}\mathrm{MT}{\gg}_K S$.*

■

An immediate consequence of theorem 7.4, described in the following corollary, is that under well-formedness of $(MT, S)$, the predicate $E{\ll}MT{\gg}_K S$ is stable with respect to the intruder inference system.

**Corollary 7.5**

*If $E{\ll}\mathrm{MT}{\gg}_K S$ and* $(\mathrm{MT}, S)$ *is well-formed then $E \nvdash S$.* ■

We now come to the computation of a well-formed pair $(MT, S)$ that ensures in addition the stability of $E \ll MT \gg_K S$ with rapport to any rule of the protocol $\Pi = (C, R)$.

**Definition 7.3**

*Let $r = t_1 \rightarrow t_2$ be a rule in $R$. The pair $(\mathrm{MT}, S)$ of a set of message transducers and a set of secrets is* **stable w.r.t. the rule** *$r$, if for every substitution $\sigma$, the property $t_1\sigma \ll \mathrm{MT} \gg_K S$ implies $t_2\sigma \ll \mathrm{MT} \gg_K S$.*

*A pair $(\mathrm{MT}, S)$ is* **stable w.r.t. a set of rules** *$R$ if it is stable with respect to each rule in $R$.*

■

Intuitively, the stability of the pair $(MT, S)$ with respect to a rule $t_1 \rightarrow t_2$ expresses the fact that the message produced by firing the transition $t_1 \rightarrow t_2$ has no effect on the protection of $S$.

Using the theorem 7.4, we can prove by induction the following theorem:

**Theorem 7.6**

*Let $S$ be a set of secrets and $\mathrm{MT}$ be a set of message transducers. If $(\mathrm{MT}, S)$ is well-formed and stable with respect to all rules in $R$, and if in addition $E_0 \ll \mathrm{MT} \gg_K S$ holds for every set of messages $E_0$ that satisfies $C$, then the secrets in $S$ are preserved in any execution of the protocol $\Pi = (C, R)$, denoted $\nvdash_\Pi S$.*

■

**Proof:**

We prove by induction that for any run $E_0 \xrightarrow{r_1} E_1 \cdots E_{n-1} \xrightarrow{r_n} E_n$, where for each $i = 1, \cdots n$, there is a substitution $\sigma_i : \mathcal{X} \rightarrow \mathcal{M}$ such that $E_{i-1} \vdash t_1\sigma$ and $E_i = E_{i-1} \cup \{t_2\sigma\}$, where $t_1 \rightarrow t_2 = r_i$, we have $E_n \nvdash S$.

First, the basic case, $i = 0$ we have $E_0 \ll MT \gg_K S$ then $E_0 \nvdash S$.

The induction step, we prove that if for any run if $E_{i-1} \ll MT \gg_K S$ then, $E_i \ll MT \gg_K S$, for all rules $r = t_1 \rightarrow t_2$ in $R$ and for all $\sigma$ such that $E_{i-1} \vdash t_1\sigma$ and $E_i = E_{i-1} \cup \{t_2\sigma\}$.

$E_{i-1} \ll MT \gg_K S$ and $E_{i-1} \vdash t_1\sigma$ we are in the hypothesis of the proposition 7.4 then $t_1\sigma \ll MT \gg_K S$. $(MT, S)$ is stable with rapport to all rules in $R$ then $t_2\sigma \ll MT \gg_K S$. Using the proposition 7.2 we have $E_{i-1} \cup \{t_2\sigma\} \ll MT \gg_K S$ and so $E_i \ll MT \gg_K S$. Hence $E_{i-1} \nvdash S$ implies $E_i \nvdash S$.

■

## 7.2 Verification algorithm - A semantic version

In this section, we present an algorithm that for a protocol $\Pi = (C, R)$ and a set $S$ of secrets, computes a set $MT'$ of message transducers and a set $S'$ of secrets such that:

- the set of messages $E_0$ initially known by the intruder – defined by the constraint

$C$ – satisfies $E_0 \ll MT' \gg_K S'$,

- $S \subseteq S'$,
- $(MT', S')$ is well-formed, and
- $(MT', S')$ is stable with respect to $R$.

First, we develop a semantic version of the algorithm in which we do not consider questions related to representing infinite sets of message transducers. Then, we define a symbolic representation for message transducers and we give the symbolic version of the algorithm.

Our algorithm takes as input: a set of rules $R$, a set of safe keys $K$, a closed set of secret messages $S$ and an empty set of message transducers $MT$ and computes a pair $(MT', S')$ which is well-formed, and stable with respect to the rules of the protocol. It is described in figure 7.1.

The algorithm uses a function *Closure* that is applied to the set of messages $S$ any time when we add a new messages to the set of secret messages. This function associates to a set of messages one of its closure against composition, following definition 2.4. We choose, for the new messages added to $S$ the closure that contains one atomic message which is already in $S$. If such a closure does not exist we take one at random. Intuitively, if such a closure does not exist that means no subterm of the new message have been secret before and then the new message can not be secret.

If the algorithm terminates, it returns an augmented set of secrets $S'$ and an augmented set of message transducers $MT'$ such that $(MT', S')$ is well-formed, and stable with rapport to all rules of the protocol.

We now explain intuitively the main idea of the algorithm.

Let us take a rule $t_p \rightarrow t_c$ in $R$, a substitution $\sigma : \mathcal{X} \rightarrow \mathcal{M}$ such that a secret $s$ is set-protected in $t_p\sigma$, the premise of the instantiated rule, despite $MT$. If the secret $s$ is not protected in $t_c\sigma$, the conclusion of the instantiated rule, then there is at least one submessage of $t_p\sigma$ of the form $\{m\}_k$, with $k \in K$, in which the secret $s$ appears and which does not protect it anymore, so it must be added to the set of message transducers. Indeed, the intruder does not need the inverse of the keys in $K$ to get the secret: it will be involuntary revealed by a principal who plays the rule $t_p\sigma \rightarrow t_c\sigma$.

Let us take for instance a protocol with $\{x, I\}_{pbk(A)} \rightarrow \{x\}_{pbk(I)}$ as a rule of principal $A$. The principal $A$ will respond $\{s\}_{pbk(I)}$ on reception of the message $\{s, I\}_{pbk(A)}$. Thus involuntary decrypting the secret for the intruder. So, the message $\{s, I\}_{pbk(A)}$ is a particular case where the key $pbk(A)$ does not protect the secret and $(\{s, I\}_{pbk(A)}, 01)$ must be added to the set of the message transducers $MT$.

The case 2 in the algorithm considers the situation when a secret $s$ is not protected in $t_c\sigma$ and it does not appear in $t_p\sigma$. In this case, the apparently harmless premise is as compromising as the secret, and so, $t_p\sigma$ must be added to the set of secrets.

The following proposition summarizes the properties of the algorithm.

**Proposition 7.7**

*If the algorithm of Figure 7.1 applied to* $(R, S, K, \mathrm{MT})$ *terminates, it returns* $S'$ *and* $\mathrm{MT}'$ *that satisfy the following conditions:*

1. $(\mathrm{MT}', S')$ *is well-formed,*
2. $(\mathrm{MT}', S')$ *is stable w.r.t.* $R$*, and*
3. $S \subseteq S'$

■

**Proof:**

The well-formed property of $(MT', S')$ derives directly from the operations made in the algorithm. First, the set of secrets $S'$ is each time closed. Second, any time a dangerous premise with respect to a secret $s$ is found we add to $MT'$, all message transducers obtained by its subterms of the form $\{m\}_k$, with $k \in K$ that dominates the secret $s$ and the related positions. Hence, that ensures the second condition of the well-formedness.

The stability. If the algorithm reaches a fixpoint $(MT', S')$ then, the **until** condition of **repeat** termination will be reached. That is, all rule in $R$ produce dangerous substitution $DS$ which generates $newMT$ and $newS$ which are already in $MT'$ respectively $S'$.

Since we start with the set $S' = S$, and then the algorithm only augments it, the last condition, $S \subseteq S'$ is obviously satisfied.

■

A direct consequence of the this proposition and the theorem 7.6 is the following corollary.

**Corollary 7.8**

*If the algorithm of figure 7.1 terminates with* $(\mathrm{MT}', S')$ *as result, and each set of messages* $E_0$ *that satisfies* $C(E_0)$ *also satisfies* $E_0 \ll \mathrm{MT}' \gg_K S'$*, we can conclude* $\not\vdash_{\Pi} S'$*, and hence,* $\not\vdash_{\Pi} S$*.*

■

**input**: $R$, $S$, $K$ and $MT = \emptyset$

**output**: $MT'$, $S'$ such that $(MT', S')$ is well-formed and stable w.r.t. $R$.

$MT' := MT$ ; $S' := S$ ;

**repeat**

$MT_c := MT'$ ; $S_c := S'$ ;

**for each** $t_p \rightarrow t_c \in R$ **do**

**for each** $r \in dom(t_c)$ *s. t.* $t_{c|_r} \in \mathcal{X} \cup S$ **do**

*(* compute all Dangerous Substitutions of rule* $t_p \rightarrow t_c$ *where *)*
*(* a secret is not kept in the conclusion for the position* $r$ **)*

$DS := \{\sigma : \mathcal{X} \rightarrow \mathcal{M} \mid \exists s \in S s.t. \neg(t_c\sigma \ll MT' \gg_K s) \wedge \neg(t_{c|_r}\sigma \ll MT' \gg_K s)\}$ ;

*(* compute the corresponding Dangerous Premises *)*

$DP := \{t_p\sigma \mid \sigma \in DS\}$ ;

*(* update the sets* $S'$ *and* MT$'$ *according to the dangerous premises: *)*
*(* case 1 add message transducers to* MT$'$ *if* $t_{c|_r} \in_c t_p$ **)*

**for each** $m \in DP$ **do**

*(* new message transducer are pairs constructed from submessage *)*
*(* of* $m$ *of the form* $\{m'\}_k$, $k \in K$ *and positions of* $t_{c|_r}$ *in them *)*

$newMT := \{(m_{|_q}, q^{-1}r') \mid \exists k \in K,\ m_{|_q} = \{m_{|_{q \cdot 0}}\}_k \wedge$
$\exists r'$ critical position s. t. $t_{p|_{r'}} = t_{c|_r} \wedge q \prec r'\}$

*(* update the set of message transducers* MT$'$ **)*

$MT' := MT' \cup newMT$ ;

**od**

*(* case 2 adds to the secrets all dangerous premises if* $t_{c|_r} \notin_c t_p$ **)*

$newS := \{m \mid m \in DP\}$;
$S' := S' \cup newS$

*(* compute the closure that adds to* $S'$ *subparts of *)*
*(* each compound secret of* $S'$ **)*

$S' := Closure(S')$ ;

**od**

**od**

**until** $(MT', S') = (MT_c, S_c)$

Figure 7.1: The semantic version of the verification algorithm

## 7.3 Verification algorithm - A symbolic version

### 7.3.1 Symbolic representation

In the previous section we have presented our verification algorithm without considering questions related to the representation of infinite sets of message transducers. To do so, we introduce **pattern terms** defined by the following grammar:

$$pt \quad ::= \quad n \mid p \mid k \mid \quad x \mid \mathbf{pair}(pt_1, pt_2) \mid \mathbf{encr}(pt, k) \mid \mathit{Sup}(pt)$$

where $n \in \mathcal{N}$, $p \in \mathcal{P}$, $k \in \mathcal{K}$, and $x \in \mathcal{X}$. The set of patterns is denoted by $\mathcal{PT}(\mathcal{X}, \mathcal{F})$. Notice that every term in $\mathcal{T}(\mathcal{X}, \mathcal{F})$ is also a pattern in $\mathcal{PT}(\mathcal{X}, \mathcal{F})$. The difference between the two is that patterns make use of the special $Sup$ function symbol.

We extend naturally the semantics of a message transducer from messages to terms as follows: Given a term $b \in \mathcal{T}(\mathcal{X}, \mathcal{F})$, and a critical position $p$, we denote by $[\![(b, p)]\!]$ the set of message transducers associated to all instances of $b$ and the position $p$.

$$[\![(b, p)]\!] = \{(b\sigma, p) \mid \sigma : \mathcal{X} \to \mathcal{M}\}.$$

Intuitively, as can be seen from the following definition, $Sup(t)$ represents all terms containing the term $t$ as a sub-term. For instance, the terms $A$, $\mathbf{pair}(x, A)$, $\{A\}_k$, $\cdots$ all belong to $[\![Sup(A)]\!]$. We define the semantics of patterns as follows:

**Definition 7.4**

*Given a pattern* pt*, the set of all corresponding terms is denoted by* $[\![\mathrm{pt}]\!]$*. It is defined as follows:*

$$\begin{array}{lcl}
[\![\mathrm{pt}]\!] & = & \{\mathrm{pt}\} \qquad \textit{if pt is a constant or a variable} \\
[\![\mathbf{pair}(\mathrm{pt}_1, \mathrm{pt}_2)]\!] & = & \{\mathbf{pair}(t_1, t_2) \mid t_1 \in [\![\mathrm{pt}_1]\!], t_2 \in [\![\mathrm{pt}_2]\!]\} \\
[\![\mathbf{encr}(\mathrm{pt}_1, k)]\!] & = & \{\mathbf{encr}(t_1, k) \mid t_1 \in [\![\mathrm{pt}_1]\!]\} \\
[\![\mathrm{Sup}(\mathrm{pt})]\!] & = & \{t \mid \exists \textit{ a position } p \textit{ in } t \textit{ s.t. } t_{|_p} \in [\![\mathrm{pt}]\!]\}
\end{array}$$

■

**Example 7.1**

$\{A, (x, B)\}_k; \{A, \{(x, B), y\}_{k'}\}_k; \{A, (B, (y, (A, B)))\}_k$ *are elements of* $[\![\{A, \mathrm{Sup}(x, B)\}_k]\!]$.

■

Now we are ready to define the semantic of pattern transducers:

**Definition 7.5**

*Given a pattern* pt*, and a critical position* $p$*, we denote by* $[\![(\mathrm{pt}, p)]\!]$ *the set of term transducers associated to the* pattern transducer $(\mathrm{pt}, p)$*. We overload the function* $[\![]\!]$ *for which the meaning is clear from its argument. For pattern transducers, the function* $[\![]\!]$ *is defined as follows:*

$$
\begin{array}{lcl}
[\![(\mathrm{pt}, p)]\!] & = & \{(\mathrm{pt}, p)\} \qquad \textit{if pt is a constant or a variable} \\
[\![(\mathbf{pair}(\mathrm{pt}_1, \mathrm{pt}_2), p)]\!] & = & \{(\mathbf{pair}(t_1, t_2), \epsilon) \mid p = \epsilon,\ t_1 \in [\![\mathrm{pt}_1]\!], t_2 \in [\![\mathrm{pt}_2]\!]\} \\
 & \cup & \{(\mathbf{pair}(t_1, t_2), 0.q) \mid p = 0.p',\ (t_1, q) \in [\![(\mathrm{pt}_1, p')]\!],\ t_2 \in [\![\mathrm{pt}_2]\!]\} \\
 & \cup & \{(\mathbf{pair}(t_1, t_2), 1.q) \mid p = 1.p',\ t_1 \in [\![\mathrm{pt}_1]\!],\ (t_2, q) \in [\![(\mathrm{pt}_2, p')]\!]\} \\
[\![(\mathbf{encr}(\mathrm{pt}_1, k), p)]\!] & = & \{(\mathbf{encr}(t_1, k), \epsilon) \mid p = \epsilon,\ t_1 \in [\![\mathrm{pt}_1]\!]\} \\
 & \cup & \{(\mathbf{encr}(t_1, k), 0.q) \mid p = 0.p', (t_1, q) \in [\![(\mathrm{pt}_1, p')]\!]\} \\
[\![(\mathrm{Sup}(\mathrm{pt}), p)]\!] & = & \{(t, \epsilon) \mid p = \epsilon,\ t \in [\![\mathrm{Sup}(\mathrm{pt})]\!]\} \\
 & \cup & \{(t, q.r) \mid p = 0.p',\ (t_{|_q}, r) \in [\![(\mathrm{pt}, p')]\!]\}
\end{array}
$$

■

**Example 7.2**

*The pattern transducer* $\big(\mathrm{Sup}(A, x), 01\big)$ *denotes all term transducer* $(b, p)$ *that contain* $(A, x)$ *as a sub-term of* $b$ *and where* $p$ *corresponds to the position of* $x$*.*

*The computation of the term transducers corresponding to pattern transducer* $\big(\mathrm{Sup}(A, x), 01\big)$ *goes through the step* $[\![\big((A, x), 1\big)]\!] = \{\big(\big((A, x), 1\big)\}$ *and ends with the set* $[\![\big(\mathrm{Sup}(A, x), 01\big)]\!] = \{(t, q.1) \mid t_{|_q} = (A, x)\}$*. This set contains for instance the terms* $\big(((A, x), B), 01\big)$*,* $\big((B, (A, x)), 11\big)$*,* $\big((A, x), 1\big)$*,* $\big(\{B, (A, x)\}_k, 111\big)$*.*

■

Using the function $[\![]\!]$ we can shift from pattern transducers to their equivalent representation as sets of term transducers of the form $(b, p)$, where $b \in \mathcal{T}(\mathcal{X}, \mathcal{F})$ is now a term. Based on this remark, we present the algorithm on terms and we explain how it extends to patterns. In the sequel, when there is no need to distinguish between terms and patterns, we use patterns.

