Modeling and analysis of advanced cryptographic primitives and security protocols in Maude-NPA

Advisor: Santiago Escobar Román

Student: Damián Aparicio Sánchez

Thesis Submitted for the degree of Philosophiae Doctor

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

VRAIN Universitat Politècnica de València, Spain

October 26, 2022

Abstract

The Maude-NPA crypto tool is a specialized model checker for cryptographic security protocols that take into account the algebraic properties of the cryptosystem. In the literature, additional crypto properties have uncovered weaknesses of security protocols and, in other cases, they are part of the protocol security assumptions in order to function properly. Maude-NPA has a theoretical basis on rewriting logic, equational unification, and narrowing to perform a backwards search from an insecure state pattern to determine whether or not it is reachable. Maude-NPA can be used to reason about a wide range of cryptographic properties, including cancellation of encryption and decryption, Diffie-Hellman exponentiation, exclusive-or, and some approximations of homomorphic encryption.

In this thesis, we consider new cryptographic properties, either as part of security protocols or to discover new attacks. We have also modeled different families of security protocols, including Distance Bounding Protocols or Multiparty key agreement protocols. And we have developed new protocol modeling techniques to reduce the time and space analysis effort. This thesis contributes in several ways to the area of cryptographic protocol analysis and many of the contributions of this thesis can be useful for other crypto analysis tools.

Resumen

La herramienta criptográfica Maude-NPA es un verificador de modelos especializado para protocolos de seguridad criptográficos que tienen en cuenta las propiedades algebraicas de un sistema criptográfico. En la literatura, las propiedades criptográficas adicionales han descubierto debilidades de los protocolos de seguridad y, en otros casos, son parte de los supuestos de seguridad del protocolo para funcionar correctamente. Maude-NPA tiene una base teórica en la *rewriting logic*, la unificación ecuacional y el *narrowing* para realizar una búsqueda hacia atrás desde un patrón de estado inseguro para determinar si es alcanzable o no. Maude-NPA se puede utilizar para razonar sobre una amplia gama de propiedades criptográficas, incluida la cancelación del cifrado y descifrado, la exponenciación de Diffie-Hellman, el *exclusive-or* y algunas aproximaciones del cifrado homomórfico.

En esta tesis consideramos nuevas propiedades criptográficas, ya sea como parte de protocolos de seguridad o para descubrir nuevos ataques. También hemos modelado diferentes familias de protocolos de seguridad, incluidos los *Distance Bounding Protocols or Multi-party key agreement* protocolos. Y hemos desarrollado nuevas técnicas de modelado para reducir el coste del análisis en protocolos con tiempo y espacio. Esta tesis contribuye de varias maneras al área de análisis de protocolos criptográficos y muchas de las contribuciones de esta tesis pueden ser útiles para otras herramientas de análisis criptográfico.

Resum

L'eina criptogràfica Maude-NPA és un verificador de models especialitzats per a protocols de seguretat criptogràfics que tenen en compte les propietats algebraiques d'un sistema criptogràfic. A la literatura, les propietats criptogràfiques addicionals han descobert debilitats dels protocols de seguretat i, en altres casos, formen part dels supòsits de seguretat del protocol per funcionar correctament. Maude-NPA té una base teòrica a la *rewriting logic*, la unificació equacional i *narrowing* per realitzar una cerca cap enrere des d'un patró d'estat insegur per determinar si és accessible o no. Maude-NPA es pot utilitzar per raonar sobre una àmplia gamma de propietats criptogràfiques, inclosa la cancel · lació del xifratge i desxifrat, l'exponenciació de Diffie-Hellman, el *exclusive-or* i algunes aproximacions del xifratge homomòrfic.

En aquesta tesi, considerem noves propietats criptogràfiques, ja sigui com a part de protocols de seguretat o per descobrir nous atacs. També hem modelat diferents famílies de protocols de seguretat, inclosos els *Distance Bounding Protocols o Multi-party key agreement* protocols. I hem desenvolupat noves tècniques de modelització de protocols per reduir el cost de l'anàlisi en protocols amb temps i espai. Aquesta tesi contribueix de diverses maneres a làrea de l'anàlisi de protocols criptogràfics i moltes de les contribucions daquesta tesi poden ser útils per a altres eines danàlisi criptogràfic.

Acknowledgments

I will start with “How a boy like me ended up in a place like this?”,

For many years, in my mind, I never considered doing a degree, much less a Ph.D., but my curiosity and my passion for computing science always requested me to know more. It has been a hard and long journey, that made me grow personally and professionally, and it has really tested me, but meeting amazing people in the computer world has made me go ahead and finish this journey successfully. For this reason, I am expressing my gratitude to those who helped in the achievement of this Ph.D. thesis.

First, my most sincere thanks to my thesis supervisor Santiago Escobar for his continued help and support. Your passion for research has been passed on to me over these years. You shared with me the world of formal cybersecurity and now it is part of me. The one who knows the most and at the same time is humble, dedicated and honest, to keep looking deep inside people and bring out the best in them. This journey would not have been made it without you.

I am indebted to Professor Jose Meseguer and Catherine Meadows. Your vast knowledge and experience have been enriching. Thanks for giving me the opportunity to collaborate with you in the process of my thesis. On the other hand, thanks to Professor Marco Comini for giving me the chance to improve my formation in my research stay, to be friendly support and make the city of Udine like home.

I would also like to express my gratitude to the people who have been part of MiST and ELP research groups. In particular, Professor María Alpuente, to be a strong computing science female reference and Professor Salvador Lucas

for his effusive passion in his improvement of knowledge. I would especially like to thank my lab mates Adrián, Carlos, David, Josep, Sergio, and Tama for sharing all these good times inside and outside work, These people started as colleagues and have become my friends. Also, an affectionate mention to Sergio for going together through the journey of the thesis.

And to finish, thanks to my mother and sister to help me in so different ways and thanks to the person most important in my life, Laia. You always have been inspiring to me, being a skilled, intelligent, and strong woman. For continuing to find us more challenges together. I love you...