Based on the symbolic representation, the potentialy infinite set $MT$ of message transducers is represented by a finite set of pattern transducers $PT$. Formally, we have the following:

**Definition 7.6 (symbolic representation)**

*A* **symbolic representation** $SR$ *is a pair* $(\mathrm{PT}, S)$*, where*

- PT *is a finite set of pattern transducers that represents the set of message transducers* MT
- $S$ *is a finite set of terms that represents the set of secrets.*

■

### 7.3.2 The patterns computation using dangerous substitution

In this section we present the symbolic computation for one rule of the protocol. Starting from a set of pattern transducers $PT$, a set of secrets $S$ and a rule $r = t_p \rightarrow t_c$ we compute the sets of pattern transducers $newPT$ and secrets $newS$ which are unsafe because of the rule $r$ and which have to be added to $PT$ respectively $S$.

First, we present the algorithm that computes the dangerous substitutions $DS$ induced by a rule $t_p \rightarrow t_c$, and a position $p$ and then we show how we compute the sets $newPT$ respectivly $newS$ in the case when the substitution $DS$ is not empty. These two steps are done for each position $p \in dom(t_c)$ such that $t_{c|_p}$ unify with a secret of $S$.

In order to define a dangerous substitution we have to define when two pattern transducers unify.

**Definition 7.7 (pattern transducer unification)**

*Let $pt = (t, p)$ and $pt' = (t', p')$ be two pattern transducers. We say that they* **unify** *if there is a substitution $\sigma : \mathcal{X} \rightarrow \mathcal{T}(\mathcal{X}, \mathcal{F})$ such that $t$ and $t'$ unify by $\sigma$ and $(t\sigma)_{|_p} \preceq (t'\sigma)_{|_{p'}}$ or $(t'\sigma)_{|_{p'}} \preceq (t\sigma)_{|_p}$. We write, also, $\sigma(t, p) = \sigma(t', p')$.*

■

Let $K$ be the fixed set of keys, called safe keys and $PT$ the set of pattern transducers. We call **protecting positions** of $p$ in $t$ all positions which are prefixes of $p$ and such that the subterms of $t$ at these positions are terms encrypted with safe keys.

**Definition 7.8 (dangerous substitution)**

*A substitution $\sigma : \mathcal{X} \rightarrow \mathcal{T}(\mathcal{X}, \mathcal{F})$ is* **dangerous** *for a term $t_c$ and a position $p$ if for all subterms of $t_c$ at protecting positions there is a pattern transducer which cancels the position $p$ restricted to that subterm.*

*Formally, for every position $q \prec p$, for which $\exists k \in K$ such that $t_{c|_q} = \{t_{c|_{q \cdot 0}}\}_k$ we have $(t_{c|_q}, q^{-1}p)$ unifies by $\sigma$ with a pattern transducer of* PT.

■

Let us give an example to illustrate this defintion:

**Example 7.3**

*Let $(t_c, p) = (\{(y, A)\}_k, 00)$ and the term transducer $t = (\{((x, B), A)\}_k, 000)$. The dangerous substitution is $\sigma = [y \rightarrow (x, B)]$ and the relative position is $0$, because in $t$ only the position $p \cdot 0$ is not protected in $t_c\sigma$.*

*Let now $(t_c, p) = (\{((x, B), A)\}_k, 000)$ and the term transducer $t = (\{(y, A)\}_k, 00)$. The dangerous substitution is $\sigma = [y \rightarrow (x, B)]$ and the relative position is $\varepsilon$, as $00 \prec 000$ and in $t$ the position $p$ is not protected in $t_c\sigma$.*

■

We define below the function $\Phi$ that computes all the unifiers between the pattern transducers of $PT$ and $(t_c, p)$ that cancel each protecting position of $t_{c|_p}$ and its related

position $r$. For a substitution $\sigma$, $r$ is a relative position to $p$ with respect to the position which is not protected in $t_c\sigma$.

Let $Pos$ be the set of protecting positions $p_i$ above $p$, i.e. for each $p_i \in Pos$ there is $k \in K$ such that $t_{c|_{p_i}} = \{t_{c|_{p_i \cdot 0}}\}_k$. Formally, the dangerous substitutions are the unifiers $\sigma$ that satisfy:

$$\bigwedge_{p_i \in Pos} \sigma(t_{c|_{p_i}}, p_i^{-1}p) = \sigma(b_i, q_i)$$

where $(b_i, q_i) \in PT$.

The parameters of $\Phi$ are: the term $t_c$ and the set of pattern transducers $PT$, the set $Pos$ of protecting positions and a set of pairs of substitutions and theirs relative positions $DS$. Then, it takes in turn each protecting position $p_i$ and if it is possible, it completes the substitutions of $DS$ in order to cancel the current position by a pattern transducer $(b, q)$ of $PT$, and completes the relative position. Formally, the function $\Phi$ is recursively defined as follows:

$$\Phi(t_c,\ PT,\ Pos,\ DS) = \begin{cases} DS & \text{if } Pos = \emptyset \\ \Phi\big(t_c,\ PT,\ Pos \setminus \{p_i\},\ \bigcup\limits_{(\sigma_j, r_j) \in DS} \{(\sigma_j \oplus \sigma_1^{i,j}, r_j \cdot r'_1) \cdots, \\ \quad (\sigma_j \oplus \sigma_{n_{i,j}}^{i,j}, r_j \cdot r'_{n_{i,j}})\}\big), & \text{if } p_i \in Pos \wedge n_{i,j} > 0 \\ \emptyset & \text{otherwise} \end{cases}$$

where the $(\sigma_k^{i,j})_{k=1,n_{i,j}}$ in the fourth arguments are the unifiers resulting from the unification of $((t_c\sigma_j)_{|_{p_i}}, p_i^{-1}p \cdot r_j)$ with some pattern transducers $(b, q)$ of $PT$ and $r'_k$ is the relative position induced by $\sigma_k^{i,j}$ for $k = 1, n_{i,j}$. More exactly if $(t_c(\sigma_j \oplus \sigma_k^{i,j}))_{|_{p_i \cdot r_j}} \preceq (b\sigma_k^{i,j})_{|_q}$ then $r'_k = \varepsilon$ else $(t_c(\sigma_j \oplus \sigma_k^{i,j}))_{|_{p_i \cdot r_j \cdot r'_k}} = (b\sigma_k^{i,j})_{|_q}$. Also, we notice here that we have to keep in $\sigma_k^{i,j}$ only the substitutions for the variables which appear in $t_c$. Also, since there is no relation between $\sigma_j$ and $\sigma_k^{i,j}$ w.r.t. the variables that come from terms of $PT$ we have to rename these variables with fresh ones, each time they appear in the right side of a substitution in $\sigma_k^{i,j}$. For the sake of clarity this renaming does not appear in examples only if it is necessary.

We denote by $\Phi_{pt}(t_c,\ PT\cdots)$, where $pt$ is a pattern transducer of $PT$, the subset of substitution $\{\sigma_j\}$ returned by $\Phi(t_c, PT\cdots)$ where at least one time a substitution $\sigma_k^{i,j}$ is obtained using $pt$. That is, the pattern transducer $pt$ has been useful to cancel one of protecting position of $t_{c|_p}$.

In the case of terms, we use the standard *most general unifier*; and for patterns (with *Sup* function), we define a unification algorithm presented in annex A.

**Example 7.4**

*We illustrate the computation of dangerous substitutions on the set of pattern transducers* PT = { $(\{(I, x)\}_{k_b}, 01)$, $(\{((a, y), z)\}_{k_b}, 01)$ }*, the set of key* $K = \{k_a, k_b\}$ *and a rule* $t_p \to t_c$ *given in figure 7.2.*

*We consider the conclusion of the rule. The first step consists of looking for all the*

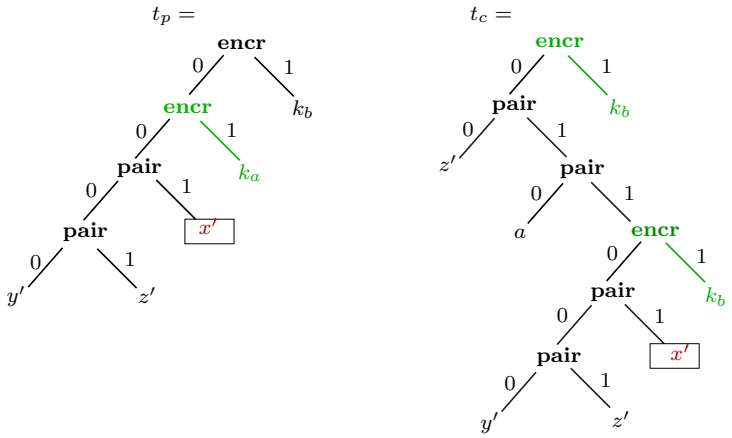


Figure 7.2: Illustration of computing dangerous substitutions

*critical positions in the conclusion where a secret or a variable appears. We find* $x'$ *at position* 01101, $y'$ *at position* 011000 *and* $z'$ *at positions* 00, 011001 *in the term* $t_c$.

*Let take the position* $p = 01101$ *of* $x'$, *we look for the positions above it that may protect it. We found exactly two protecting positions:* $p_1 = \epsilon$ *and* $p_2 = 011$.

*Then, the function* $\Phi$ *looks for all substitutions that unify some pattern transducers of* PT *with the terms at the protecting positions* $p_1$ *and* $p_2$ *and the restricted respective positions of* $p$.

*Starting with position* $p_1 = \epsilon$, *it unifies* $(t_{c|_\epsilon}, p)$ *with the pattern transducer* $(\{(I, x)\}_{k_b}, 01)$ *we have* $01 \prec p$, *and the unifier* $\sigma' = [z' \to I, x \to t_{c|_{01}}]$. *The relative position is* $\varepsilon$. *This cancels the top most protection.*

*Then, the function* $\Phi$ *attempts to complete the substitution* $\sigma'$ *such that it also cancels the protection at position* $p_2 = 011$.

*To do so, it tries to unify* $(\sigma'(t_c)_{|_{p_2}} = \{((y', I), x')\}_{k_b}, p_2^{-1}p = 01)$ *with some pattern transducers of* PT *and succeeds with the pattern transducer* $(\{((a, y), z)\}_{k_b}, 01)$. *We have* $01 = 01$ *and the unifier* $\sigma'' = [y' \to a, y \to I, x' \to z]$. *The relative position is, again,* $\varepsilon$.

*The two unifiers are then composed and restricted to the domain var* $(t_c)$ *resulting the substitution* $\sigma = (\sigma' \oplus \sigma'')_{var\ (t_c)} = [y' \to a, z' \to I]$.

*Pursuing the computation does not provide other substitutions and finally* $\Phi$ *returns for the position* $p$ *of* $t_c$ *the set of dangerous substitutions and relative positions* $\{(\sigma, \varepsilon)\}$.

■

If the set of dangerous substitution $DS$ is empty then this rule does not generate new pattern transducers or new secrets otherwise we compute the set of dangerous premises, $DP$, which generate new pattern transducers or new secrets as in the semantic version of the algorithm. The set of *dangerous premises* is: $DP = \{\sigma(t_p) \mid \sigma \in DS\}$.

If there exists a position $q$ such that $t_{c|_p} = t_{p|_q}$ then for every pattern $t$ of $DP$ we construct a set of new pattern transducers that consists of pairs of sub-terms of $t$, that are encrypted by keys from $K$, and positions restricted to sub-terms of $q$. Formally,

$$newPT = \{(t_{|_r}, r^{-1}q) \mid t \in DP, \exists k \in K, t_{|_r} = \{t_{|_{r\cdot 0}}\}_k \wedge t_{c|_p} = t_{p|_q} \wedge r \prec q\}.$$

Otherwise, if such a $q$ does not exist then the set of dangerous premises represents the set of new secrets, $newS := \{t \mid t \in DP\}$.

**Example 7.5**

*We continue the example 7.4 to illustrate the computation of new pattern transducers. We now look at the premise of the rule to compute the new pattern transducers induced by the set of dangerous substitutions and relative positions computes by* $\Phi$. *We have* $(\sigma = [y' \to a, z' \to I], r = \varepsilon)$ *the only element returns by* $\Phi$.

*The variable* $x'$ *appears in* $t_p$ *at the position* $q = 001$ *and it is protected by the key* $k_b$ *at the position* $r_1 = \epsilon$ *and by the key* $k_a$ *at the position* $r_2 = 0$.

*However, the dangerous substitution* $\sigma$ *tells that these protections will not work in case where* $y'$ *is* $a$ *and* $z'$ *is* $I$. *Consequently, we increase the set of pattern transducers* PT *by adding these particular cases. In our symbolic representation, this comes out to add* $(\sigma(t_p) = \{\{((a, I), x')\}_{k_a}\}_{k_b}, r_1^{-1}q \cdot r = 001)$ *and* $(\sigma(t_p)_{|_0} = \{((a, I), x')\}_{k_a}, r_2^{-1}q \cdot r = 01)$ *to the set of pattern transducers* PT.

■

### 7.3.3 Symbolic verification algorithm

The symbolic verification algorithm takes as input a set of rules $R$, a closed set of secret terms $S$, a set of key $K$ and an empty set of pattern transducers. It computes new pairs of pattern transducers and secrets $(PT', S')$ until it becomes stable with respect to all rules in $R$.

Let us now sketch the main steps of the symbolic algorithm:

1. Input: $R$, $S$, $K$ and $PT = \emptyset$ pattern transducers.
2. We initialize two global variables $S' = S$ and $PT' = \emptyset$.
3. The **explore** function completes the sets $PT'$ and $S'$ by depth-first exploration, figure 7.3:
4. The **one_Step** function takes two parameters, a rule $r = t_p \to t_c$ and a sequence of pattern transducers *path*. It returns the sets $newPT$ and $newS$ computed as defined in the section 7.3.2.

   For each position $p$ of $t_c$ that corresponds to a variable or a secret this function computes the set of dangerous premises $DS$. If the second parameter is the empty sequence then it computes the set of dangerous substitution (function $\Phi$) else, if

```
function explore (path : (rule × pattern transducer)*)
for each r ∈ R do
    (newPT, newS) := one_Step (r, path);
    S' := closure (newS, S');
    if newPT ⊆ PT' then
        return
    else
        for each pt ∈ newPT do
            if pt ∉ PT' then
                PT' := PT' ∪ {pt};
                explore ((r, pt)@path)
            fi
        od
    fi
od
```

Figure 7.3: The main function of the symbolic algorithm

the sequence is not empty then it computes the set of dangerous substitution in which the first pattern transducer of the sequence; let's say $pt_0$, has been implied (function $\Phi_{pt_0}$).

5. The **closure** function, returns a set of closed terms by adding to $S'$ all elements of $newS$ along with some terms such that the $newS \cup S'$ is closed. Where we extend the notion of closure of sets of messages to sets of terms considering the case of a variable $x$: $\mathrm{wc}(x) = x \uplus \{K^{-1}\}$.

## 7.4 Termination

### 7.4.1 Why the algorithm does not terminate

Let us consider the following rule from the session $(A, A)$ of Needham-Schroeder-Lowe protocol presented in section 6.3:

$$r = \{(A, (N_1^{AA}, y))\}_{pbk(A)} \rightarrow \{y\}_{pbk(A)}.$$

and the pattern transducer $\{(I, x)\}_{pbk(A)}, 01)$. A dangerous substitution is $\sigma = [y \rightarrow (I, x)]$ and the related position is $r = 1$. Then the pattern transducer $(\{(A, (N_1^{AA}, (I, x)))\}_{pbk(A)}, 011 \cdot 1)$ is generated. For this one and the same rule $r$ we have the dangerous substitution $\sigma' = [y \rightarrow (A, (N_1^{AA}, (I, x)))]$ and the related position is $r = 111$. Hence, the pattern transducer $(\{(A, (N_1^{AA}, (A, (N_1^{AA}, (I, x)))))\}_{pbk(A)}, 011{\cdot}111)$ is generated and so on.

Thus, a naive application of our symbolic algorithm will not terminate.

### 7.4.2 Enforcing termination

We present a technique that makes the depth-first implementation of the symbolic verification algorithm terminate more often, at the price of a safe approximation of the results.

We provide a widening operator that forces termination in the presence of rules that can loop and produce an infinite set of growing pattern transducers. The additional operator *Sup* is used to encompass all these growing patterns in a finite pattern.

**Definition 7.9**

*A sequence* $(t_i, p_i)_{i\geq 0}$ *of pattern transducers is called* **increasing at a sequence** $(q_i)_{i\geq 0}$ *of positions, if the following conditions are satisfied for every* $i \geq 0$*:*

1. $q_i$ *is a prefix of* $p_i$ *such that* $q_i^{-1}p_i$ *is a constant,*
2. $q_i \in \mathrm{dom}(t_i)$ *and* $q_i \preceq q_{i+1}$*,*
3. $t_i[t_{0|_{q_0}}/q_{i-1}] = t_{i-1}$*, for* $i > 0$*.*

*For an intuitive idea see the figure 7.4.*

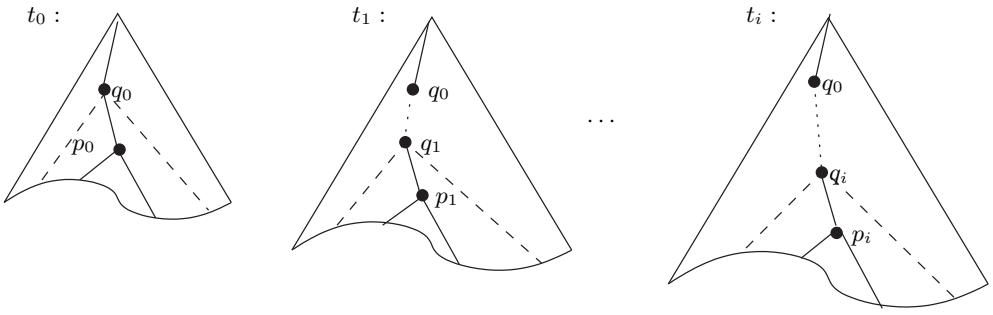


Figure 7.4: Increasing sequence definition

■

Let us consider an example to clarify these definitions.