Damián Aparicio-Sánchez
Valencia, October 2022

Contents

List of Figures

List of Tables

Part I

Introduction and Objectives

CHAPTER 1

Introduction

The Maude-NPA is a cryptographic tool for analyzing security protocols that takes into account the algebraic properties and a wide range of cryptographic properties like cancellation of encryption and decryption, Diffie-Hellman exponentiation, exclusive-or, and some approximations of homomorphic encryption. In this thesis, we have focused on improving the capabilities of the Maude-NPA tool by implementing new cryptographic properties and studying the results. The Maude-NPA tool has a theoretical basis on rewriting logic, narrowing, and equational unification to perform a backwards search from an attack state pattern to determine whether or not it is reachable.

We give an overview of the methods used in this thesis for analyzing cryptographic protocols. First, in Section 1.1, we recall some relevant algebraic properties of cryptographic operators, used in protocols in this thesis. Second, in Section 1.2, we describe the different characteristics of some families of communication cryptographic protocols. Third, in Section 1.3, we consider several improvements to the modeling process for protocols.

1.1 Cryptographic properties

The equational unification of two terms is of special relevance to many areas in computer science, including logic programming. It consists of finding a sub-

stitution that, when applied to both terms, makes them equal modulo some equational properties. Several algorithms have been developed in the literature for specific equational theories, such as associative-commutative symbols, exclusive-or, Diffie-Hellman, or Abelian Groups (see [15]).

Maude 3.2.1 offers quite sophisticated symbolic capabilities (see [90] and references therein). Among these symbolic features, equational unification [35] is a twofold achievement. On the one hand, Maude provides an order-sorted equational unification command for any combination of symbols having any combination of associativity, commutativity, and identity [46]. This is remarkable since there is no other system with such an advanced unification algorithm. On the other hand, a narrowing-based equational unification algorithm relying on the concept of the *variants* [38] of a term is also available. A variant of a term t is a pair consisting of a substitution σ and the canonical form of $t\sigma$. Narrowing was proved to be complete for unification in [68], but variant-based unification is decidable when the equational theory satisfies the *finite variant property* [38, 60]. The finite variant property has become an essential property in some research areas, such as cryptographic protocol analysis, where Maude-NPA [55], Tamarin [45] and AKiSS [17] rely on the different unification and variant features of Maude.
In the following subsections, we focus on some algebraic properties relevant to the cryptographic protocols of this thesis

1.1.1 Lists-Associativity

A list or sequence of elements is defined in computer science as a data type that represents a finite number of values of (usually) the same type, where a value may occur more than once. In many programming paradigms, a list is defined by specific syntax and semantics for list operations such as:

- a symbol for denoting an empty list;
- a symbol for knowing whether or not a list is empty;
- a symbol for prepending an element to a list;
- a symbol for appending an element to a list;
- a symbol for obtaining the first element of a list;
- a symbol for removing the first element of a list and

- a symbol for obtaining the element at a given index in the list.

For different programming languages, a list can often be constructed by either writing the items in sequence, separated by commas, colons, semicolons, and/or spaces, within a pair of delimiters such as parentheses "( )", square brackets "[ ]", curly brackets "{ }", or angle brackets "< >".

There are algorithms where the use of lists (or ordered elements) are relevant to their proper functioning such as encryption algorithms. A block cipher is a deterministic algorithm in cryptography that operates on a sequence of blocks. It takes a block of plaintext bits from the sequence of blocks and generates a block of ciphertext bits, generally of the same size. The size of each block in the sequence is fixed by the algorithm. The choice of block size does not directly affect the strength of the encryption scheme. The strength of the block cipher depends on the key length. The most used block cipher is Advanced Encryption Standard (AES). It uses mathematic operations, e.g. bitwise rotation, to transform a block of plaintext into a ciphertext block. The AES uses a symmetric-key algorithm, that is the same key is used for both encrypting and decrypting of the payload.

A list can be expressed by binary operators that may satisfy the associative property $(a * b) * c = a * (b * c)$. It reorders or shifts the parentheses in an expression without modifying the meaning. In the area of computer science, an operator with the associativity property can group elements arbitrarily. A binary operator for lists may also satisfy the identity property $x * identity = x$ where *identity* is the empty list.

In Maude, the associativity and identity properties are used as follows. We define a list with `identity` as the empty list and `++` as an infix concatenation symbol. The symbol is associative with the identity symbol `identity`. We define three auxiliary constants `a`, `b`, and `c` of sort `Elem`. We represent the properties of associativity and identity with the labels `[assoc id]` in the corresponding `_++_` symbol.

```
fmod ASSOC-ID is
  sort Elem List .
  subsort Elem < List .
  ops a b c : -> Elem .
  op identity : -> List .
  op _++_ : List List -> List [assoc id: identity] .
endfm
```

Lists are used in this thesis as follows. In Chapter 2, where authenticated USB devices are analyzed, the associativity and identity properties are used in

the Yubikey & YubiHSM APIs for event lists. Every time a new event occurs, it is inserted as a new element at the end of a new data structure for the event list. The leftmost elements are the oldest ones, whereas the rightmost elements are the newest. When we want to check whether event e_1 has occurred before event e_2, we can express this with the event list L_1 ++ e_1 ++ L_2 ++ e_2 ++ L_3, where any of the L_i could be an empty list.

1.1.2 Exclusive-OR

In cryptography, a method for message encryption is based on the simple exclusive-or cipher, which is a type of additive cipher. The exclusive disjunction (XOR) operation, sometimes called modulus 2 addition, is denoted by the $\oplus$ symbol and satisfies the following four properties:

$$
\begin{aligned}
A \oplus (B \oplus C) &= (A \oplus B) \oplus C && \text{(associativity)} \\
A \oplus B &= B \oplus A && \text{(commutativity)} \\
A \oplus \emptyset &= A && \text{(identity element is } \emptyset\text{)} \\
A \oplus A &= \emptyset && \text{(self-cancellation)}
\end{aligned}
$$

1. Associativity. Where reordering or shifting the parentheses does not modify the meaning.

2. Commutativity. Where swapping elements does not change the meaning.

3. Identity. There exists an element z that does not disrupt the meaning of any other element x.

4. Self-cancellation. Each element$x$ cancels with itself.

A payload can be encrypted by applying the bitwise XOR operator to every piece using a given key. To decrypt it, just reapplying the XOR function with the same given key will remove the cipher.

The main reason why XOR is so common in cryptography is because of its perfectly-balanced property: Given a truly random key, the ciphertext is equally likely to be either 0 or 1. The first vulnerability is that when the key is not truly random the ciphertext is very similar to the original plaintext. The second vulnerability, called known-plaintext attack, allows an attacker to obtain a cipher key if it knows the original plaintext and the

ciphertext and is based on the property $plaintext \oplus ciphertext = key$ where $ciphertext = plaintext \oplus key$. The third vulnerability, called the malleability property, is the idea of flipping arbitrary bits in the decrypted plaintext by manipulating the ciphertext, making it possible for an intruder to produce a different decrypted plaintext.

In Maude, the exclusive-or cipher is used as follows. We define three auxiliary constants `a`, `b`, and `c` of sort `Elem`. We represent the properties of associativity and commutativity of exclusive-or with the labels `[assoc comm]` in the corresponding `_*_` symbol. The second equation is necessary for coherence modulo *AC* (see [2]).

```
fmod EXCLUSIVE-OR is
  sorts Elem Xor .
  subsort Elem < Xor .
  ops a b c : -> Elem .
  op identity : -> Xor .
  op _*_ : Xor Xor -> Xor [assoc comm] .
  vars X Y Z U V : [Xor] .
  eq X * X = mt    [variant] .
  eq X * X * Z = Z [variant] .
  eq X * identity = X    [variant] .
endfm
```

The attribute `variant` specifies that these equations will be used for variant-based unification. Since this theory has the finite variant property (see [38,60]), given the term `X * Y` .

The exclusive-or cipher is used in this thesis in the following chapters. In Chapter 2, where the formal specification of YubiKey and YubiHSM are explained, the authenticated encryption with associated data (AEAD) is modeled using exclusive-or cipher. It is necessary to verify the property: *If the intruder has access to the server running YubiKey, where the YubiHSM AES keys are generated, then it is able to obtain plaintext in the clear.*

In Chapter 4, where several distance-bounding-protocols are explained, the exclusive-or cipher is used in the Brands-Chaum, MAD (Mutual Authentication with Distance-Bounding), Meadows's Protocols and Swiss-knife protocols. In Chapter 5, where several protocols using physical properties are explained, the exclusive-or cipher is used in the Brands-Chaum protocol. In Chapter 6, focused on the computation of most general equational unifiers, the exclusive-or cipher is used in the experiments.

1.1.3 Diffie-Hellman

The Diffie-Hellman key exchange protocol is a famous algorithm for securely exchanging cryptographic keys over an insecure channel. Ralph Merkle first created it as one of the public-key protocols. The work published in 1976 by Whitfield Diffie and Martin Hellman is the earliest practical example of public key exchange implemented within the field of cryptography. The U.S. Patent 4,200,770 from 1977, now expired, presents the well-known algorithm, credited to Hellman, Diffie, and Merkle as inventors.

Before Diffie-Hellman, when two parties required secure encrypted communication, the exchange used keys had to be provided by some secure physical means. The Diffie–Hellman key exchange algorithm allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure channel. This key can then be used to encrypt subsequent communications using a symmetric-key cipher.

The Diffie–Hellman algorithm provides a general mechanism for a variety of authenticated protocols and is used to provide forward secrecy in the ephemeral modes of the Transport Layer Security (TLS) protocol, referred to as EDH or DHE depending on the cipher suite.

A general description of the protocol is as follows. Alice and Bob agree on a finite cyclic group G of order n and a generating element g in G. Note that elements in the group G are written multiplicatively, e.g. $n_1 \cdot n_2$.

1. Alice chooses a random number n_a with $1 < n_a < n$, then sends g^{n_a} of G to Bob.

2. Bob chooses a random number n_b with $1 < n_b < n$, then sends g^{n_b} of G to Alice.

3. Alice calculates $(g^{n_b})^{n_a} = g^{n_b \cdot n_a}$ of G.

4. Bob computes the element $(g^{n_a})^{n_b} = g^{n_a \cdot n_b}$ of G.

After these messages, Alice and Bob have in their knowledge the shared secret key of the group $g^{n_a \cdot n_b} = g^{n_b \cdot n_a}$. At this moment, with the secret key shared in group G, it is possible to have secure communication between honest participants.

One of the disadvantages of Diffie-Hellman is that it does not support authentication for honest participants in communication. It is highly vulnerable to a Man-In-The-Middle attack (MITM), giving an attacker the possibility to

modify the communications between two honest and making them believe they are talking directly to each other, when in fact the messages are manipulated by the attacker.

The modular exponentiation property typical of Diffie-Hellman protocols is represented in Maude as follows. We define three auxiliary constants `a`, `b`, and `c` of sort `Elem`. The exponentiation operator is represented by the symbol `exp` and the multiplicative operator $\cdot$ is represented by the symbol `*`, which is an associative-commutative symbol so that $(z^x)^y = (z^y)^x = z^{x \cdot y}$.

```
fmod Diffie-Hellman is
  sorts Exp Elem NeElemSet Gen .
  subsort Elem < NeElemSet .
  subsort Gen < Exp .
  ops a b c : -> Elem .
  op exp : Exp NeElemSet -> Exp .
  op _*_ : NeElemSet NeElemSet -> NeElemSet [assoc comm] .
  var X : Exp . vars Y Z : NeElemSet .
  eq exp(exp(X,Y),Z) = exp(X,Y * Z) [variant] .
endfm
```

The attribute `variant` specifies that these equations will be used for variant-based unification. Since this theory has the finite variant property (see [38,60]), given the term `exp(x,y)` it is easy to verify that there are two most general variants.

The exponentiation operator is used in this thesis as follows. In Chapter 3, where the formal specification of key agreement group protocols such as Diffie-Hellman, Skinny Tree (STR), Joux, and Tripartite Authenticated Key agreement (TAK) is explained, the exponentiation property is used in increasingly complex cryptographic theories.

1.1.4 Bilinear Pairing

In 1984, Adi Shamir proposed ID-based encryption, or identity-based encryption (IBE). It is an important primitive of ID-based cryptography where public-key encryption uses a public key of an honest participant based on some unique information about the identity of the honest participant. It is particularly useful because there is no need to know an identity's public key prior to encryption. Shamir was unable to come up with a concrete solution, and identity-based encryption remained an open problem for many years. Bilinear pairings are currently a solution.