**Example 7.6**

*Consider the sequence* $(\{\theta^i(I,x)\}_{\mathrm{pbk}(A)}, p_i)_{i\geq 0}$*, where*
$\theta(z) = (A, (N_1^{AA}, z))$ *and* $p_i = 0 \cdot (11)^i \cdot 1$*.*

*This sequence of pattern transducers is obtained by the rule* $r$*, given in section 7.4.1 and starting from the pattern transducer* $(\{(I,x)\}_{\mathrm{pbk}(A)}, 01)$*.*

*The first three terms of the sequence, described in section 7.4.1 are:*

$$(\{\theta^0(I,x)\}_{\mathrm{pbk}(A)} = \{(I,x)\}_{\mathrm{pbk}(A)}, 01)$$
$$(\{\theta^1(I,x)\}_{\mathrm{pbk}(A)} = \{(A,(N_1^{AA},(I,x)))\}_{\mathrm{pbk}(A)}, 0111) \text{ and}$$
$$(\{\theta^2(I,x)\}_{\mathrm{pbk}(A)} = \{(A,(N_1^{AA},(A,(N_1^{AA},(I,x)))))\}_{\mathrm{pbk}(A)}, 011111).$$

*This sequence is increasing at* $(q_i = 0 \cdot (11)^i)_{i\geq 0}$. *Indeed,*

$q_i^{-1}p_i = 1$ *and*

$\{\theta^i(I,x)\}_{\mathrm{pbk}(A)}[(I,x)/q_{i-1}] = \{\theta^i(I,x)\}_{\mathrm{pbk}(A)}$

*for every* $i \geq 0$.

■

As direct consequence of the definition 7.9 we have the following property:

**Proposition 7.9**

*Let* $(t_i,p_i)_{i\geq 0}$ *be an increasing sequence of pattern transducers at a sequence* $(q_i)_{i\geq 0}$ *and let* $j,k$ *be natural numbers such that* $k \geq j$. *Then* $t_k[z/q_j] = t_j[z/q_j]$, *where* $z$ *is a fresh variable.*

■

**Proof:**

We will argue by induction. This result is trivial for $k = j$.

Now assume that for $j \leq i < l$ it holds that $t_i[z/q_j] = t_j[z/q_j]$ we prove that $t_l[z/q_j] = t_j[z/q_j]$.

From the definition 7.9 we have $t_l[t_{0|_{q_0}}/q_{l-1}] = t_{l-1}$ that implies

$(t_l[t_{0|_{q_0}}/q_{l-1}])[z/q_{l-1}] = t_{l-1}[z/q_{l-1}]$

$t_l[z/q_{l-1}] = t_{l-1}[z/q_{l-1}]$

but $q_j \preceq q_{l-1}$ hence $t_l[z/q_j] = t_{l-1}[z/q_j]$

and using the induction hypothesis we have: $t_l[z/q_j] = t_j[z/q_j]$.

Hence, $\forall j \geq 0 \forall k \geq j t_k[z/q_j] = t_j[z/q_j]$.

■

The idea for enforcing termination is to approximate an increasing sequence with a finite set of pattern transducers that represents at least all elements of the initial sequence. We express that formally by the following proposition:

**Proposition 7.10**

*Let* $(t_i,p_i)_{i\geq 0}$ *be increasing at* $(q_i)_{i\geq 0}$. *Then,*

$$\bigcup_{i\geq 0}[\![(t_i,p_i)]\!] \subseteq \bigcup_{i<j}[\![(t_i,p_i)]\!] \cup [\![t_j[\mathrm{Sup}(t_{j|_{q_j}})/q_j], q_j \cdot 0 \cdot q_j^{-1}p_j]\!], \text{ for every } j \geq 0.$$

■

**Proof:**

Since $q_j^{-1}p_j$ is a the constant which let to the same sub-term of $t_{j_{|q_j}} = t_{0_{|q_0}}$, the positions correspondence is trivial. Hence, it is enough to prove that $[\![t_k]\!] \subseteq [\![t_j[Sup(t_{j_{|q_j}})/q_j]]\!]$.

From the proposition 7.9 we have: $t_k[z/q_j] = t_j[z/q_j]$ $(*)$

also, from the definition 7.9 we have: $t_{k_{|q_k}} = t_{j_{|q_j}} = t_{0_{|q_0}}$ $(**)$

$q_j \preceq q_k$ implies $t_{k_{|q_k}} \preceq t_{k_{|q_j}}$ and using $(**)$ we obtain: $t_{j_{|q_j}} \preceq t_{k_{|q_j}}$ hence,
$[\![t_{k_{|q_j}}]\!] \subseteq [\![Sup(t_{j_{|q_j}})]\!]$ finally, using $(*)$ we obtain:
$[\![t_k]\!] \subseteq [\![t_j[Sup(t_{j_{|q_j}})/q_j]]\!]$.

■

**Example 7.7**

*Consider again our example 7.6.*

*Then, if we choose $j = 1$, we obtain a set consisting of the two pattern $(\{(I,x)\}_{\mathrm{pbk}(A)}, 01)$ and $(\{(A,(N_1^{AA}, \mathrm{Sup}(I,x)))\}_{\mathrm{pbk}(A)}, 01101)$ which approximates the whole sequence $\left(\{\theta^i(I,x)\}_{\mathrm{pbk}(A)}, p_i\right)_{i\geq 0}$.*

■

**Definition 7.10**

*We define a widening operator $\nabla$ as follows:*
$\nabla : \mathcal{PT} \times \{0,1\}^* \to \mathcal{PT}$,
$\nabla((t,p),q) = (t[\mathrm{Sup}(t_{|_q})/q], q \cdot 0 \cdot q^{-1}p)$

■

The main function of our symbolic algorithm using widening is given in figure 7.5.

The **exists_increasing** function returns true if a sequence of rules is repeated $M$ times at the end of the *path* and if, for each repeated rule, the corresponding pattern transducers form an increasing sequence. $M$ is a parameter of our algorithm and must be strictly greater than 1.

Formally, let *path* be $(r_1, pt_1) \cdots (r_n, tp_n)$. The exists_increasing function returns true if $\exists k > 0$, the length of the repetition, such that starting from $b = n - k \cdot M$ we have:

- $\forall i = 1, k \ \forall j = 1, M-1 \ \ r_{b+i} = r_{b+j\cdot k+i}$ and
- $\forall i = 1, k \ \exists (q_j^i)_{j=0,M-1}$ such that $(pt_{b+j\cdot k+i}, q_j^i)_{j=0,M-1}$ is an increasing sequence.

The **widening** function replaces the last occurrence of the repeated sequence in the *path*, with its approximation by applying the widening operator $\nabla$ on its pattern transducers. The approximated *path* becomes:

$$(r_1, pt_1), \cdots (r_{n-k}, pt_{n-k}), (r_{n-k+1}, \nabla(pt_{n-k+1}, q^1_{M-1})) \cdots (r_{n-k+k}, \nabla(pt_{n-k+k}, q^k_{M-1}))$$

```
function explore (path : (rule × pattern transducer)*)
if exists_increasing (path, M) then
    path' := widening (path);
    explore (path')
else
    for each r ∈ R do
        (newPT, newS) := one_step (r, path);
        S' := closure (newS, S');
        if newPT ⊂ PT' then
            return
        else
            for each pt ∈ newPT do
                if pt ∉ PT' then
                    PT' := PT' ∪ {pt};
                    explore ((r, pt)@path)
                fi
            od
        fi
    od
fi
```

Figure 7.5: The main function of the symbolic algorithm with widening

The heuristics can be resumed as follows. First, detect the repeated applications of the same (sequence of) rule which leads to an increasing sequence of pattern transducers. Second, approximate this infinite increasing sequence by its finite "limit" obtained by widening i.e, application of some *Sup* operators.

At this time, we do not know if this heuristics is *sufficient* to guarantee the termination of the symbolic depth-first search algorithm. We do not have yet a formal argument neither proving or disproving this property.

Nevertheless, this heuristics has been proven very efficient in practice. In fact, the algorithm with widening terminates on all protocols tested, in reasonable time, whereas without widening, it terminates on very few of them. Concrete results and running time are presented in section 8.3.

# Chapter 8

# Hermes

In this chapter we describe HERMES, a tool for automated cryptographic protocol verification which implements the verification techniques introduced in the previous chapters. HERMES is part of a verification toolbox for cryptographic protocols developed in the French national project EVA.

The EVA toolbox architecture is illustrated in figure 8.1. The entry point in the tool-box is LAEVA, a high level specification language for security protocols and their properties. Using the automatic translator EVATRANS [GL02a], LAEVA specifications are transformed into an intermediate representation CPL. This intermediate representation is actually the common input of three automatic verification tools: HERMES [BLP03a, BLP02a, BLP03b], SECURIFY [CMR01, Cor02] and CPV 1 & 2 [GL00, GL02b].

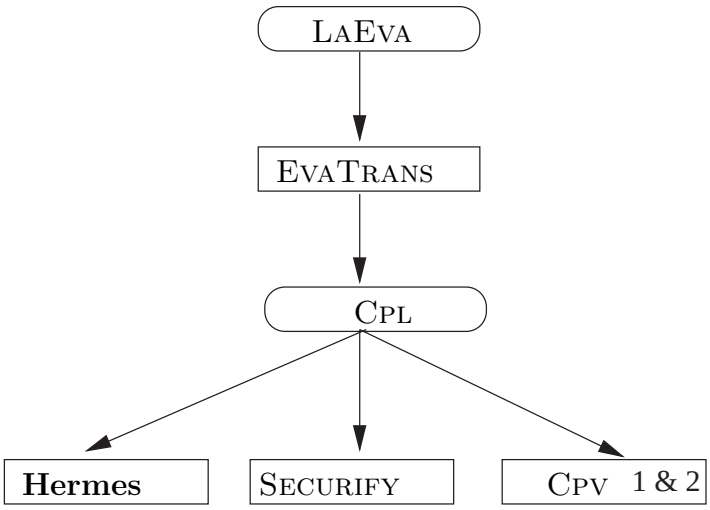


Figure 8.1: The architecture of the EVA tool-box.

## 8.1 Input Language

The input language, LaEva, is a high level specification language designed in the Eva project for the description of security protocols and their properties. This language is introduced in [JM01] and is closed to several known security specification languages including Casper [Low97a], Casrul [JRV00] or Capsl [DMR00]. A LaEva protocol specification is automatically translated into Cpl specification, an intermediary format for protocol specification that constitute the entry point of Hermes (see Figure 8.1). We explain, with the help of Needham-Schroeder Lowe protocol, how a protocol is specified in LaEva and what is the semantics given by Hermes for such a specification. The complete specification of Needham-Schroeder Lowe protocol follows:

```
 1.   alg : asym_algo

 2.   A, B : principal
 3.   N1, N2 : number
 4.   keypair^alg  pbk,prk (principal)

 5.   everybody knows alg

 6.   everybody knows pbk
 7.   A knows A, B, prk(A)
 8.   B knows B, prk(B)

 9.   {
10.      1.A->B: {A, N1}_(pbk(B))
11.      2.B->A: {B, N1, N2}_(pbk(A))
12.      3.A->B: {N2}_(pbk(B))
13.   }

14.   s.session* {A,B,N1,N2}

15.   assume
16.      secret(prk(A)@s.A),
17.      secret(prk(B)@s.B),
18.      secret(prk(B@s.A)),
19.      secret(prk(A@s.B))

20.   claim
21.      *A*G secret(N1@s.A),
22.      *A*G secret(N2@s.B)
```

The first part of protocol specification consists of declarations of typed variables and key constructors. `pbk` and `prk` take as argument a principal and return an asymmetric key with respect to an algorithm `alg`. The constructor `pbk(A)` stands for public key of principal `A`. The private key of `A`, denoted by `prk(A)`, is the inverse of `pbk(A)`. The

type `number` is predefined and represents any message.

```
1.   alg : asym_algo

2.   A, B : principal
3.   N1, N2 : number
4.   keypair^alg  pbk,prk (principal)
```

The second part specifies the knowledge of the principals. It is necessary to rule out ambiguities when we generate the formal model of the protocol from the description of the messages exchanged during a session of the protocol. Notice here that principal `B` does not known the identity of principal `A`.

```
5.   everybody knows alg

6.   everybody knows pbk
7.   A knows A, B, prk(A)
8.   B knows B, prk(B)
```

The protocol is described in a style that is similar to the standard notation, see for example the survey [CJ97]. It defines the order of the messages exchanged during an ideal session. The role of initiator and responder – respectively denoted by `A` and `B` – and the nonces `N1` and `N2` that they create are the parameters of a session of the protocol.

```
9.   {
10.     1.A->B: {A, N1}_(pbk(B))
11.     2.B->A: {B, N1, N2}_(pbk(A))
12.     3.A->B: {N2}_(pbk(B))
13.  }
```

The last part of specification describes the verification configuration, i.e. the hypothesis and the properties to be verified.

```
14.  s.session* {A,B,N1,N2}

15.  assume
16.     secret(prk(A)@s.A),
17.     secret(prk(B)@s.B),
18.     secret(prk(B@s.A)),
19.     secret(prk(A@s.B))

20.  claim
21.     *A*G secret(N1@s.A),
22.     *A*G secret(N2@s.B)
```

For `session*` configuration, HERMES considers an unbounded number of sessions in parallel.

The `assume` section provides the secrecy hypothesis that are exploited in HERMES's reasoning. It defines the properties holded by the system which can be used to prove

the claims, for example keys unknown to the intruder which can be used to safely encrypt messages. `secret(prk(B@s.A))` means that the private key – of the entity playing the role of the responder (`B`) from `A`'s point of view in session `s` – is secret.

The `*A*G` is required in claims for HERMES to check that secrecy properties hold *Always* and *Globally*. This asks that the nonces `N1` and `N2` created by role `A` (respectively role `B`) in any session `s` which satisfies secrecy hypothesis are secrets.

We can also describe a "fixed number of sessions" configuration. This case is useful for debugging purpose, but is not the complete and effective calculus that we have presented in the first part of this thesis. We will see later how we can use this functionality to reconstruct attacks. To specify a fixed number of session we write:

```
s1. session  A=a, B=b
s2. session  A=a, B=I
```

instead of `s.session* A,B,N1,N2`. Hence, the configuration is described by two parallel sessions one between $a, b$ and one between $a, I$. Notice that the names of participants do not have any meaning, all informations about theirs honesty is given in the `assume` part of description. Hence, if we want to say that $a, b$ are honests participants and $I$ is dishonest then the assume part is the following:

```
assume
   secret (prk(A)@s1.A), (* or secret (prk(A)@s2.A)    *)
                          (* as both are equal with $a$ *)
   secret (prk(A@s1.B)),
   secret (prk(B)@s1.B)
```

The adequate claims are in the finite case:

```
claim
   *A*G secret (N1@s1.A),
   *A*G secret (N2@s1.B)
```

## 8.2 Hermes modules

In this section we will describe the main modules of HERMES and their functionalities. The global architecture is given in the figure 8.2.

### 8.2.1 Extraction unit

In the context of EVA toolbox we have developed a translator which takes as input the protocol description in CPL representation and transforms it in a list of rules for the "bounded configuration" and respectively a list of parametrized rules for the "unbounded configuration". Also, from the CPL description our translator extracts two lists of terms which correspond to secrecy hypothesis and secrecy claims.

Using EVATRANS, a protocol specification LAEVA is compiled into a CPL specification (see appendix B for the CPL specification of NSL Protocol) and then the extraction

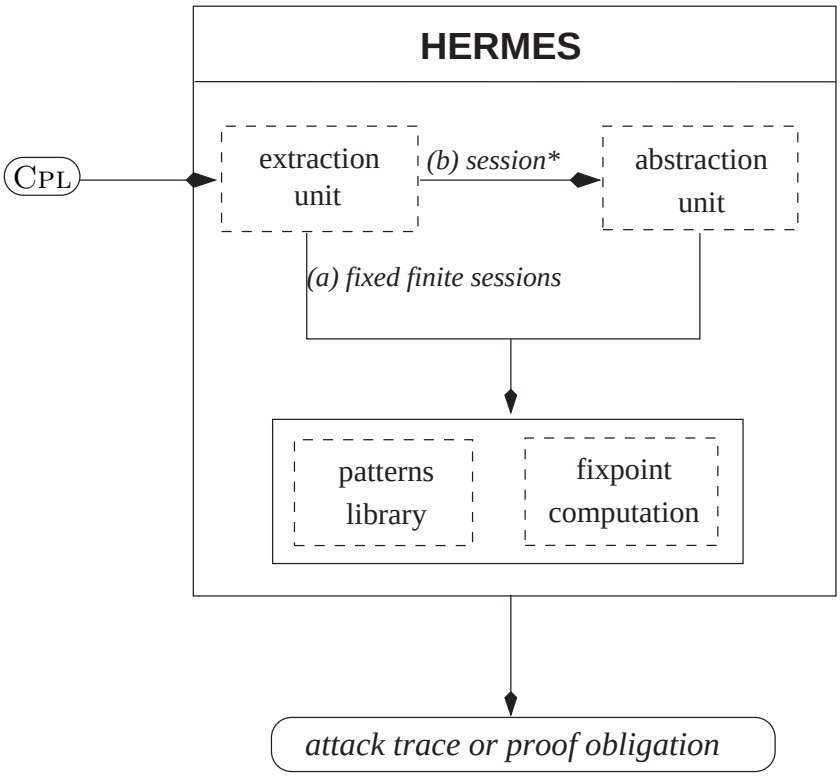


Figure 8.2: The architecture of the HERMES tool.

unit of HERMES draw out

(a) in the case of fixed number of sessions: the list of instantiated rules for each session instance and each participant of it

(b) in the case of unbounded number of sessions: the set of parameterized rules that define a generic session

and, in both cases, the set of secrecy hypothesis and secrecy claims. We illustrate these extraction phase on our running example, the Needham-Schroeder Lowe protocol.