One of the goals to use pairing-based cryptography is to obtain a smaller finite field from an elliptic curve, and with that, we can analyze specific problems or attacks. It makes use of a pairing function $\hat{e} : G_1 \times G_2 \to G_T$ of two cryptographic groups G_1 and G_2 into a third group G_T. Typically, $G_1 = G_2$ and it will be a subgroup of the group of points on an elliptic curve over a finite field, and G_T will be a subgroup of the multiplicative group of a related finite field, and the map $\hat{e}$ will be derived from either the Weil or Tate pairing on the elliptic curve. When $G = G_1 = G_2$, the pairing is called *symmetric* and the pairing function $\hat{e}$ is commutative, i.e., if the participants agree on a generator $g \in G$, for any P,Q in G there exist integers i, j s.t. $P = g^i$, $Q = g^j$, $\hat{e}(P,Q) = \hat{e}(g^i, g^j) = \hat{e}(g,g)^{i*j} = \hat{e}(g^j, g^i) = \hat{e}(Q,P)$.

We follow the syntax of [70] and use the letter P as the agreed generator. We write aP instead of P^a for P added to itself a times, also called scalar multiplication of P by a. Note that we write `[a]P` in the equational theory below for clarification.

Some protocols that use the bilinear pairing cryptographic properties also require a hash function h and the following additive property (and its symmetric version, since $\hat{e}$ is commutative)

$$\hat{e}(Q, W + Z) = \hat{e}(Q,W) \cdot \hat{e}(Q,Z) \tag{1.1}$$

where $+$ is the additive symbol for the group G and $\cdot$ is the additive symbol for the group G_T given $\hat{e} : G \times G \to G_T$.

The bilinear pairing theory is represented in Maude as follows. We define three auxiliary constants `a`, `b`, and `c` of sort `Elem`. These properties are specified as follows. [1]

```
fmod Bilinear-Pairing is
  sorts Elem NeElemSet Gen GenP Exp ExpP ExpT .
  subsort Elem < NeElemSet .
  subsort Exp < ExpT .
  ops a b c : -> Elem .
  op exp : Gen NeElemSet -> Exp [ctor] .
  op exp : Exp NeElemSet -> Exp .
  op _*_ : NeElemSet NeElemSet -> NeElemSet [ctor assoc comm] .
  op p : -> GenP [ctor] .
  op em : GenP GenP -> Gen [ctor comm] .
  op em : ExpP ExpP -> Exp [comm] .
```

[1] This theory labels some symbols as constructors, which will be useful for Section 1.3.1

```
  op [_]_ : NeElemSet GenP -> ExpP [ctor] .
  op [_]_ : NeElemSet ExpP -> ExpP .
  op _+_ : NeElemSet NeElemSet -> NeElemSet [ctor assoc comm] .
  op _+_ : ExpP ExpP -> ExpP .
  op _·_ : ExpT ExpT -> ExpT [ctor assoc comm] .
  var X : Gen .
  vars Y Z : NeElemSet .
  vars P Q : GenP .
  eq exp(exp(X,Y),Z) = exp(X,Y * Z) [variant] .
  eq [Z]([Y]P) = [Z * Y]P [variant] .
  eq em([Y]P, [Z]Q) = exp(em(P,Q),Y * Z) [variant] .
  eq ([Y]P) + ([Z]P) = [Y + Z]P [variant] .
endfm
```

Note that the additive property 1.1 does not appear explicitly in the equational theory above and it is transformed as follows. The addition symbol + is split into two versions, one of them being an associative-commutative symbol. A new equation relating to these two versions of + is added. This equation denotes a homomorphic addition and it is easily handled by variant-based unification because it is defined on disconnected sorts ExpP and NeNonceSet. Also, symbol · is simply represented as an associative-commutative symbol.

Bilinear-pairing is used in this thesis as follows. In Chapter 3, where the formal specification of key agreement group protocols such as Diffie-Hellman, Skinny Tree (STR), Joux, and Tripartite Authenticated Key agreement (TAK) are explained, the bilinear-pairing property is used in TAK protocols.

1.1.5 Abelian-Group

In mathematics, an abelian group is a set G and an operation + that combines any two elements A and B of G to form another element of G, written as $A+B$. They are also called commutative groups because the result of applying the operation + to two elements does not depend on the order in which they are combined. An Abelian Group must satisfy four properties:

$$A+(B+C)=(A+B)+C \qquad \text{(associativity)}$$
$$A+B=B+A \qquad \text{(commutativity)}$$
$$A+\emptyset=A \qquad \text{(identity element is } \emptyset\text{)}$$
$$A+-A=\emptyset \qquad (-A \text{ is the inverse of } A)$$

1. Associativity. Where reordering or shifting the parentheses does not modify the meaning

2. Commutativity. Where reordering the element in G does not change the meaning

3. Identity. There exists an element z in G, applied to the element x in G that does not disrupt the meaning of x.

4. Inverse. For each x in G there exists an element y in G, where they cancel each other.

For the addition operation $+$, the Integers, and the Real numbers are abelian groups, with 0 as the identity element and $-x$ as the inverse of x where $x+(-y)$ is denoted as $x-y$. For the multiplication operation $*$, the integers and the Real numbers are abelian groups, with 1 as the identity element and $1/x$ as the inverse of x where $x*(1/y)$ is denoted as x/y [2]. Every cyclic group G is abelian, because if x,y are in G, then $x*y = g^m * g^n = g^{m+n} = g^n * g^m = y*x$.

Let us consider a protocol requiring an abelian group for its cryptographic properties, a generic three-pass key agreement protocol is proposed that is based on a generic trapdoor one-way function. When the generic protocol is specialized to the RSA setting, it is named KAS2 protocol, which was standardized by the National Institute of Standards and Technology (NIST). When the KAS2 protocol is specialized to the discrete logarithm, it is named DH2. This protocol is similar to the KEA+ protocol. It allows that parties can use different groups of elliptic curves assuming each party can operate on the other party's group. Party A selects a cyclic group G_1 with generator g_1. Party A selects a private key a and a public key is $(g_1)^a$. Party B selects a cyclic group G_2 with generator g_2. Party B selects a private key b and a public key is $(g_2)^b$. Party A selects an ephemeral private key $(g_2)^x$ and an ephemeral public key $((g_2)^b)^x$ for a random x and sends the public key to party B. Upon reception, party B raises the ephemeral public key to his inverse $1/b$ of private key b to obtain the ephemeral private key of A, i.e., $(((g_2)^b)^x)^{1/b} = (g_2)^x$.