### Unbounded case

In the unbounded case, the generic rules obtained from the CPL description are:

$$A: \quad (1.) - \rightarrow \{\mathtt{A}, \mathtt{N1}\}_{pbk(\mathtt{B})}; \qquad (2.)\{\mathtt{B}, \mathtt{N1}, \mathtt{x_3}\}_{pbk(\mathtt{A})} \rightarrow \{\mathtt{x_3}\}_{pbk(\mathtt{B})}$$

$$B: \quad (3.)\{\mathtt{x_1}, \mathtt{x_2}\}_{pbk(\mathtt{B})} \rightarrow \{\mathtt{B}, \mathtt{x_2}, \mathtt{N2}\}_{pbk(\mathtt{x_1})};$$

Generic rules of the concrete model parameterized by $(A,B,N1,N2)$

where $(A, B, N1, N2)$ are parameters and $x_1, x_2$ and $x_3$ denote local variables.

From the `assume` and `claim` sections we compute the secrecy property to prove:

$$\text{if } \mathit{Secret}(\mathcal{S}, \{\!|\mathit{pvk}(A), \mathit{pvk}(B), \mathit{pvk}(x_1)|\!\}, E_0) \text{ then } \mathit{Secret}(\mathcal{S}, \{\!|N1, N2|\!\}, E_0)$$

Where $\mathcal{S}$ is the set of generic rules, i.e. the parametrized session description and $E_0$ the intruder initial knowledge.

The above rule and the secrecy property are input data for the abstraction unit.

### Bounded case

In the case of finite number of sessions, we obtain the list of instantiated rules for each session instance and each principal. We have to take care of the order in which the rules may be executed and to the names of local variables as the same principal may be involved in several sessions. For the fixed configuration defined in the previous section the rules are:

$$A, s1: \quad (1.) - \rightarrow \{\mathtt{a}, \mathtt{N1}^{\mathtt{s1}}\}_{pbk(\mathtt{b})}; \qquad (2.)\{\mathtt{b}, \mathtt{N1}^{\mathtt{s1}}, \mathtt{x_3^{s1}}\}_{pbk(\mathtt{a})} \rightarrow \{\mathtt{x_3^{s1}}\}_{pbk(\mathtt{b})}$$

$$B, s1: \quad (1.)\{\mathtt{x_1^{s1}}, \mathtt{x_2^{s1}}\}_{pbk(\mathtt{b})} \rightarrow \{\mathtt{b}, \mathtt{x_2^{s1}}, \mathtt{N2}^{\mathtt{s1}}\}_{pbk(\mathtt{x_1^{s1}})};$$

$$A, s2: \quad (1.) - \rightarrow \{\mathtt{a}, \mathtt{N1}^{\mathtt{s2}}\}_{pbk(\mathtt{I})}; \qquad (2.)\{\mathtt{I}, \mathtt{N1}^{\mathtt{s2}}, \mathtt{x_3^{s2}}\}_{pbk(\mathtt{a})} \rightarrow \{\mathtt{x_3^{s2}}\}_{pbk(\mathtt{I})}$$

$$B, s2: \quad (1.)\{\mathtt{x_1^{s2}}, \mathtt{x_2^{s2}}\}_{pbk(\mathtt{I})} \rightarrow \{\mathtt{I}, \mathtt{x_2^{s2}}, \mathtt{N2}^{\mathtt{s2}}\}_{pbk(\mathtt{x_1^{s2}})};$$

The secrecy property is:

$$\text{if } \mathit{Secret}(\mathcal{S}, \{\!|\mathit{pvk}(a), \mathit{pvk}(b), \mathit{pvk}(x_1^s 1)|\!\}, E_0) \text{ then } \mathit{Secret}(\mathcal{S}, \{\!|N1^{s1}, N2^{s1}|\!\}, E_0)$$

Now, we are ready to prepare the input for the verification module.

First we eliminate the variables from the left side of property, if possible. In our case we replace the variable $x_1^{s1}$ with $a$ respectively $b$ in every rule where it appears because the $\mathit{pvk}(x_1^s 1)$ must be secret and in our hypothesis only $\mathit{pvk}(a)$ and $\mathit{pvk}(b)$ are secrets. Hence, instead of the parameterized rule:

$$\{x_1^{s1}, x_2^{s1}\}_{\mathit{pbk}(b)} \rightarrow \{b, x_2^{s1}, N2^{s1}\}_{\mathit{pbk}(x_1^{s1})}$$

we will have the two concrete rules:

$$\{a, x_2^{s1}\}_{\mathit{pbk}(b)} \rightarrow \{b, x_2^{s1}, N2^{s1}\}_{\mathit{pbk}(a)} \text{ and}$$

$$\{b, x_2^{s1}\}_{\mathit{pbk}(b)} \rightarrow \{b, x_2^{s1}, N2^{s1}\}_{\mathit{pbk}(b)}.$$

In general, the `assume` section states the honest principals, given the information about their private or shared keys. Therefore, we use first this information to replace the key variables that are specified in the `assume` section with a subset of their corresponding keys and second, all others key variables are replaced with all possible values.

Also, we have to eliminate all variables which are in key positions, as our method does not work with non-atomic keys. Therefore, we replace all key variables with all possible values. In the example above, because of the variable key $\mathit{pbk}(x_1^{s2})$ we replace the variable $x_1^{s2}$ with principals of our configuration, i.e. $I, a$ and $b$. Finally, we have to verify six configurations which corresponds to the configuration $s1, s2$ given in the protocol description.

In our running example, the set of safe keys is $K = \{\!|\mathit{pbk}(a), \mathit{pbk}(b)|\!\}$ and the initial set of secrets is $S = \{\!|\mathit{pvk}(a), \mathit{pvk}(b), N1^{s1}, N2^{s1}|\!\}$.

### 8.2.2 Abstraction unit

This unit is used only in the case of unbounded verification. Starting from the set of generic rules it generates all abstract rules as described in the section 6.3.

Also, we have implemented the particular abstraction where we consider only two principals, one honest principal $H$ and one dishonest principal $I$. That this abstraction is safe, and actually also exact, is proved in a nearby model given by the authors of [CLC04]. This abstraction reduces considerably the execution time for the verification algorithm.

Let consider again the Needham-Schroeder Lowe protocol. In the role of $B$ the variable $x_1$ represents the participant with whom $B$ plays the session. This variable is instantiated with $H$ or with $I$ following the abstraction from $B$'s point of view. Hence, for our running protocol we have the following abstract rules:

| Sessions | Transitions | | |
|---|---|---|---|
| fixed session $(H, H)$ | $- \rightarrow \{H, N_1\}_{pbk(H)}$; | $\{H, N_1, z\}_{pbk(H)} \rightarrow \{z\}_{pbk(H)}$; | $\{H, y\}_{pbk(H)} \rightarrow \{H, y, N_2\}_{pbk(H)}$; |
| any session $(H, H)$ | $- \rightarrow \{H, N_1^{HH}\}_{pbk(H)}$; | $\{H, N_1^{HH}, z\}_{pbk(H)} \rightarrow \{z\}_{pbk(H)}$; | $\{H, y\}_{pbk(H)} \rightarrow \{H, y, N_2^{HH}\}_{pbk(H)}$; |
| session $(I, H)$ | $- \rightarrow \{I, N_I\}_{pbk(H)}$; | $\{H, N_I, z\}_{pbk(I)} \rightarrow \{z\}_{pbk(H)}$; | $\{I, y\}_{pbk(H)} \rightarrow \{H, y, N_2^{IH}\}_{pbk(I)}$; |
| session $(H, I)$ | $- \rightarrow \{H, N_1^{HI}\}_{pbk(I)}$; | $\{I, N_1^{HI}, z\}_{pbk(H)} \rightarrow \{z\}_{pbk(I)}$; | $\{H, y\}_{pbk(I)} \rightarrow \{I, y, N_I\}_{pbk(H)}$; |

The set of abstract safe keys is $K = \{pbk(H)\}$ and the initial set of abstract secrets is $S = \{pvk(H), N1, N2\}$.

### 8.2.3 Verification unit

The verification unit of HERMES implements the depth-first search fix-point algorithm described in section 7.3.

### 8.2.4 Output Interpretation

At termination, HERMES yields an augmented set of secrets $S'$ ($S \subseteq S'$) and a set of pattern transducers that is called *BadPatterns*. This result is interpreted as follows: *"the protocol satisfies the secrecy property if, in the set $E_0$ of messages initially known by the intruder the set of terms $S'$ is set-protected despite the set of pattern transducers BadPatterns"*.

Also, HERMES provides a tree that can be interpreted to construct a proof for the correctness of the protocol or may help to understand and reconstruct the attacks if the secrecy property is not verified. In the later case, HERMES points out attacks in the case of obvious invalidation of the property, i.e. secrets that are not fresh variables or that have not been secrets in the initial hypothesis.

Let us interpret the HERMES results on the running example.

#### Correct version

In the unbounded case, HERMES finishes with the set of secrets $\{pvk(H), N1, N2\}$ and the set of pattern transducers

$$\{(\{(H, (N1^{HH}, Sup(I, x2s)))\}_{pbk(H)}, 01101);$$
$$(\{(I, x2s)\}_{pbk(H)}, 01);$$
$$(\{(H, (N1, Sup(I, x2s)))\}_{pbk(H)}, 01101)\}.$$

As an implementation detail, notice that positions in pattern transducers are tracked by special variables. That is, pattern transducers are simply terms which use special

variables (i.e, with names finishing in the letter $s$, as *special*). Transducer positions are positions where a special variable occurs. For example, the pattern $\{(I, x2s)\}_{pbk(H)}$ corresponds to the pattern transducer $(\{(I, x2)\}_{pbk(H)}, 01)$.

Hence, the protocol satisfies the secrecy property if, in the set of messages initially known by the intruder the set of terms $\mathbf{\{}pvk(H), N1, N2\mathbf{\}}$ is set-protected despite the set of pattern transducers returned by HERMES. This condition holds for the secrets $N_1$ and $N_2$ since their freshness ensures that they do not appear in any message before a session run. Also, the $pvk(H)$ is in the secrecy hypothesis of our verification problem.

We conclude that the secrecy property is satisfied and the tree of pattern transducers (see the figure 8.3) can be interpreted to extract a proof for correctness of the protocol. More precisely, starting from HERMES output one can automatically extract complete proof scripts which may be interpreted by proof assistants. For example, proof scripts for Coq [BBC+97] have been obtained by an automatic proof extractor developed by our team in the context of EVA project [JPL03].

**Bugged version**

In the original version of the protocol due to Needham-Schroeder, the identity of role `B` did not appear in the message sent by $B$. On this version HERMES returns the set of secrets $\mathbf{\{}pvk(H), N1, N2, \{H, N1^{HI}\}_{pbk(H)}, \{H, I\}_{pbk(H)}\mathbf{\}}$

Obviously, the message $\{H, N1^{HI}\}_{pbk(H)}$ cannot be secret: it does not contain any fresh nonce or initial secret, so it could be forged by the intruder.

Starting from this message and applying the rule:

$$\{H, y\}_{pbk(H)} \rightarrow \{y, N_2\}_{pbk(H)}$$

of the set of abstract rules, the intruder can obtain the message $\{(N1^{HI}, N2)\}_{pbk(H)}$, and finally applying the rule:

$$\{N_1^{HI}, z\}_{pbk(H)} \rightarrow \{z\}_{pbk(I)}$$

he obtains $N_2$. This attack corresponds to the well known attack of Needham-Schroeder discovered by Lowe [Low95].

Also, message $\{H, I\}_{pbk(H)}$ cannot be a secret. This secret leads to an attack which is not as well known as the previous one. It is an attack due to type confusion, a nonce is confused with an agent name. We typed the messages which are used to encrypt messages, as we allow encryption only with keys.

Starting from the message $\{H, I\}_{pbk(H)}$ and applying the rule:

$$\{H, y\}_{pbk(H)} \rightarrow \{y, N_2\}_{pbk(H)}$$

of the set of abstract rules, the intruder can obtain the message $\{(I, N2)\}_{pbk(H)}$, and applying the rule:

$$\{I, y\}_{pbk(H)} \rightarrow \{y, N_2^{IH}\}_{pbk(I)}$$

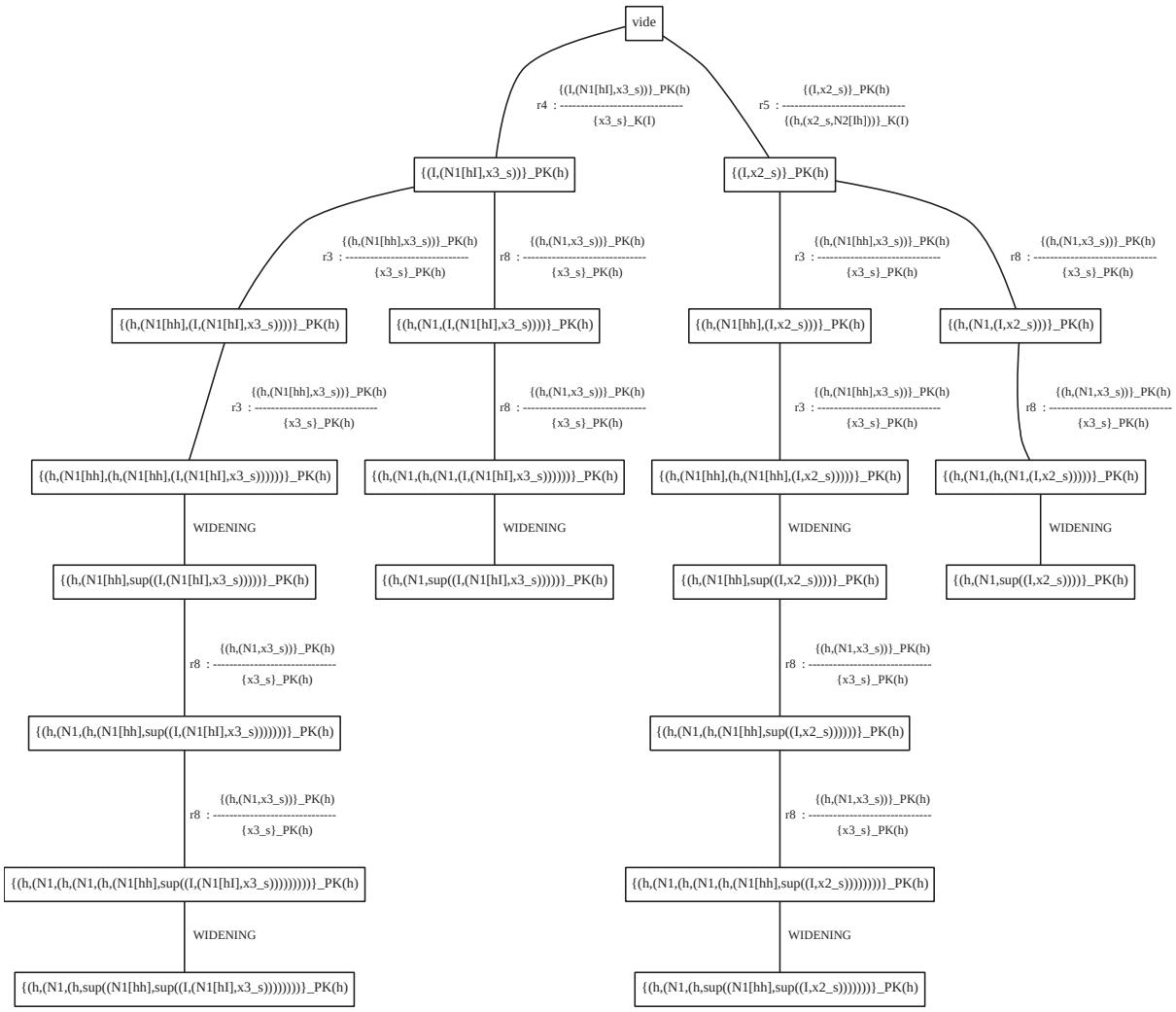


Figure 8.3: HERMES proof tree output for Needham-Schroeder Lowe

the intruder obtains the message $\{N2, N_2^{IH}\}_{pbk(I)}$ hence, $N_2$.

We present in the figure 8.4 a part of the tree generated by HERMES. We can see the traces which have been stopped with attack and also how each pattern transducers has been generated.

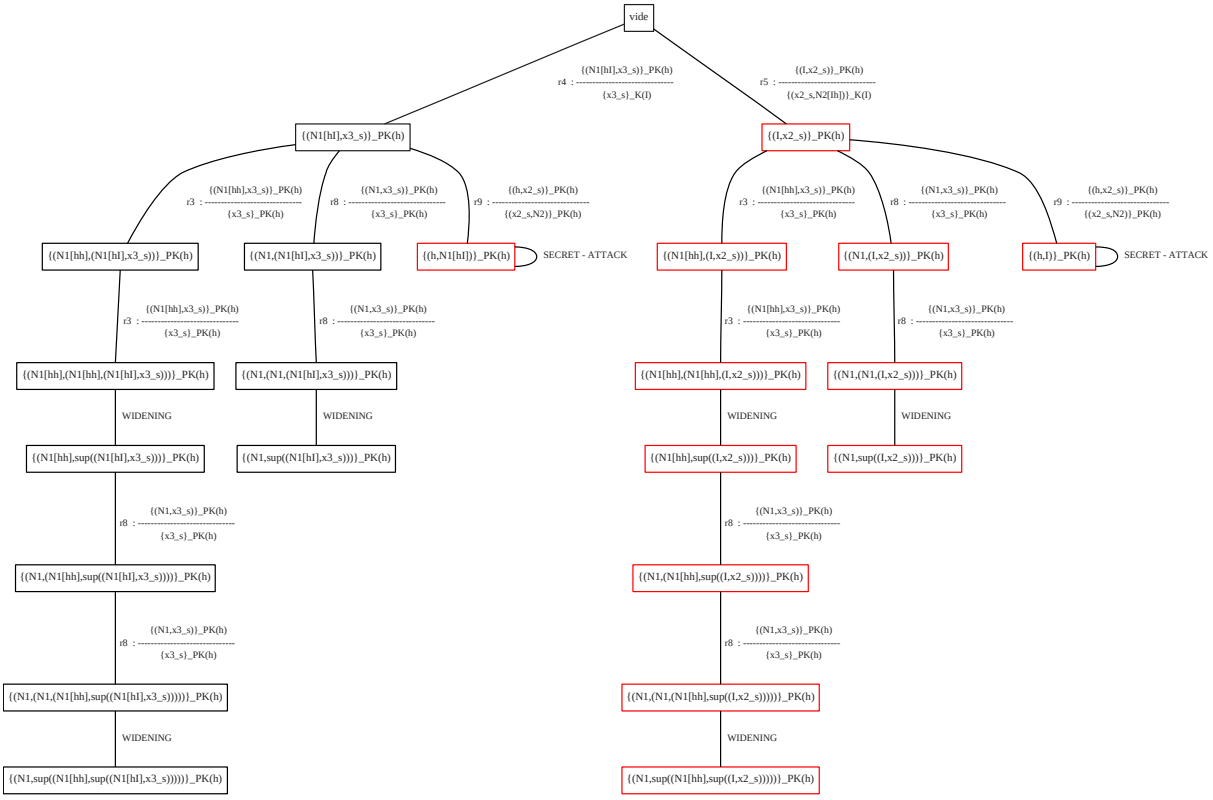


Figure 8.4: HERMES attack tree output for Needham-Schroeder unbounded version

Finally, let us observe that the attack has been generated using the abstract rules. In the first attack we used rules corresponding to the abstract session $(H, H)$ and respectively the abstract session $(H, I)$. In the second attack, we used rules corresponding to the abstract session $(H, H)$ and respectively the abstract session $(I, H)$. Starting from this information we can execute HERMES with a fixed number of sessions in order to find a concrete attack and to understand it better, since the concrete rules correspond to concrete sessions, and therefore to concrete executions.

For example, the first attack is recover in the following bounded configuration:

```
s1. session  A=a, B=b
s2. session  A=a, B=I
```

Which means the configuration is described by two parallel sessions one between $a, b$ and one between $a, I$, with $a$,$b$ honests and $I$ dishonest. The honesty hypothesis are given in the `assume` part of description as follows:

```
assume
   secret (prk(A)@s1.A),  (* i.e. a is honest *)
   secret (prk(A@s1.B)),  (* i.e. the initiator in the session s1
                             from b's point of view is honest *)
   secret (prk(B)@s1.B)   (* i.e. b is honest *)
```

The adequate claims are:

```
claim
   *A*G secret (N1@s1.A),
   *A*G secret (N2@s1.B)
```

HERMES analyses each possible execution trace and, if the attack is not a false one, it will be found on one of them.

### 8.2.5 User Interface

HERMES is available online at the url
`http://www-verimag.imag.fr/~Liana.Bozga/eva/hermes.php`.

Several protocol examples are available in the web page and others may be written directly in the input area using the LAEVA syntax. As output, HERMES provides, in the output area, the result sets $S'$ (Secrets), and $PT'$ (BadPatterns) as well as the sequence of rules leading to an attack if it is the case.

Also, at each execution HERMES generates a tree allowing to follow the sequence of rules leading to each new secret or pattern transducer. A link to this tree is available at the end of the output area. The information provide by HERMES are very useful for someone how known the verification method, because that information is derived from the abstract version of the algorithm. We are working to improve the output information. For the case when HERMES fails in proving the correctness of the protocol we have started to developed a module which reconstruct the concrete attack if it exists.

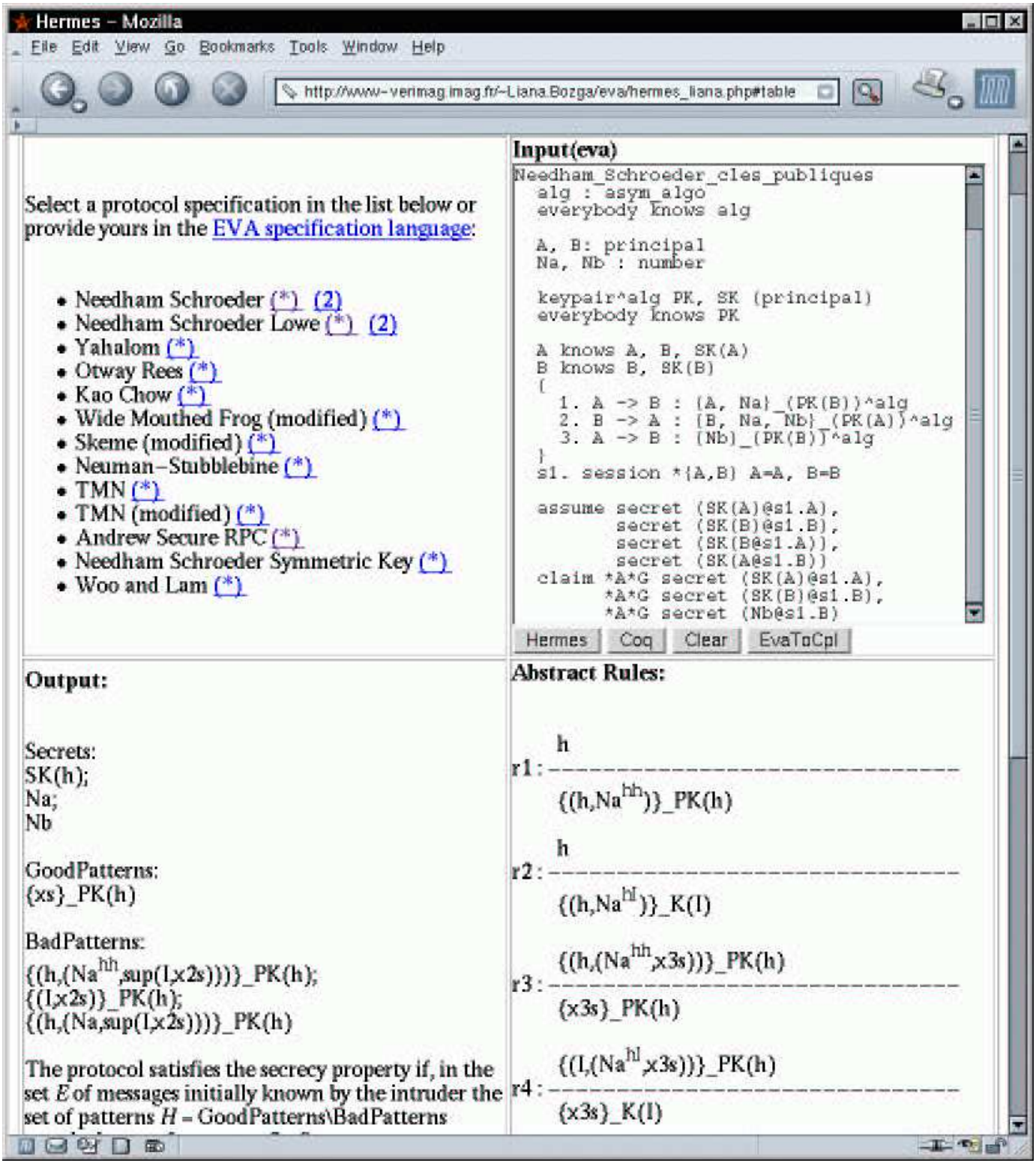


Figure 8.5: The user interface of HERMES.

## 8.3 Hermes results

The table 8.1 summarizes the results obtained by HERMES on some protocols regarding secrecy properties. The LAEVA description and a bibliographic reference for each of these protocols are given in the appendix C.

The ♯R column represents the number of abstract rules. The ♯N column represents the number of nodes in the derivation tree, the term transducers and the secrets generated during the computation. The ♯W column represents the number of times the widening procedure has been applied. The *time* column represents the overall execution time (in seconds). Finally, in the last column *OK* means that the protocol has been successfully verified for the secrecy property. *Attack* means that an attack has been found and, as all these attacks have been already discovered and published we give also their principal references. In general, a third result, *Inconclusive* can be obtained: because of abstraction step, it is possible to find an attack at abstract level, which in fact, is not a valid one, on the concrete level.

| Protocol Name | ♯ R | ♯ N | ♯ W | Time | Result |
|---|---|---|---|---|---|
| Needham Schroeder Public Key | 9 | 46 | 14 | 0.04 | Attack [Low96] |
| Needham Schroeder Lowe | 9 | 20 | 6 | 0.02 | OK |
| Yahalom | 35 | 563 | 132 | 29.42 | OK |
| Otway Rees | 15 | 0 | - | 0.04 | OK* |
| Denning Sacco Key Distribution with Public Key | 18 | 3 | - | 0.03 | Attack [AN96] |
| Wide Mouthed Frog (modified) | 14 | 2 | - | 0.03 | OK |
| Kao Chow | 50 | 74 | 16 | 1.08 | OK |
| Neumann Stubblebine | 75 | 3 | 1 | 0.10 | OK* |
| Needham Schroeder Symmetric Key | 65 | 6 | - | 0.08 | Attack [DS81] |
| TMN | 41 | 17 | - | 0.01 | Attack [LR97] |
| Andrew Secure RPC | 27 | 5 | - | 0.01 | OK |
| Woo and Lam Mutual Authentication (modified) | 102 | 4 | - | 0.10 | OK |
| Skeme [Kra96] (modified) | 69 | 5 | - | 0.03 | OK |

* There is a known attack of the untyped version of the protocol. Discovering this type attack automatically requires to deal with non-atomic keys. This is not yet implemented in HERMES.

Table 8.1: These results have been obtained by running HERMES on a Pentium III 600Mhz PC under Linux.

## 8.4 Comparison with others tools

We present below some of the verification tools for cryptographic protocols, including their main functionalities and limitations.

### 8.4.1 Model-checking tools

The first category of protocol verification tools are model-checking tools. In general, such tools have been applied to discover flaws in cryptographic protocols. Usually, these tools bound the number of sessions and the number of participants. But, even in this case the problem is difficult because of the explosion of the state space due to interleaved execution of sessions and message sizes. Hence, each such tool requires some supplementary simplifying assumptions e.g, bound on the size of messages. Finally, the tools in this category are automatic and deal with both secrecy and authentication properties. The most known are Casper and Casrul, presented below.

**Casper** [Low97a] is a compiler which takes a security protocol specification and produces an equivalent description in process algebra CSP [Hoa85]. This description is then checked using the model-checker FDR [Ros94]. The supplementary assumptions done by FDR are that messages are bounded, well-typed and the keys are atomic. Nevertheless, it is one of the most complex tool for cryptographic protocols and it is able to find all reported attacks from the survey of Clark and Jacob [CJ97] except type attacks. In particular, the well-known attack of Needham-Schroeder Protocol has been first discovered by Gavin Lowe using CSP/FDR tools [Low95].

Similar to this tool, but without significant improvements are Murphi [MMS97] and Athena [Son99]. In particular, the second one deals with unbounded but typed messages.

The AVISS tool [AVC+02] implements model-checking methods for security protocol analysis over a common high level protocol specific language. A static analysis is performed to check the executability of the protocol and then the protocol and the intruder actions are compiled into an intermediate format. The intermediate format (IF) is the start point for three complementary automated protocol analysis techniques: On-the-Fly model-checking (OFMC) [BMV03], Constraint-Logic-Based model-checking Casrul [JRV00] and SAT-based model-checking (SATMC) [AC04]. Using at least one of them, sometimes all of them, one found all the reported attacks for protocols in Clark's survey. We remark that Casrul deals with messages in all their generality. More precisely, the messages are not typed, there is no bound on their size and composed key are taken into account. It found also type attacks.

Finally, remember that Hermes can be exploited in the case of a bounded number of sessions and partially typed messages: it requires only that all messages used as keys to be of key type. But, it deals with an abstract representation of the protocol though each rule is played one time and the order is taken in to account. Hence it may find fake attacks when the variables assignation is different for different rules of the same session. Hermes found also all reported attacks for secrecy properties of protocols from Clark's

survey, including also type attacks if they do not involve composed keys.

### 8.4.2 Induction and Theorem-proving based tools

The second category of protocol verification tools includes tools based on induction and theorem proving. Usually, these tools provide a general proof method of correctness for protocols. In this case, as in our, the approach is oriented around proving guarantees, while their absence indicates possible attacks. The tools in this category deal with unbounded number of sessions and participants. We mention here the Paulson tool [Pau97] and SECURIFY [CMR01].

In the tool developed by Paulson, protocols are defined inductively as sets of traces. It does not require an a priori typing of messages. It deals with both secrecy and authentication properties. The proofs are generated using Isabelle/HOL [Pau94]. The tool is not completely automatic, i.e. user interaction is required to obtain the proof. More precisely, the user has to choose among several proof strategies.

SECURIFY [CMR01, Cor02] is completely automatic. It deals with an unbounded number of sessions and does not require an a priori typing of messages. However, usually it can't conclude if some of the messages are not well typed. In the case where the protocol can be proved correct, SECURIFY produces a corresponding proof tree.

The main difference between HERMES and the others automatic tools of this category is that the HERMES algorithm is based on abstract interpretation and hence the proof tree obtained is in the abstract domain. The advantage of abstraction is that HERMES concludes more often than SECURIFY.

### 8.4.3 Abstract Interpretation based tools

In this category we found partial algorithms based on abstract interpretation. We mention here the tools that have been presented in [Mon99, GL00, GK00]. All of them use a model based on tree automata to abstract the set of intruder knowledge.

[Mon99] deals with a bounded number of sessions and participants. It is completely automatic but analyzes only secrecy properties. Usually, it is able to analyze only a small number of instances of the protocol, being very limited by the number of interleavings to consider. Some possible approximations, such as widening, are proposed to increase the number of sessions that can be analyzed.

CPV [GL00] is also fully automatic and proves correctness of protocols with respect to both secrecy properties. It deals with unbounded number of sessions and participants. The protocol is approximated by a set of rewriting rules and also the set of intruder knowledge is over approximated. It has been successfully applied on protocols from the Clark's survey and seems to be very efficient (less than 1 second on all tested examples).

TIMBUK [GK00] also deals with unbounded number of sessions and participants. This tool allows to prove correctness of protocols with respect to both secrecy and authentication properties. The protocol is represented by a set of rewriting rules. The translation

step from the standard protocol description to rules representation is manual and not easy, therefore we have just a few experimental results. Some acceleration techniques are used but the termination is not always guaranteed.

Regarding the assumptions made, HERMES is very close to these two last tools, but instead of computing the set of messages that can be known by the intruder, HERMES provides an invariant on the intruder knowledge that is a sufficient condition for secrecy. Moreover, when a protocol is proved correct, it returns an abstract proof tree that can be exploited for certification [JLP04]. Also, when it can not prove the correctness, it returns an abstract attack that can be used to construct a concrete attack, if such a concrete attack exists [JLP04, JPL03].

# Chapter 9

# Conclusions

## 9.1 Achievements

The first achievement of the thesis is the definition of a general model for cryptographic protocols. Our model includes a language for the description of cryptographic protocols together with an operational semantics allowing to understand their execution in a potentially hostile environment. The model is rich enough to represent protocol sessions at a high level of abstraction as sequences of message exchange between participants. In addition, the model allows to capture time-sensitive information using primitives derived from the theory of real-time systems.

The second achievement is the definition of a complete and effective Hoare Logic for timed bounded cryptographic protocols and an expressive assertion language. This assertion language allows to specify secrecy as well as authentication and other properties. As a consequence of this result, we have a decision procedure for timed bounded cryptographic protocols and a large class of security properties allowing an infinite set of messages initially known by the intruder. The latter point might seem minor but is not. Indeed, if we are interested in composing protocols we have to take into account that we have no bound on how many sessions have taken place before, and hence, we should allow infinite sets of messages. Thus, besides developing a result concerning the existence of an effective and complete Hoare Logic for cryptographic protocols, we significantly extend existing decidability results in two directions:

1. larger class of properties and
2. more general initial conditions.