The Abelian group theory is represented in Maude as follows. We define three auxiliary constants `a`, `b`, and `c` of sort `Elem`. These properties are specified as follows.

[2]Note that an expression such as 0/0 is problematic in different ways, since it is represented as 0*(1/0).

```
fmod Abelian-Group is
   sorts Elem AG .
   subsort Elem < AG .
   ops a b c : -> Elem .
   op _+_ : AG AG -> AG [assoc comm] .
   op -_ : AG -> AG .
   op 0 : -> AG .
   vars X Y Z : AG .
   eq X + 0 = X [variant] .
   eq X + - X = 0 [variant] .
   eq X + - X + Y = Y [variant] .
   eq - - X = X [variant] .
   eq - 0 = 0 [variant] .
   eq - X + - Y = -(X + Y) [variant] .
   eq -(X + Y) + Y = - X [variant] .
   eq -(- X + Y) = X + - Y [variant] .
   eq - X + - Y + Z = -(X + Y) + Z [variant] .
   eq -(X + Y) + Y + Z = - X + Z [variant] .
endfm
```

The Abelian Group is used in this thesis as follows. In Chapter 6, where we have defined a new unification algorithm that could be used to improve several cryptographic protocol analysis tools, such as Maude-NPA, we have to use the abelian group theory for the experiments. Unfortunately, the abelian group theory has a high computational cost and we have not specified and analyzed any protocol with an abelian group in this thesis.

1.1.6 Constraints

In computer science, the satisfaction of a set of constraints with variables of a given domain is the process of finding a solution, i.e. a set of values for the variables that satisfies all constraints. A user may be interested in finding just one solution, finding all solutions, or proving the unsatisfiability of the set of constraints. The techniques are different depending on the domain of the constraint. For constraints on a finite domain, satisfaction is usually solved via search, in particular a form of backtracking or local search. For constraints on the Real or Rational numbers, satisfaction is usually done via variable elimination or the Simplex Algorithm for Linear Programming (a special case of mathematical optimization).

In the 1970s, constraint satisfaction started to become used in the field of artificial intelligence. During the 1980s and 1990s, constraint satisfaction started to be embedded into a programming language. One of the first lan-

guages with specific support for constraint programming was Prolog. Since then, libraries for constraint programming have become available in other programming languages, such as Java or C++.

In the area of constraint programming, we are interested in some specific domains. Some popular domains are: booleans (SAT problem), integers, rationals, intervals, linear domains (linear real arithmetic, where only linear functions are used), non-linear domains, finite domains, and hybrid domains (involving more than one).

Satisfiability modulo theories (SMT) generalize the Boolean satisfiability problem (SAT) to more complex formulas involving Real numbers, integers, and/or various data structures such as lists, arrays, bit vectors, and strings. There are several SMT solvers such as Z3 from Microsoft, and the open-source CVC5. They have been used in a wide range of applications including automated theorem proving, program analysis, program verification, and software testing.

Constraints are used in this thesis as follows. In Chapter 2, where the formal specification of YubiKey and YubiHSM are explained, constraints associated with Lamport clocks are used. Lamport clocks are used to provide a partial ordering of events with minimal overhead. In Chapter 4, where several distance-bounding protocols are specified and analyzed, and linear arithmetic constraints on the Real numbers are used to represent traveled distances. In Chapter 5, where protocols with physical properties are specified and analyzed, and non-linear arithmetic constraints on the Real numbers are used to represent the participant's location.

1.2 Cryptographic protocol families

Cryptographic protocols are successfully analyzed using formal methods. However, formal approaches usually consider the encryption schemes as black boxes and insufficiently assume that an adversary cannot learn anything from an encrypted message unless he knows the key. The previous section on cryptographic properties tries to solve these problems. In the following, we present some advanced protocols analyzed in this thesis

1.2.1 APIs and Global Mutable Memory

Standards for cryptographic protocols have long been attractive candidates for formal verification. Cryptographic protocols are tricky to design and subject

to non-intuitive attacks even when the underlying cryptosystems are secure. Furthermore, when protocols that are known to be secure are implemented as standards, the modifications that are made during the standardization process may introduce new security flaws. Thus a considerable amount of work has been done in the application of formal methods to cryptographic protocol standards and the standards are treated symbolically, with the cryptosystems treated as black-box function symbols.

A cryptographic API is a set of instructions by which a developer of an application may allow it to take advantage of the cryptographic functionality of a secure module. These APIs allow an application to perform such functions as creating keys, using keys to encrypt and decrypt data, and exporting and importing keys to and from other devices. Cryptographic APIs should also enforce security policies. In particular, no application should be able to retrieve a key in the clear.

The IBM 4758 PCI Cryptographic Coprocessor is a secure cryptoprocessor implemented on a high-security, tamper-resistant, programmable PCI board. This highly secure subsystem includes specialized cryptographic electronics, a microprocessor, a memory, and a random number generator. As of June 2005, the 4758 was discontinued and was replaced by an improved, faster model called the IBM 4764. IBM supplies two cryptographic APIs for custom application development: the IBM Common Cryptographic Architecture (CCA) and the Public-Key Cryptography Standards version 11 (PKCS#11). These are standards that provide both a set of commands that could be used by a cryptographic API and mechanisms for setting and enforcing security policies. These security policies are specified in terms of attributes on keys and other data that declare which operations using these terms are legal or illegal.

In [64], the IBM 4758 CCA, an XOR-based API, was specified and analyzed for the first time using a general-purpose unbounded session cryptographic protocol verification tool, Maude-NPA, that provides direct support for AC theories. That paper analyzes not only the original protocol, but the different fixes provided by IBM, and the different XOR-linear versions that appeared in the literature. In particular, the paper reproduces Bond's attack on the different versions. In [65], the PKCS#11 was specified and analyzed using the Maude-NPA tool. Firstly, verifying the security of PKCS#11 was harder than verifying the security of an API such as IBM's CCA because the former was intended to be applied to a wide variety of platforms whereas the latter was only intended to be used for applications running on certain IBM systems. Secondly, the set of attributes forms a mutable global state which must be

accounted for. This issue was avoided in that paper by taking advantage of previous work in the literature to simplify the types of policies that need to be analyzed: a construct called an attribute policy is shown for a large class of "reasonable" attribute policies called complete policies, it is enough to prove security in the case of static policies, in which the attributes of a key are never changed after it is created. Thirdly, any formal verification system must be capable of verifying not just one or two policies specified by the developers of the API, as was the case of CCA, but any of a large class of policies that could be specified by a user.

In Chapter 2, the *YubiKey*, a small USB designed by Yubico to authenticate a user against network-based services, and the *YubiHSM*, Yubico's hardware security module (HSM), were specified and analyzed in Maude-NPA. The YubiKey allows for the secure authentication of a user against network-based services by considering different methods: one-time password (OTP), public-key encryption, public key authentication, and the Universal 2nd Factor (U2F) protocol [9]. YubiHSM is intended to operate in conjunction with a host application.

In the literature, two kinds of attacks on the first released version of YubiHSM API were shown. There has not been any completely automated analysis of these two attacks before this thesis. Furthermore, a mutable (non-monotonic) memory for storing previously seen keys or nonces is used in Yubikey and YubiHSM. The YubiHSM stores a very limited number of AES keys so that the server can use them to perform cryptographic operations without the key values ever appearing in the server's memory. The YubiHSM is designed to protect the YubiKey AES keys when an authentication server is compromised by encrypting the AES keys using a master key stored inside the YubiHSM. For the YubiKey and YubiHSM APIs, if each command is represented by an event, then a sequence of commands can be represented by the concatenation of the events associated with the sequence. However, the YubiKey and YubiHSM APIs also require different information to be stored from one API command to the next. Some command information is read-only, but other information is updated. The old data will appear in the precondition part of an API event, and the unmodified data, the new data, and the updated data will appear in the postcondition part of that event, which will then become the precondition part of the next API event.

1.2.2 Multi-party key agreement protocols

The Diffie-Hellman key exchange protocol of Section 1.1.3 is the earliest practical example of a public key agreement protocol implemented within the field of cryptography. The Diffie–Hellman key exchange algorithm allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure channel. The protocol is described in Alice & Bob notation as follows where we remark the shared key between participants:

$$\begin{aligned} A &\longrightarrow B : g^a \\ B &\longrightarrow A : g^b \\ Key_{AB} &: (g^b)^a = (g^a)^b = g^{a \cdot b} \end{aligned}$$

A multi-party key establishment can be seen as a generalization of a two-party key establishment. They are different versions of Diffie–Hellman key agreement that can negotiate a key shared by two or more participants. In the case of three honest participants in an insecure channel, a malicious participant can only see $g^a, g^b, g^c, g^{a \cdot b}, g^{a \cdot c}, g^{b \cdot c}$ and cannot create $g^{a \cdot b \cdot c}$.

$$\begin{aligned} A &\longrightarrow B,C : g^a \\ B &\longrightarrow A,C : (g^a)^b, g^b \\ C &\longrightarrow A,B : (g^a)^c, (g^b)^c, g^c \\ Key_{ABC} &: ((g^b)^c)^a) = ((g^a)^c)^b) = ((g^a)^b)^c) = g^{a \cdot b \cdot c} \end{aligned}$$

These honest participants have different options for choosing the order in which they contribute to the key. In the case of four honest participants, the protocol is as follows:

$$\begin{aligned} A &\longrightarrow B,C,D : g^a \\ B &\longrightarrow A,C,D : (g^a)^b, g^b \\ C &\longrightarrow A,B,D : (g^a)^b)^c, (g^a)^c, (g^b)^c, g^c \\ D &\longrightarrow A,B,C : ((g^a)^b)^d, ((g^a)^c)^d, ((g^b)^c)^d, g^d \\ Key_{ABCD} &: g^{a \cdot b \cdot c \cdot d} \end{aligned}$$

In Chapter 3, we specify several three-party key agreement protocols where each protocol creates a shared secret key different from the three-party Diffie-Hellman. The STR protocol uses nested exponentiations. The Joux protocol

uses exponentiation and bilinear pairing. The TAK protocols use signatures, hash, exponentiation, and bilinear pairing.

1.2.3 Distance bounding protocols

Distance bounding protocols are security protocols that rely on the traveled distance between two participants. Usually, there are two honest participants, but only one establishes an upper bound on the traveled distance between both. One of the participants is called the verifier V and the other is called the prover P because it has to prove it is within the allowed distance. Verifiers can be eavesdropped on, for example by manipulating the delay time, collaborating with dishonest participants, or by exploiting the presence of an honest prover. A common method of prevention is by using cryptography.

Since electromagnetic communications cannot travel faster than the speed of light, the verifier V approximates the traveled distance from the difference time between exchanged messages. The approximated distance is calculated as the delay time between a challenge sent by the verifier V and the response associated with this challenge. This round-trip delay time is divided by double the speed of light to calculate the traveled distance. The family of distance bounding protocols already has different applications. An application is credit card payment using EMV standard. EMV stands for "Europay, Mastercard, and Visa", the three companies that created the standard. These cards had to be inserted into a point-of-sale terminal. Nowadays, they have been replaced by contactless payment methods, and the card (prover) can be read by a point-of-sale terminal (verifier) within a short distance to guarantee correct communication. Another extensively used application is RFID tags. RFID tags are used in identification tasks for access control, payment, retail storage, and shipments. These tags must be held within a specific distance of the reader (verifier) to identify the tag (prover).

It is hard to model the physical properties of security protocols such as time and location. In general, there are two ways of handling those properties: an explicit model with physical information, or by using an abstract model (without physical information) and showing it is sound and complete with respect to a theoretical model. The former is more intuitive for the user, but the latter is often chosen because not many tools support reasoning about physical properties. By adding support for these physical properties, the former can handle many more protocols. Physical information is not represented by specific values but by unknown variables, in which the physical properties

are represented as constraints on those unknown physical variables.