Finally, the third achievement is an abstract interpretation based method for verifying secrecy properties of cryptographic protocols in a general model. We deal with unbounded number of sessions, unbounded number of principals, unbounded message depth and unbounded creation of fresh nonces. However, in contrast to the work presented in the first part of this thesis, where the session number is bounded, this method is not complete. Indeed, in this case, the problem is undecidable (e.g. see [AC02]). The

main contribution is a verification algorithm that consists of computing an inductive invariant using patterns as symbolic representation. In case the given protocol is correct, our method provides a proof tree that can be exploited for certification. More precisely, from the obtained proof tree we can automatically deduce a Coq proof that can serve for certification [JLP04].

## 9.2 Future work

Our results can be extended in several directions.

Regarding verification of bounded protocols, we expect that decidability results as well as the weakest precondition calculus remains valid on slightly different intruder models e.g, extensions of Dolev-Yao models. Moreover, we believe that our bounded timed model can be extended to handle a richer class of timed cryptographic protocols e.g., including for instance drifting clocks.

Several perspectives are still open concerning the verification of unbounded protocols. Secrecy is not always sufficient to prove the security of a protocol. We have to study if our verification method can work for a richer class of security properties e.g., including at least authentication, or whether it is limited to secrecy. Moreover, even for secrecy, the general abstraction we propose can be too coarse and therefore useless, if it does not take into account extra-information about the overall behavior of the protocol e.g., when only iterative sessions are considered.

### Extensions of Dolev-Yao models

Our results rely upon common hypothesis about cryptographic protocols and their execution environment. In the Dolev-Yao model, the intruder has complete control on the network i.e., it can intercept all messages sent between participants. Moreover, the cryptography is considered perfect i.e, no decryption being possible without knowing the inverse key.

However, in practice we are often faced with much more specific situations. For example, we may have *secure communication channels*, that is, communication among some of the participants escapes from the intruder's control. Also, *cryptography is not perfect* in reality, for example, the xor-encryption has strong algebraic properties which can be easily exploited in order to decrypt messages.

We believe that *term transducers* represent a general framework for a uniform presentation of different results for bounded cryptographic protocols with both stronger communication e.g., secure channels and weaker cryptographic hypothesis.

As a concrete example, consider protocols which use *Cipher Block Chaining* (CBC) encryptions. Following the idea of [Rus03] the intruder may exploit prefix properties of encryption algorithms based on CBC. This can be expressed by the rule:

- If $E \vdash \{m_1, m_2\}_k$ then $E \vdash \{m_1\}_k$.

In order to catch this new intruder capability we can extend message transducers with a *cbc* function having the following operational semantics:

$$\frac{-}{M \xrightarrow{cbc} \{\!\{\{m_1\}_k \mid \{m_1, m_2\}_k \in M\}\!\}}$$

Then, the $I_d$ message transducer which encodes the decomposition ability of the intruder becomes $I_d = (cbc + pr_1 + pr_2 + \sum_{k \notin K} decr(\cdot, k))^*$.

Extended our results in this direction seems natural and we guess it represents a first step towards taking into account algebraic properties of encryption algorithms for the verification of correctness.

### Richer Time-sensitive protocols

The time model for bounded protocols can be also extended in several ways.

First, it can be naturally used to associate time values to *short term keys* such that if the intruder obtains a message encrypted by a short term key then after the specified amount of time elapses the key becomes known by the intruder. Then our verification method can be extended to handle this model.

Intuitively, to each short term key $k$ we associate a constant $\Delta_k$ and a clock $c(k)$ that can be activated when the intruder deduces a message of the form $\{x\}_k$. Then, when the value of the clock $c(k)$ reaches $\Delta_k$, the key $k$ becomes deducible by the intruder. That is, a short term key $k$ is "cracked" $\Delta_k$ time units after a message $\{x\}_k$ becomes known.

Second, our time model can be extended to handle *drifting clocks*. It is well-known that models with clocks with drifts in bounded intervals can be transformed into models with perfect clocks modulo an abstraction, that is, taking into account more behavior. As discussed by Gong [Gon92] drifting clocks can add subtle attacks.

### Abstraction and Unbounded verification

A clear limitation of our verification method for unbounded number of sessions is that we can verify only secrecy properties. We are working to extend it for other security properties such as authentication, anonymity, opacity or fairness.

Moreover, our abstraction scheme obeys some common general hypothesis about the behavior of participants in cryptographic protocols. In particular, we allow participants to be involved in several parallel sessions at the same time. But, in practice the situation is sometimes different, that is, there are protocols with *iterative sessions* where, due to physical reasons, a participant can be involved only in one session at a time. When this hypothesis is crucial for the correctness of the protocol, it must be preserved by the abstraction, otherwise no meaningful result can be a priori obtained. For example, we

may consider other abstractions e.g, abstract sessions like

$$((H, I)^*; (H, H)^*; (I, H)^*)^*; (H, H); ((H, I)^*; (H, H)^*; (I, H)^*)^*$$

in order to preserve the iterative nature of the protocol.

# Appendix A

# Dealing with patterns

In fact, introducing the interpreted function symbol *Sup* corresponds to adding the sub-term relation to a logic on terms.

Unification and matching are the key operations in dangerous substitution computation of the symbolic verification algorithm, chapter 7. The problem we need to solve for obtaining our symbolic algorithm is, however, *not* unification of patterns. The problem we need to solve is the following: Given two patterns $u$ and $v$, we have to determine a set $\mathcal{U}(u, v)$ of substitutions $\sigma$ such that there exist terms $u' \in [\![u]\!]$ and $v' \in [\![v]\!]$ such that $\sigma(u') = \sigma(v')$. More precisely, we want to characterize the set of most general unifiers that unify some terms in $[\![u]\!]$ and $[\![v]\!]$. Actually, the problem we need to solve for our symbolic algorithm is a simpler one where at least one of the patterns $u$ and $v$ is simply a term, i.e. without occurence of *Sup* in it. Indeed, we need to unify the conclusion of a rule, which is a term, with a pattern from a pattern transducer.

We prefer, however, to present a solution for the general case. We will do this in a general setting.

Let us consider a finite set $\mathcal{F}$ of function symbols such that $Sup \notin \mathcal{F}$ and let $\mathcal{X}$ be a countable set of variables. The set of patterns induced by $\mathcal{F}$ and $\mathcal{X}$, denoted by $\mathcal{PT}(\mathcal{F}, \mathcal{X})$ is defined by the following BNF:

$$t ::= x \mid f(t_1, \cdots, t_n) \mid Sup(t)$$

where $x$ is a variable in $\mathcal{X}$ and $f$ is a function symbol of arity $n \geq 0$. Let $\mathcal{F}^{(i)}$ denote the function symbols in $\mathcal{F}$ of arity $i$. As usual, function symbols of arity 0, i.e. elements of $\mathcal{F}^{(0)}$, are called constants. The meaning $[\![t]\!]$ of a pattern $t$ is a set of terms in $\mathcal{T}(\mathcal{X}, \mathcal{F})$ defined in the same way as in definition 7.4.

**Definition A.1**

*Given two patterns $u$ and $v$, a substitution $\sigma : \mathcal{X} \to \mathcal{PT}(\mathcal{X}, \mathcal{F})$ is called a* maximal general unifier *for $u$ and $v$, if the following conditions are satisfied:*

1. *it is a most general unifier for some terms $u' \in [\![u]\!]$ and $v' \in [\![v]\!]$ and*

We consider the case:
$\bigcup_{i\geq 2}\mathcal{F}^{(i)} \neq \emptyset$ with $f,g \in \mathcal{F}^{(n)}$ ; $x \in \mathcal{X}$ ; $u,v,u_i,v_i \in \mathcal{PT}$ ; $t,t_i \in \mathcal{T}$ and $Sup \notin \mathcal{F}$

| | | | |
|---|---|---|---|
| Delete | $\{u = u\} \cup E$ | $\Rightarrow$ | $E$ |
| Orient | $\{u = x\} \cup E$ | $\Rightarrow$ | $\{x = u\} \cup E$, if $u \notin \mathcal{X}$ |
| Decompose | $\{f(u_1,\cdots,u_n) = f(v_1,\cdots,v_n)\} \cup E$ | $\Rightarrow$ | $\{u_i = v_i \mid i \in \mathbb{N}_n\} \cup E$ |
| Clash | $\{f(u_1,\cdots,u_n) = g(v_1,\cdots,v_m)\} \cup E$ | $\Rightarrow$ | $\bot$ |
| Eliminate | $\{x = u\} \cup E$ | $\Rightarrow$ | $\{x = u\} \cup E[u/x]$, if $x \notin var\,(u)$ |
| Occurs-Check | $\{x = u\} \cup E$ | $\Rightarrow$ | $\bot$, if $x \in var\,(u) \wedge u \neq Sup(x) \wedge u \neq x$ |
| Sup-Delete-1 | $\{x = Sup(x)\} \cup E$ | $\Rightarrow$ | $E$ |
| Sup-Delete-2 | $\{Sup(u) = v\} \cup E$ | $\Rightarrow$ | $E$, if $Sup(.)$ appears in $v$ |
| Sup-Splitting | $\{Sup(u) = f(t_1,\cdots,t_n)\} \cup E$ | $\Rightarrow$ | $\{u = t\} \cup E$, for $t \preceq f(t_1,\cdots,t_n)$, if $Sup(.)$ does not appear in $f(t_1,\cdots,t_n)$ |

Figure A.1: Usual rules for solving unification extended to deal with superterms

2. *for every substitution $\sigma'$ that unifies terms in $[\![u]\!]$ and $[\![v]\!]$, $\sigma'$ is not more general than $\sigma$, that is, for no substitution $\rho$, we have $\sigma = \rho \oplus \sigma'$.*

*We denote by $\mathcal{U}(u,v)$ the set of maximal general unifiers for $u$ and $v$.*

■

In general there will be more than one maximal general unifier for $u$ and $t$ even modulo renaming. The definition of $\mathcal{U}$ can be extended in the usual way –as for unification– to sets $\{(u_i,v_i) \mid i \in [1,n]\}$ of pairs of patterns. In the sequel, we prefer to write $u_i = v_i$ instead of $(u_i,v_i)$ as our algorithm essentially consists in manipulating some kind of equations.

We present an algorithm that given a set of patterns pairs $E = \{u_i = v_i \mid i \in [1,n]\}$ determines $\mathcal{U}(E)$. From now on, we will call such a set $E$ a *generalized equational problem*, written *GEP* for short.

It turns out that an extension of the set of transformations that solve the usual unification problem (cf. [BN98]) will give the solution.

We recall in the figure A.1 the usual six rules of [BN98] for solving unification and we add three rules to deal with the *Sup* operator.

We only solve the unification problem in the case of a signature $\mathcal{F}$ with at least a constructor of arity greater than one ; we do not present here the rules for the case of signature with only unary constructors which are useless in the context of crytography. We attract the reader's attention to the fact that the *Sup-Splitting* rule transforms a *GEP* $E$ into a set of *GEP*s. Indeed, it yields a new *GEP* for each sub-term of $f(t_1,\cdots,t_n)$. This is not the case for the usual unification rules.

**Example A.1**

*Consider the following* GEP

$$\begin{cases} [1] & \mathrm{Sup}(x) = f(b, g(\mathrm{Sup}(a))) \\ [2] & f(b, \mathrm{Sup}(g(x))) = f(b, g(b)) \end{cases}$$

*Equation 1 is eliminated by the rule (Sup-Delete-2) of the figure A.1. Indeed, it is equivalent to* $x \preceq f(b, g(\mathrm{Sup}(a)))$ *which puts no constraint on* $x$ *as long as the term signature contains a contructor with arity greater than one (e.g.,* **pair***).*

*Indeed, whatever the term we obtain for* $x$ *it is always possible, using binary contructors, to adjust the* Sup *part in* $f(b, g(\mathrm{Sup}(a)))$ *in order to obtain a term that contains* $x$*. As an example,* $f(b, g(\mathbf{pair}(x, a))$ *contains* $x$ *and it is an instance of* $f(b, g(\mathrm{Sup}(a)))$*.*

*The rule* Decompose *removes Equation 2 and produces the constraints* $\mathrm{Sup}(g(x))) = g(b)$ *and* $b = b$ *(which is eliminated by the rule* Delete*). Then, by rule* Sup-Splitting*, the former equation yields two* GEP*s:* $\{g(x) = g(b)\}$ *and* $\{g(x) = b\}$*. Finally, we obtain the solution* $x = b$*.*

■

Termination of the algorithm can be proved using lexicographic ordering and the ranking function that maps a *GEP* $E$ to $(m_1, m_2, m_3)$, where:

- $m_1$ is the number of variables in $E$ that are not solved. As usual, a variable $x$ is *solved* in $E$ if it occurs exactly once in $E$, namely on the left-hand side of some equation $x = u$ with $x \notin var\,(u)$.
- $m_2$ is the measure of $E$ defined by $\mathcal{M}(E) = \sum_{u=v \in E} (|u| + |v|)$ and $\mathcal{M}(\bot) = 0$,
- $m_3$ is the number of equations $u = x$ in $E$ with $x \in \mathcal{X}$ and $u \notin \mathcal{X}$.

The application of a rule to a *GEP* $E$ leads to one or more *GEP*s with a lower rank than $E$. Although the *Sup-Splitting* rule of the figure A.1 increases the number of *GEP*s, this number is bounded by the number of subterms of the right-hand side term. The ranking function and the bounded number of deriveable *GEP*s ensure the termination of the algorithm.

To prove soundness of the algorithm, we prove for each rule $E \Rightarrow E_1, \cdots, E_n$ that we have $\mathcal{U}(E) = \bigcup_{1 \leq i \leq n} \mathcal{U}(E_i)$.

# Appendix B

# Needham-Schroeder Lowe

## B.1 CPL Descriptions

### B.1.1 Unbounded version

Below, there is included the CPL description of the Needham-Schroeder Lowe Protocol [Low96], automatically generated from LAEVA specification, in the case of an unbounded number of sessions:

```
s.session* {A,B,N1,N2}
```

---

```
**asymk**(*D*) : number hash
A : principal
B : principal
N1 : number
N2 : number
pbk(principal) : number hash
prk(principal) : number hash
_kpf1(principal) : number hash
alg : asym_algo
alias prk = lambda-privk^(alg)(_kpf1)
alias pbk = lambda-pubk^(alg)(_kpf1)
process*{A,B,N1,N2} s1.A : 1. {
  1. -> \%1. : new N1-1
  \%1. -> 2. : send {(A,N1-1)}_(apply-pubk^(alg)(_kpf1,B))^(alg)
  2. -> 3. : recv {(<B>,<N1-1>,_x3)}_(apply-privk^(alg)(_kpf1,A))^(alg)
  3. -> end. : send {_x3}_(apply-pubk^(alg)(_kpf1,B))^(alg)
}
(A = A
 apply-privk^(alg)(_kpf1,A) = apply-privk^(alg)(_kpf1,A)
 N2 = _x3
 N1 = N1-1
 B = B
 alg = alg
```

```
 lambda-pubk^(alg)(_kpf1) = lambda-pubk^(alg)(_kpf1))

process*{A,B,N1,N2} s1.B : 1. {
  1. -> 2. : recv {(_x1,_x2)}_(apply-privk^(alg)(_kpf1,B))^(alg)
  2. -> \%4. : new N2-1
  \%4. -> 3. : send {(B,_x2,N2-1)}_(apply-pubk^(alg)(_kpf1,_x1))^(alg)
  3. -> end. : recv <{N2-1}_(apply-pubk^(alg)(_kpf1,B))^(alg)>
}
(A = _x1
 N2 = N2-1
 apply-privk^(alg)(_kpf1,B) = apply-privk^(alg)(_kpf1,B)
 N1 = _x2
 B = B
 alg = alg
 lambda-pubk^(alg)(_kpf1) = lambda-pubk^(alg)(_kpf1))

assume secret((apply-privk^(alg)(_kpf1,A))@s1.A)
assume secret((apply-privk^(alg)(_kpf1,B))@s1.B)
assume secret(apply-privk^(alg)(_kpf1,(B)@s1.A))
assume secret(apply-privk^(alg)(_kpf1,(_x1)@s1.B))
claim *A *G secret((apply-privk^(alg)(_kpf1,A))@s1.A)
claim *A *G secret((apply-privk^(alg)(_kpf1,B))@s1.B)
claim *A *G secret((N1-1)@s1.A)
claim *A *G secret((N2-1)@s1.B)
```

### B.1.2 Bounded version

Below, we include the CPL description generated from LAEVA in the case of fixed number of sessions:

```
 s1. session  A=a, B=b
 s2. session  A=a, B=I
```