This family of distance bounding protocols is vulnerable to different situations. Mafia fraud, where an honest prover is outside the neighborhood of the verifier but an intruder is inside pretending to be the honest prover, taking advantage of his position to relay the honest prover's messages. Terrorist fraud is an extension of Mafia fraud where an adversary tries to obtain the challenge from the verifier, either by extracting the key from a captured message or by tampering the verifier. Distance hijacking attacks, where an intruder located outside the neighborhood of the verifier succeeds in convincing the verifier that he is inside the neighborhood by exploiting the presence of an honest prover in the neighborhood to achieve his goal.

In Chapter 4, we specify several distance bounding protocols using only traveled distance, where the Mafia fraud and the Hijacking attack have been analyzed, and in Chapter 5 we extend this work to consider also the location of participants.

1.3 Modeling improvement

Several techniques for the optimization of cryptographic tools have been defined but little interest has been given to improving the modeling process. In this section, we consider two approaches studied in this thesis.

1.3.1 Constructors

Constructor symbols are extensively used in computer science: for representing data instead of functions, for manipulating programs as data, or for reasoning in complex semantic structures. Constructors can be characterized in the *no junk, no confusion* style of Goguen and Burstall, providing the mathematical semantics as the *initial term algebra* of a functional program, which corresponds to the least Herbrand model in logic programming.

Both the *logic* notion of a functor and the *functional* notion of a constructor refers to a symbol not appearing in the root position of the left-hand side of any predicate or equation. This notion of constructor allows to split a signature Σ as a disjoint union $\Sigma = \mathscr{D} \uplus \mathscr{C}$ where $\mathscr{D}$ are called *defined* symbols and $\mathscr{C}$ are called *constructor* symbols. In a functional program, the normalized (simplified) terms are typically made of constructor terms.

Since 2002, the Proverif crypto tool incorporated the distinction between what they called *destructor (defined) and constructor symbols.* This has never

been incorporated into other crypto tools such as AKiSS [17], Maude-NPA [55], OFMC [22], Scyther [40], Scyther-proof [3], and Tamarin [45] .

The notion of constructor sub-theory in Maude differs from the "logic" and "functional" notions. An equational theory (Σ, B, E) protects a constructor sub-theory $(\mathscr{C}, B_{\mathscr{C}}, E_{\mathscr{C}})$ if the set of constructor symbols $\mathscr{C}$ is a subset of Σ, the set of constructor axioms $B_{\mathscr{C}}$ is a subset of B, the set of constructor equations $E_{\mathscr{C}}$ is a subset of E, and two constructor terms are equal w.r.t. axioms $B_{\mathscr{C}}$ and equations $E_{\mathscr{C}}$ iff they are equal w.r.t. axioms B and equations E.

In the following equational theory, the modular exponentiation property typical of Diffie-Hellman protocols is defined using two versions of the exponentiation operator where, in the lefthand side of the equation, the outermost exponentiation operator is the defined symbol whereas the innermost exponentiation operator is the constructor symbol. Note that an auxiliary associative-commutative constructor symbol $*$ is used for exponents.

```
fmod DH-CFVP is
  sorts Exp Elem ElemSet Gen .
  subsort Elem < ElemSet .
  ops a b c : -> Elem [ctor] .
  op exp : Gen ElemSet -> Exp [ctor] .
  op exp : Exp ElemSet -> Exp .
  op _*_ : ElemSet ElemSet -> ElemSet [assoc comm ctor] .
  var X : Gen .
  vars Y Z : ElemSet .
  eq exp(exp(X,Y),Z) = exp(X,Y * Z) [variant] .
endfm
```

The constructor subtheory is as follows:

```
fmod DH-SubCFVP is
  sorts Exp Elem ElemSet Gen .
  subsort Elem < ElemSet .
  ops a b c : -> Elem [ctor] .
  op exp : Gen ElemSet -> Exp [ctor] .
  op _*_ : ElemSet ElemSet -> ElemSet [assoc comm ctor] .
endfm
```

Note that it may not always be possible to provide a constructor sub-theory, for example, the cancellation property of the exclusive-or theory forbids a constructor symbol.

Note also that we have used constructor symbols in the bilinear pairing theory of Section 1.1.4.

In Chapter 3, the cryptography theories of all the analyzed protocols include constructor symbols. In Chapter 6, we integrate the notion of constructor symbol into the variant-based unification algorithm with an impressive speed-up.

1.3.2 Protocol transformation

Program transformations are defined as any action that takes a computer program and generates another computer program. Usually, the transformed program is semantically equivalent to the original program. In other cases, the transformed program is not semantically equivalent to the original program but it works better in practice. Some transformations can be performed manually but it is more common to use an automated program transformation. Program transformations should be able to effectively process programs written in programming syntax. The problem of building and integrating adequate program transformations for conventional languages such as Java or C++ is as difficult as building the program transformation itself because of the complexity of such languages.

Küsters and Truderung created a protocol transformation that, given a Proverif protocol specification for some crypto properties and simple authentication properties, produces a Proverif protocol specification without any crypto properties. They have applied this transformation to the exclusive-or theory and the Diffie-Hellman exponentiation theory. These works do not really provide a protocol transformation, since they provide just classes of protocols where the analysis using Proverif is sound. Guttman also studied protocol transformations but his goal was not to optimize the verification but to ensure that a transformed protocol satisfies some security goals, when the source protocol did, focusing on incremental protocol construction.

Some cryptographic properties and protocols either cannot be expressed using equational unification or their state space is unmanageable. We have studied a protocol transformation that relies on rewriting theories. It relies on constructor term variants, which is an extension of term variants. Several crypto analysis tools rely on the variant-based equational unification capabilities of Maude such as Maude-NPA, Tamarin, and AKISS in such a way that these tools may benefit from the protocol transformation.

In the following, the Diffie-Hellman protocol using modular exponentiation is recalled.

$$A \longrightarrow B : g^a = Y$$
$$B \longrightarrow A : g^b = X$$
$$Key_{AB} : (X)^a = (Y)^b = g^{a \cdot b}$$

This protocol assumes the equational theory `DH-CFVP` above and its constructor sub-theory `DH-SubCFVP`. The expression X^a computed by Alice is transformed into $g^{a \cdot N}$ using $X \mapsto g^N$. The transformed version of the protocol is as follows:

$$A \longrightarrow B : g^a = g^{Ny}$$
$$B \longrightarrow A : g^b = g^{Nx}$$
$$Key_{AB} : (g^{Nx})^a = (g^{Ny})^b = g^{a \cdot b}$$

where variables X and Y before were of sort `Exp` and here variables Ny and Nx are of sort `ElemSet`. In Chapter 3, the cryptography theories of all the analyzed protocols include constructor symbols, all the protocols have been transformed and the variant equations are no longer necessary.