---

```
**asymk**(*D*) : number hash
A : principal
B : principal
I : principal
N1 : number
N2 : number
pbk(principal) : number hash
prk(principal) : number hash
_kpf1(principal) : number hash
a : principal
alg : asym_algo
b : principal
alias pbk = lambda-privk^(alg)(_kpf1)
alias prk = lambda-pubk^(alg)(_kpf1)
process s1.A : 1. {
```

```
  1. -> \%1. : new N1-1
  \%1. -> 2. : send {(a,N1-1)}_(apply-privk^(alg)(_kpf1,b))^(alg)
  2. -> 3. : recv {(<b>,<N1-1>,_x3)}_(apply-pubk^(alg)(_kpf1,a))^(alg)
  3. -> end. : send {_x3}_(apply-privk^(alg)(_kpf1,b))^(alg)
}
(apply-pubk^(alg)(_kpf1,A) = apply-pubk^(alg)(_kpf1,a)
 B = b
 A = a
 N2 = _x3
 alg = alg
 N1 = N1-1
 lambda-privk^(alg)(_kpf1) = lambda-privk^(alg)(_kpf1))

process s1.B : 1. {
  1. -> 2. : recv {(_x1,_x2)}_(apply-pubk^(alg)(_kpf1,b))^(alg)
  2. -> \%4. : new N2-1
  \%4. -> 3. : send {(b,_x2,N2-1)}_(apply-privk^(alg)(_kpf1,_x1))^(alg)
  3. -> end. : recv <{N2-1}_(apply-privk^(alg)(_kpf1,b))^(alg)>
}
(B = b
 apply-pubk^(alg)(_kpf1,B) = apply-pubk^(alg)(_kpf1,b)
 A = _x1
 N2 = N2-1
 alg = alg
 N1 = _x2
 lambda-privk^(alg)(_kpf1) = lambda-privk^(alg)(_kpf1))

process s2.A : 1. {
  1. -> \%1. : new N1-1
  \%1. -> 2. : send {(a,N1-1)}_(apply-privk^(alg)(_kpf1,I))^(alg)
  2. -> 3. : recv {(<I>,<N1-1>,_x3)}_(apply-pubk^(alg)(_kpf1,a))^(alg)
  3. -> end. : send {_x3}_(apply-privk^(alg)(_kpf1,I))^(alg)
}
(apply-pubk^(alg)(_kpf1,A) = apply-pubk^(alg)(_kpf1,a)
 B = I
 A = a
 N2 = _x3
 alg = alg
 N1 = N1-1
 lambda-privk^(alg)(_kpf1) = lambda-privk^(alg)(_kpf1))

process s2.B : 1. {
  1. -> 2. : recv {(_x1,_x2)}_(apply-pubk^(alg)(_kpf1,I))^(alg)
  2. -> \%4. : new N2-1
  \%4. -> 3. : send {(I,_x2,N2-1)}_(apply-privk^(alg)(_kpf1,_x1))^(alg)
  3. -> end. : recv <{N2-1}_(apply-privk^(alg)(_kpf1,I))^(alg)>
}
(B = I
 apply-pubk^(alg)(_kpf1,B) = apply-pubk^(alg)(_kpf1,I)
 A = _x1
 N2 = N2-1
```

```
 alg = alg
 N1 = _x2
 lambda-privk^(alg)(_kpf1) = lambda-privk^(alg)(_kpf1))

assume secret((apply-pubk^(alg)(_kpf1,a))@s1.A)
assume secret(apply-pubk^(alg)(_kpf1,(_x1)@s1.B))
assume secret((apply-pubk^(alg)(_kpf1,b))@s1.B)
claim secret((N1-1)@s1.A)
claim secret((N2-1)@s1.B)
```

# Appendix C

# Protocol Examples

We present here the LAEVA description of the protocols used in section 8.3.

### Needham Schroeder Public Key [NS78]

```
alg : asym_algo
everybody knows alg

A, B: principal
Na, Nb : number

keypair^alg PK, SK (principal)
everybody knows PK

A knows A, B, SK(A)
B knows B, SK(B)
{
  1. A -> B : {A, Na}_(PK(B))^alg
  2. B -> A : {Na, Nb}_(PK(A))^alg
  3. A -> B : {Nb}_(PK(B))^alg
}

s. session* {Na,Nb} A=A, B=B

assume secret (SK(A)@s1.A),
       secret (SK(B)@s1.B),
       secret (SK(B@s1.A)),
       secret (SK(A@s1.B))

claim *A*G secret (SK(A)@s1.A),
      *A*G secret (SK(B)@s1.B),
      *A*G secret (Na@s1.A),
      *A*G secret (Nb@s1.B)
```

### Yahalom [BAN90]

```
A, B, S :       principal
Na, Nb :        number

basetype key
Kab : key

shr (principal,principal) : number secret
alias Kas = shr(A,S)
alias Kbs = shr(B,S)

A knows A, B, S, Kas
B knows B, S, Kbs
S knows S, shr
{
  1. A -> B : A, Na
  2. B -> S : B, { A, Na, Nb }_Kbs
  3. S -> A : { B, Kab, Na, Nb }_Kas, { A, Kab }_Kbs
  4. A -> B : { A, Kab }_Kbs, { Nb }_Kab
}

s. session* {Kab} A=A, B=B, S=S

assume secret (Kas@s.A),
       secret (Kbs@s.B),
       secret (Kas@s.S),
       secret (Kbs@s.S)

claim *A*G secret (Kas@s.A),
      *A*G secret (Kbs@s.B),
      *A*G secret (Kab@s.S)
```

### Otway-Rees [OR87]

```
A, B, S : principal
Kab : number
N, Na, Nb : number

shr (principal,principal) : number secret
alias Kas = shr(A,S)
alias Kbs = shr(B,S)

A knows A, B, Kas
B knows A, B, S, Kbs
S knows S, shr
{
  1. A -> B : N, A, B, {Na, N, A, B}_Kas
  2. B -> S : N, A, B, {Na, N, A, B}_Kas, {Nb, N, A, B}_Kbs
  3. S -> B : N, {Nb, Kab}_Kbs
  4. S -> A : N, {Na, Kab}_Kas
}
```

```
s. session *{Kas, Kbs} A=A, B=B, S=S

assume secret (Kas@s1.A),
       secret (Kbs@s1.B),
       secret (Kas@s1.S),
       secret (Kbs@s1.S)

claim *A*G secret (Kas@s1.A),
      *A*G secret (Kbs@s1.B),
      *A*G secret (Kab@s1.S)
```

### Denning-Sacco Key Distribution with Public Key [DS81]

```
alg : asym_algo
everybody knows alg

A, B : principal
Na, Nb : number

basetype key
Ka : key

keypair^alg SK, PK (principal)
everybody knows PK

A knows A, B, SK(A)
B knows B, SK(B)
{
  1. A -> B : A, {{Ka, Na}_(SK(A))^alg}_(PK (B))^alg
  2. B -> A  : {Nb}_Ka
}

s. session* {Ka, Nb} A=A, B=B

assume secret (SK(A)@s.A),
       secret (SK(B)@s.B),
       secret (SK(A@s.B))

claim *A*G secret (Nb@s.B),
      *A*G secret (Ka@s.A)
```

### Wide-Mouthed-Frog [BAN90] - modified

```
A, B, S : principal
Na : number

basetype key
Kab : key

shr (principal,principal) : number secret
alias Kas = shr(A,S)
```

```
alias Kbs = shr(B,S)

A knows A, B, S, Kas
B knows B, S, Kbs
S knows S, shr
{
  1. A -> B : A, Na
  2. B -> S : B, {Na, A, Kab}_Kbs
  3. S -> A : {Na, B, Kab}_Kas
}

s. session *{A, B} A=A, B=B, S=S

assume secret (Kas@s.A),
       secret (Kbs@s.B),
       secret (shr(A@s.B,S@s.B)),
       secret (Kas@s.S),
       secret (Kbs@s.S)

claim  *A*G secret (Kas@s.S),
       *A*G secret (Kbs@s.S),
       *A*G secret (Kab@s.B)
```

### Kao-Chow [CJ97](6.5.4)

```
A, B, S : principal
Na, Nb : number

basetype key
Kab : key

shr (principal, principal) : number secret
alias Kas = shr(A,S)
alias Kbs = shr(B,S)

A knows A, B, Kas
B knows B, S, Kbs
S knows S, shr
{
 1. A -> S : A, B, Na
 2. S -> B : {A, B, Na, Kab}_Kas, {A, B, Na, Kab}_Kbs
 3. B -> A : {A, B, Na, Kab}_Kas, {Na}_Kab, Nb
 4. A -> B : {Nb}_Kab
}

s. session *{Kab} A=A, B=B, S=S

assume secret (Kas@s1.A),
       secret (Kbs@s1.B),
       secret (shr(A@s1.B,S@s1.B)),
       secret (Kas@s1.S),
```

```
       secret (Kbs@s1.S)

claim *A*G secret (Kas@s1.A),
      *A*G secret (Kbs@s1.B),
      *A*G secret (Kab@s1.S)
```

### Neumann-Stubblebine [NS93]

```
alg : asym_algo
everybody knows alg

A, B, S : principal
Na, Ma, Nb, Mb, Ta, Tb : number

basetype key
Kab : key

shr (principal,principal) : number secret
alias Kas = shr(A,S)
alias Kbs = shr(B,S)

A knows A, B, S, Kas
B knows A, B, S, Kbs
S knows S, shr
{
  1. A -> B : A, Na
  2. B -> S : B, {A, Na, Tb}_Kbs, Nb
  3. S -> A : {B, Na, Kab, Tb}_Kas, {A, Kab,Tb}_Kbs, Nb
  4. A -> B : {A, Kab,Tb}_Kbs, {Nb}_Kab
  5. A -> B : Ma, {A, Kab,Tb}_Kbs
  6. B -> A : Mb, {Ma}_Kab
  7. A -> B : {Mb}_Kab
}

s. session* {Kab} A=A, B=B, S=S

assume secret (Kas@s.A), secret (Kbs@s.B),
       secret (shr(A@s.S,S@s.S)),
       secret (shr(B@s.S,S@s.S))

claim *A*G secret (Kas@s.A),
      *A*G secret (Kbs@s.B),
      *A*G secret (Kab@s.S)
```

### Needham-Schroeder Symmetric Key [NS78]

```
A, B, S : principal
Na, Nb  : number

basetype key
Kas, Kbs, Kab, Kplus : key
```

```
shr (principal,principal) : key secret
alias Kas = shr(A,S)
alias Kbs = shr(B,S)

A knows A, B, S, Kas, Kplus
B knows A, B, S, Kbs, Kplus
S knows S, shr, Kplus
{
  1. A -> S :  A, B, Na
  2. S -> A : {Na,B, Kab, {Kab, A}_Kbs}_Kas
  3. A -> B : {Kab, A}_Kbs
  4. B -> A : {Nb}_Kab
  5. A -> B : {{Nb}_Kplus}_Kab
}

s. session* {Kab,Nb} A=A, B=B, S=S

assume secret (Kas@s.A),
       secret (Kbs@s.B),
       secret (Kas@s.S),
       secret (Kbs@s.S)

claim *A*G secret (Kab@s.S),
      *A*G secret (Nb@s.B),
      *A*G secret (Kas@s.A),
      *A*G secret (Kbs@s.B)
```

### TMN [TMN90, LR97]

```
alg : asym_algo
everybody knows alg

A, B, S : principal

basetype key
Ka, Kb : key

keypair^alg PK, SK (principal)
everybody knows PK

A knows A, B, S
B knows A, B, S
S knows A, B, S, SK(S)
{
  1. A -> S : B, {Ka}_(PK(S))^alg
  2. S -> B : A
  3. B -> S : A, {Kb}_(PK (S))^alg
  4. S -> A : B, {Kb}_Ka
}
```

```
s. session* {Ka} A=A, B=B, S=S

assume secret (SK(S)@s.S)

claim *A*G secret (Ka@s.A),
      *A*G secret (Kb@s.B)
```

### ISO-Symmetric-Key-Two-Pass-Unilateral-Authentication [CJ97](6.2.2)

```
A, B : principal
Text1, Text2, Text3, Nb : number

basetype key
Kab : key

A knows  A, B, Kab
B knows  A, B, Kab
{
  1. B -> A : Nb, Text1
  2. A -> B : Text3,{Nb, B,Text2}_Kab
}

s. session* {Kab, Text2} A=A, B=B

assume secret (Text2@s.A),
       secret (Kab@s.A)

claim *A*G secret (Kab@s.A),
      *A*G secret (Text2@s.A)
```

### Andrew-Secure-RPC [Sat89]

```
A, B : principal
Na, Nb, Nb1, Na1  : number

basetype key
Kab, Kab1, Kplus : key

A knows A, B, Kab, Kplus
B knows A, B, Kab, Kplus
{
  1. A -> B :  A, {Na}_Kab
  2. B -> A : {{Na}_Kplus, Nb}_Kab
  3. A -> B : {{Nb}_Kplus}_Kab
  4. B -> A : {Kab1, Nb1}_Kab
  5. A -> B : {Na1}_Kab1
}

s. session* {Na,Nb,Na1,Nb1} A=A, B=B

assume secret (Kab@s.A)
```

```
claim *A*G secret (Kab@s.A),
      *A*G secret (Kab1@s.B),
      *A*G secret (Na@s.A),
      *A*G secret (Nb@s.B),
      *A*G secret (Na1@s.A),
      *A*G secret (Nb1@s.B)
```

### Woo and Lam [WL94] - modified

```
A, B, S : principal

basetype key
Na, Nb, Kab : key

shr (principal,principal) : number secret
alias Kas = shr(A,S)
alias Kbs = shr(B,S)

A knows A, B, S, Kas
B knows A, B, S, Kbs
S knows S, shr
{
  1. A -> B : A, Na
  2. B -> A : A, Na, B, Nb
  3. A -> B : Na, Nb, {A, B, Na, Nb}_Kas
  4. B -> S : A, B, {A, B, Na, Nb}_Kas, {A, B, Na, Nb}_Kbs
  5. S -> B : {B, Na, Nb, Kab}_Kas, {A, Na, Nb, Kab}_Kbs
  6. B -> A : {B, Na, Nb, Kab}_Kas, {Na, Nb}_Kab
  7. A -> B : {Nb}_Kab
}

s. session *{A, B} A=A, B=B, S=S

assume secret (Kas@s.A),
       secret (Kbs@s.B),
       secret (Kas@s.S),
       secret (Kbs@s.S)

claim  *A*G secret (Kas@s.A),
       *A*G secret (Kbs@s.B),
       *A*G secret (Kab@s.S)
```

### Skeme [Kra96] - modified

```
alg : asym_algo
everybody knows alg

A, B : principal
Na, Nb, Na1, Nb1  : number
```

```
basetype key
Ka, Kb : key

keypair^alg SK, PK (principal)
everybody knows PK

A knows A, B, SK(A)
B knows A, B, SK(B)
{
  1. A -> B : {A,Ka}_(PK (B))^alg,Na
  2. B -> A : {Kb}_(PK (A))^alg,Nb,{Na,Nb,B,A}_Ka
  3. A -> B : {Nb,Na,A,B,Ka}_Kb
  4. B -> A : {Nb1}_Kb
  5. A -> B : {Na1}_Ka
}

s. session* {Na} A=A, B=B

assume secret (SK(A)@s.A),
       secret (SK(B)@s.B)

claim *A*G Secret (Nb1@s.B),
      *A*G Secret (Na1@s.A),
      *A*G Secret (Kb@s.B),
      *A*G Secret (Ka@s.A)
```

# Bibliography


[AB01] M. Abadi and B. Blanchet. Secrecy types for asymmetric communication. In F. Honsell and M. Miculan, editors, *Proceedings of FOSSACS'01 - 4th International Conference on Foundations of Software Science and Computation Structures, Genova, Italy*, volume 2030 of *LNCS*. Springer, April 2001.

[AB02] M. Abadi and B. Blanchet. Analyzing security protocols with secrecy types and logic programs. In *Proceedings of POPL'02 - 29th Annual ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages, Portland, Oregon*. ACM Press, January 2002.

[Aba97] M. Abadi. Secrecy by typing in security protocols. In M. Abadi and T. Ito, editors, *Proceedings of TACS'97 - Third International Symposium on Theoretical Aspects of Computer Software, Sendai, Japan*, volume 1281 of *LNCS*. Springer, September 1997.

[AC02] R. M. Amadio and W. Charatonik. On name generation and set-based analysis in the dolev-yao model. In L. Brim, P. Jancar, M. Kretinsky, and A. Kucera, editors, *Proceedings of CONCUR'02 - 13th International Conference on Concurrency Theory, Brno, Czech Republic*, volume 2421 of *LNCS*. Springer, August 2002.

[AC04] A. Armando and L. Compagna. Satmc: a sat-based model checker for security protocols. In *Proceedings of JELIA'04 - 9th European Conference on Logics in Artificial Intelligence*, number 3229 in LNCS. Springer-Verlag, 2004.

[AD94] R. Alur and D. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126, 1994.

[AFH91] R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. In *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*. ACM Press, 1991.

[AG99] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The Spi calculus. *Journal of Information and Computation*, 148(1), 1999. An extended abstract appeared in the *Proceedings of the Fourth ACM Conference on Computer and Communications Security (Zürich, April 1997)*.

[AL00] R. M. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In C. Palamidessi, editor, *Proceedings of CONCUR'00 - 11th International Conference on Concurrency Theory, University Park, PA, USA*, volume 1877 of *LNCS*. Springer, August 2000.

[ALV02] R. M. Amadio, D. Lugiez, and V. Vanackere. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 290(1), 2002. Also INRIA Research Report 4147, March 2001.

[AN96] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transaction on Software Engineering*, 22(1), January 1996.

[AVC+02] A. Armando, L. Vigneron, C. Compagna, M. Bouallagui, M. Rusinowitch, and Y. Chevalier. The AVISS security protocol analysis tool. In Ed Brinksma and K. G. Larsen, editors, *Proceedings of CAV'02 - 14th International Conference of Computer Aided Verification, Copenhagen, Denmark*, LNCS, July 2002.

[BAN90] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1), February 1990.

[BB01] M. Bidoit and A. Boisseau. Algebraic abstractions. In M. Cerioli and G. Reggio, editors, *Proceedings of WADT'01 - 15th International Workshop on Algebraic Development Techniques, Genova, Italy*, volume 2267 of *LNCS*. Springer, April 2001.

[BBC+97] B. Barras, S. Boutin, C. Cornes, J. Courant, J.C. Filliatre, E. Giménez, H. Herbelin, G. Huet, C. Munoz, C. Murthy, C.Parent, C. Paulin, A. Saibi, and B. Werner. The Coq proof assistant reference manual. Technical Report 0203, INRIA, August 1997.

[BEL04a] L. Bozga, C. Ene, and Y. Lakhnech. On the existence of an effective and complete inference system for cryptographic protocols. In *Proceedings of FOSSACS'04 - Conference on Foundations of Software Science and Computation Structures, Barcelon, Spain*, number 2987 in LNCS, April 2004.

[BEL04b] L. Bozga, C. Ene, and Y. Lakhnech. A symbolic decision procedure for cryptographic protocols with time stamps. In *Proceedings of CONCUR'04 - 15th International Conference on Concurrency Theory, London, United Kingdom*, August 2004.

[BG01] M. Bellare and S. Goldwasser. Lecture notes on cryptography, 2001. Course notes of summer course Cryptography and Computer Security at MIT, 1996–2001.

[BL95] A. Bouajjani and Y. Lakhnech. Temporal logic + timed automata: expressiveness and decidability. In I. Lee and S. A. Smolka, editors, *Proceedings of CONCUR'95: 6th International Conference on Concurrency Theory, Philadelphia, PA, USA*, volume 962 of *LNCS*. Springer, August 1995.

[Bla01] B. Blanchet. Abstracting cryptographic protocols by prolog rules (invited talk). In P. Cousot, editor, *Proceedings of SAS'01 - 8th International Static Analysis Symposium, Paris, France*, volume 2126 of *LNCS*. Springer, July 2001.

[BLP02a] L. Bozga, Y. Lakhnech, and M. Périn. L'outil de vérification hermes. Technical report, Projet EVA, `http://www-eva.imag.fr/`, may 2002.

[BLP02b] L. Bozga, Y. Lakhnech, and M. Périn. Pattern-based abstraction for verifying secrecy in protocols. Technical report, Projet EVA, `http://www-eva.imag.fr/`, December 2002.

[BLP03a] L. Bozga, Y. Lakhnech, and M. Périn. Abstract interpretation for secrecy using patterns. In H. Garavel and J. Hatcliff, editors, *Proceedings of TACAS'03 - 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Warsaw, Poland*, volume 2619 of *LNCS*. Springer, April 2003.

[BLP03b] L. Bozga, Y. Lakhnech, and M. Périn. Hermes: An automatic tool for verification of secrecy in security protocols. In Jr. W. A. Hunt and F. Somenzi, editors, *Proceedings of CAV'03 - 15th International Conference on Computer Aided Verification, Boulder, Colorado*, volume 2725 of *LNCS*. Springer-Verlag, July 2003.

[BMV03] D. Basin, S. Mödersheim, and L. Viganò. An on-the-fly model-checker for security protocol analysis. In *Proceedings of ESORICS'03 - 13th European Symposium on Research in Computer Security*, number 2808 in LNCS. Springer-Verlag, 2003.

[BN98] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, New York, 1998.

[Boi03] A. Boisseau. *Abstractions pour la vérification de propriétés de sécurité de protocoles cryptographiques*. PhD thesis, LSV - ENS Cachan, September 2003.

[Bol96] D. Bolignano. An approach to the formal verification of cryptographic protocols. In *Proceedings of CCS'96 - 3rd ACM Conference on Computer and Communications Security, New Delhi, India*. ACM Press, March 1996.

[Bol97] D. Bolignano. Towards a mechanization of cryptographic protocol verification. In O. Grumberg, editor, *Proceedings of CAV'97 - 9th International Conference on Computer-Aided Verification, Haifa, Israel*, volume 1254 of *LNCS*. Springer, June 1997.

[Bol98] D. Bolignano. Integrating proof-based and model-checking techniques for the formal verification of cryptographic protocols. In M. Y. Vardi A. J. Hu,

editor, *Proceedings of CAV'98 - 10th International Computer Aided Verification Conference, Vancouver, BC, Canada*, volume 1427 of *LNCS*. Springer, June 1998.

[Bol00] D. Bolignano. Using abstract interpretation for the safe verification of security protocols. In S. Brookes, A. Jung, and M. Mislove, editors, *Electronic Notes in Theoretical Computer Science*, volume 20. Elsevier, 2000.

[Bor01] M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Proceedings of ICALP'01 - 28th Colloquium on Automata, Languages and Programming*, volume 2076 of *LNCS*. Springer, July 2001.

[BP98] G. Bella and L. C. Paulson. Mechanizing BAN Kerberos by the inductive method. In A. J. Hu and M. Y. Vardi, editors, *Proceedings of CAV'98 - 10th International Conference on Computer-Aided Verification*, Vancouver, B.C., Canada, June 1998. Springer-Verlag LNCS 1427.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of POPL'77 - 4th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Los Angeles, California*. ACM Press, January 1977.

[CC92a] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2–3), 1992.

[CC92b] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4), 1992.

[CCM01] H. Comon, V. Cortier, and J. Mitchell. Tree automata with one memory, set constraints, and ping-pong protocols. In J.v. Leeuwen F. Orejas, P.G. Spirakis., editor, *Proceedings of ICALP'01 - 28th International Colloquium on Automata, Languages and Programming, Crete, Greece*, volume 2076 of *LNCS*. Springer, July 2001.

[Che03] Y. Chevalier. *Résolution de problèmes d'accessibilité pour la compilation et la validation des protocoles cryptographiques*. PhD thesis, LORIA - INRIA Nancy, December 2003.

[CJ97] J. A. Clark and J. L. Jacob. A survey of authentication protocol literature: Version 1.0., November 1997.

[CJM97] E. Clarke, S. Jha, and W. Marrero. Model checking for security protocols. Technical report, The Pennsylvania State University, 1997.

[CJM98] E. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In D. Gries and W.-P. de Roever, editors, *Proceedings of PROCOMET'98 - IFIP Working Conference on Programming Concepts and Methods*. IFIP, 1998.

[CLC04] H. Comon-Lundh. and V. Cortier. Security properties: Two agents are sufficient. *Science of Computer Programming*, 50(1-3), March 2004. Special issue on 12th European symposium on programming (ESOP'03).

[CMR01] V. Cortier, J. Millen, and H. Rueß. Proving secrecy is easy enough. In *Proceedings of CSFW'01 - 14th IEEE Computer Security Foundations Workshop, Cape Breton, Nova Scotia, Canada*. IEEE Computer Society Press, June 2001.

[Coh00] E. Cohen. Taps: A first-order verifier for cryptographic protocols. In *Proceedings of CSFW'00 - 13th IEEE Computer Security Foundations Workshop*. IEEE Computer Society, 2000.

[Com91] H. Comon. Disunification: A survey. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*. MIT-Press, 1991.

[Cor02] V. Cortier. L'outil de vérification securify. Technical report, Projet EVA, `http://www-eva.imag.fr/`, November 2002.

[Cor03] V. Cortier. *Vérification automatique des protocoles cryptographiques*. PhD thesis, LSV - ENS Cachan, March 2003.

[Cou90] P. Cousot. Methods and logics for proving programs. In *Handbook of Theoretical Computer Science, Volume B: Formal Methods and Semantics*. Elsevier Science Publishers B. V., 1990.

[CS02] H. Comon and V. Shmatikov. Is it possible to decide whether a cryptographic protocol is secure or not? *Journal of Telecommunications and Information Technology*, 4, 2002.

[DEK82] D. Dolev, S. Even, and R. M. Karp. On the security of ping-pong protocols. In D. Chaum, R. L. Rivest, , and A. T. Sherman, editors, *Proceedings of CRYPTO'82 - Advances in Cryptology*. Plenum Publishing, 1982.

[Dij76] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[DLMS99] N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In N. Heintze and E. Clarke, editors, *Proceedings of FMSP'99 - Workshop on Formal Methods and Security Protocols, Trento, Italy*, July 1999.

[DMR00] G. Denker, J. Millen, and H. Ruess. The CAPSL integrated protocol environment. Technical Report SRI-CSL-2000-02, Computer Science Laboratory, SRI International, October 2000.

[DP04] G. Delzanno and P.Ganty. Automatic verification of time sensitive cryptographic protocols. In *Proceedings of TACAS'04 - 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 2988 in LNCS, 2004.

[DS81] D. E. Denning and G. M. Sacco. Timestamps in key distribution systems. *Communications of the ACM*, 24(8), August 1981.

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

[EG83] S. Even and O. Goldreich. On the security of multi-party ping pong protocols. Technical Report 285, Israel Institute of Technology, 1983.

[ES00] N. Evans and S. Schneider. Analyzing time dependent security properties in CSP using PVS. In *Proceedings of ESORICS'00 - 6th European Symposium on Research in Computer Security*, volume 1895 of *LNCS*, 2000.

[FHG88] F. J. T. Fábrega, J. C. Herzog, and J. D. Guttman. Strand Spaces: Why is a security protocol correct? In *Proceedings of SP'98 - IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 1988.

[GJ01] A. Gordon and A. Jeffrey. Authenticity by typing for security protocols. In *Proceedings of CSFW'01 - 14th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, June 2001.

[GK00] T. Genet and F. Klay. Rewriting for cryptographic protocol verification. In D. McAllester, editor, *Proceedings of CADE'00 - 17th International Conference on Automated Deduction, Pittsburgh, PA, USA*, volume 1831 of *LNCS*. Springer, 2000.

[GL00] J. Goubault-Larrecq. A method for automatic cryptographic protocol verification. In J. Rolim, editor, *Proceedings of 15 IPDPS 2000 Workshops on Parallel and Distributed Processing, Cancun, Mexico*, volume 1800 of *LNCS*. Springer, 2000.

[GL02a] J. Goubault-Larrecq. La syntaxe et la sémantique du langage de spécification eva. Technical report, Projet EVA, `http://www-eva.imag.fr/`, November 2002.

[GL02b] J. Goubault-Larrecq. Outils CPV et CPV2. Technical report, Projet EVA, `http://www-eva.imag.fr/`, May 2002.

[Gon92] Li Gong. A security risk of depending on synchronized clocks. *ACM Operating Systems Review*, 26(1), 1992.

[HNSY92] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model-checking for real-time systems. In *Seventh Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1992.

[Hoa69] C. A. R. Hoare. An Axiomatic Basis of Computer Programming. *Communications of the ACM*, 12, 1969.

[Hoa85] C. A. R. Hoare. *Communications Sequential Processes*. Prentice-Hall International, 1985.

[IT94] ITU-T. Recommendation Z.120. message sequence charts. Technical Report Z-120, International Telecommunication Union – Standardization Sector, Genève, 1994.

[JK91] J.-P. Jouannaud and C. Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*. MIT-Press, 1991.

[JLP04] R. Janvier, Y. Lakhnech, and M. Périn. Correctness of abstract-based verification tool by proof concretization, 2004.

[JM01] F. Jacquemard and D. Le Métayer. Langage de spécification de protocoles cryptographiques de eva: syntaxe concrète. Technical Report 1, Projet EVA, `http://www-eva.imag.fr/`, November 2001.

[JPL03] R. Janvier, M. Périn, and Y. Lakhnech. Coq certification for verification with hermes tool. Technical report, Projet EVA, `http://www-eva.imag.fr/`, November 2003.

[JRV00] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and verifying security protocols. In A. Voronkov M. Parigot, editor, *Proceedings of LPAR'00 - 7th International Conference on Logic for Programming and Automated Reasoning, Reunion Island, France*, volume 1955 of *LNCS*. Springer, November 2000.

[KD97] R. A. Kemmerer and Z. Dang. Using the ASTRAL model checker for cryptographic protocol analysis. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, August 1997.

[KMM94] R. Kemmerer, C. Meadows, and J. Millen. Three system for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.

[Kra96] H. Krawczyk. SKEME: A versatile secure key exchange mechanism for the Internet. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS '96)*, San Diego, California, February 1996. Internet Society.

[Low95] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3), November 1995.

[Low96] G. Lowe. Breaking and fixing the needham-schroeder public-key protocol using FDR. In T. Margaria and B. Steffen, editors, *Proceedings of TACAS'96 - Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, Passau, Germany*, volume 1055 of *LNCS*. Springer, March 1996.

[Low97a] G. Lowe. Casper: A compiler for the analysis of security protocols. In *Proceedings of CSFW'97 - 10th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1997.

[Low97b] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of CSFW'97 - 10th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1997.

[Low98] G. Lowe. Towards a completeness result for model checking of security protocols. In *Proceedings of CSFW'98 - 11th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 1998.

[LR97] G. Lowe and B. Roscoe. Using CSP to detect errors in the TMN protocol. *IEEE Transactions on Software Engineering*, 23(10), October 1997.

[Maz04] L. Mazaré. Using unification for opacity properties. In *Proceeding of WITS'04 - 5th Workshop on Issues in the Theory of Security*, 2004.

[Mea96] C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2), February 1996.

[Mea00] C. Meadows. Invariant generation techniques in cryptographic protocol analysis. In *Proceedings of CSFW'00 - 13th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, July 2000.

[Mea03] C. Meadows. Formal methods for cryptographic protocol analysis: Emerging issues and trends. *IEEE Journal on Selected Areas in Communication*, 21(1), January 2003.

[Min61] M.L. Minsky. Recursive unsolvability of post's problem of "tag" and other topics in the theory of turing machines. *Annals of Mathematics, Second Series*, 74(3), 1961.

[MMS97] J.C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using mur$\phi$. In *Proceedings of SP'97 - IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 1997.

[Mon99] D. Monniaux. Abstracting cryptographic protocols with tree automata. In A. Cortesi and G. File, editors, *Proceedings of SAS'99 - 6th International Static Analysis Symposium, Venice, Italy*, volume 1694 of *LNCS*. Springer, September 1999.

[MS01] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of CCS'01 - 8th ACM Conference on Computer and Communications Security*. ACM Press, 2001.

[MvOV96] A. J. Menezes, P. C. van Oorschot, and S. A. Vanston. *Handbook of Applied Cryptography*. CRC Press, 1996.

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

[NS93] B.C. Neuman and S.G. Stubblebine. A note in the use of timestamps as nonces. *Operating Systems Review*, 27(2), April 1993.

[OR87] D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1), January 1987.

[ORS92] S. Owre, J. M. Rushby, , and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *Proceedings of CADE'92 - 11th International Conference on Automated Deduction*, volume 607 of *LNAI*. Springer, June 1992.

[Pau94] L.C. Paulson. *Isabelle, A Generic Theorem Prover*, volume 828 of *LNCS*. Springer-Verlag, 1994.

[Pau97] L. Paulson. Proving properties of security protocols by induction. In *Proceedings of CSFW'97 - 10th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1997.

[Pra65] D. Prawitz. *Natural Deduction: A Proof-Theoretical Study*. Almquivst & Wikseel, 1965.

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

[RSG+01] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *Modelling and Analysis of Security Protocols*. Addison-Wesley, 2001.

[RT01] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proceedings of CSFW'01 - 14th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, June 2001.

[Rus03] Michael Rusinowitch. Automated analysis of security protocols. In G. Vidal, editor, *Electronic Notes in Theoretical Computer Science*, volume 86. Elsevier, 2003.

[Sat89] M. Satyanarayanan. Integrating security in a large distributed system. *ACM Transactions on Computer Systems*, 7(3), August 1989.

[Sch86] A. Schrijver. *Theory of linear and integer programming*. Wiley-Interscience Series in Discrete Mathematics. John Wiley & Sons Ltd, 1986.

[Sch94] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. Wiley, 1994.

[Sch97] S. Schneider. Verifying authentication protocols with CSP. In *Proceedings of CFSW'97 - 10th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1997.

[Son99] D. X. Song. Athena: A new efficient automatic checker for security protocol analysis. In *Proceedings of CSFW'99 - 12th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1999.

[THG98] J. Thayer, J. Herzog, and J. Guttman. Honest ideals on strand spaces. In *Proceedings of CSFW'98 - 11th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1998.

[TMN90] M. Tatebayashi, N. Matsuzaki, and N. Newman. Key distribution protocol for digital mobile communication systems. In G. Brassard, editor, *Advances in Cryptology*, volume 435 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[Tur03] M. Turuani. *Sécurité des protocoles cryptographiques : décidabilité et complexité*. PhD thesis, LORIA - INRIA Nancy, December 2003.

[Ven87] K. N. Venkataraman. Decidability of the purely existential fragment of the theory of term algebras. *Journal of the ACM*, 34(2), April 1987.

[Wei99] C. Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In H. Ganzinger, editor, *Proceedings of CADE'99 - 16th International Conference on Automated Deduction*, volume 1632 of *LNCS*. Springer, July 1999.

[WL92] T. Y. C. Woo and S. S. Lam. Authentication for distributed systems. *Computer*, 25(1), January 1992.

[WL94] T. Y. C. Woo and S. S. Lam. A lesson in authentication protocol design. *ACM Operating Systems Review*, 28(3), 1994.