1.4 Objectives and Contributions

The main contribution of this thesis is to explore new techniques to adapt and specify complex cryptographic properties that are used in advanced security protocols and these properties must be verified to ensure safety and security. The objective of this thesis is to use formal methods as a basis for the formal specification and formal analysis of the security properties of the security protocols. The whole contribution of this thesis is focused on these categories:

- Complex cryptographic properties. For example, the complex cryptographic operations of bilinear pairing are used in identity-based protocols.
- Cryptographic protocol families. For example, the specification of Distance Bounding protocols that require time and distance information to function correctly.
- Protocol modeling improvements. For example, the use of constructor symbols to reduce the space and time analysis process.

1.5 Structure of this Thesis

This thesis is written as a collection of articles. The Compendium of publication thesis is more suited for Ph.D. theses where contributions have already been published in international conferences, which we believe may facilitate the reading and understanding of this thesis. Therefore, this thesis is organized as follows. Part II comprises a suite of conference papers that support this thesis. As shown each chapter is an adjusted version of an already-published paper at a scientific conference. All modifications have been edited from the author's version and followed as established by the regulations from the Doctoral School. Next, Part III shows up general conclusions of results obtained in this thesis. These conclusions are separated into article categories.

1.6 Publications

During the research of his Ph.D. studies, the student has published several scientific articles which are listed in this section. We list papers published in high-impact international conferences. A bolded title of the paper has been included in Part II. These publications follow a rating impact on their community CORE[3] and GGS[4]
The publications derived from this thesis are:

- Antonio González-Burgueño, Damián Aparicio-Sánchez, Santiago Escobar, Catherine A. Meadows and José Meseguer. **Formal verification of the YubiKey and YubiHSM APIs in Maude-NPA**. *(LPAR-22) 22nd International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, Awassa, Ethiopia, 16-21 November 2018.*

  GGS Rating: B | **CORE A**
  DOI: https://doi.org/10.48550/arXiv.1806.07209

- Damián Aparicio-Sánchez, Santiago Escobar, Raúl Gutiérrez and Julia Sapiña. **An Optimizing Protocol Transformation for Constructor Finite Variant Theories in Maude-NPA**. *Computer Security (ESORICS) 25th*

[3]The Computing Research and Education Association of Australasia
[4]GII (Group of Italian Professors of Computer Engineering), GRIN (Group of Italian Professors of Computer Science), and SCIE (Spanish Computer-Science Society)

European Symposium on Research in Computer Security, Guildford, UK, 14-18 September 2020.

GGS Rating: A+ | CORE A
DOI: https://doi.org/10.1007/978-3-030-59013-0_12

- Damián Aparicio-Sánchez, Santiago Escobar and Julia Sapiña. **Variant-based Equational Unification under Constructor Symbols**. *Technical Communications (ICLP) 36th International Conference on Logic Programming UNICAL, Rende (CS), Italy, 18-24 September 2020.*

GGS Rating: Not Rated | **CORE A**
DOI: https://arxiv.org/abs/2009.11070

- Damián Aparicio-Sánchez, Santiago Escobar, Catherine A. Meadows, José Meseguer and Julia Sapiña. **Protocol Analysis with Time**. *Progress in Cryptology (INDOCRYPT) 21st International Conference on Cryptology in India, Bangalore, India, 13-16 December 2020.*

GGS Rating B | CORE: Not Rated
DOI: https://doi.org/10.48550/arXiv.2010.13707

- Damián Aparicio-Sánchez, Santiago Escobar, Catherine A. Meadows, José Meseguer and Julia Sapiña. **Protocol Analysis with Time and Space**. *Protocols, Strands, and Logic - Essays Dedicated to Joshua Guttman on the Occasion of his 66th Birthday, pages 22-49, LNCS volume 13066, 19th November 2021*

Chapter in Festschrift in honor of Joshua Guttman
DOI: https://doi.org/10.1007/978-3-030-91631-2_2

1.7 Research Projects

This thesis would not have been possible without the funding of a set of research projects. The main contributions and derivative works of this thesis have been made in the context of the following projects:

- **Ministry of Economy and Business | Spain :** Project LoBaSS *Effective Solutions Based on Logic*, Scientific Research under award number TIN2015-69175-C4-1-R, this project was focused on using powerful logic-based technologies to analyze safety-critical systems. Period of 1st January 2016 to 31st December 2017. PI of the project was Santiago Escobar.

- **Air Force Office of Scientific Research | United States of America :** Project *Advanced symbolic methods for the cryptographic protocol analyzer Maude-NPA* Scientific Research under award number FA9550-17-1-0286, period of 1st January 2018 to 30th June 2020. This project was focused on incorporating in the Maude-NPA tool a hybrid equational unification framework. PI of the project was Santiago Escobar.

- **State Investigation Agency | Spain :** Project FREETech: *Formal Reasoning for Enabling and Emerging Technologies* Scientific I+D-i Research under award number RTI2018-094403-B-C32, this project was focused on the use of logic-based tools to analyze and verify vulnerabilities existing in emerging applications as well as communication protocols. Period of 1st January 2019 to 31st August 2019. PI of the project was Santiago Escobar.

1.8 Research Stays

The objectives of the research stay in international institutions are essential for the development of skills and improving competencies, observing others types of research methodologies in foreign work teams. I had the opportunity during my period as a Ph.D. student to collaborate and stay with:

- **Università degli Studi di Udine | Udine, Italy :** The research stay was carried out from 1st September 2021 to 1st December 2021 (3 months) and supervised by Marco Comini, an international expert in the field of abstract interpretation. During this period I was focused on analyzing the possible use of abstract interpretation for the analysis of communication protocols.

Part II

Selected Papers

CHAPTER 2

Formal verification of the YubiKey and YubiHSM APIs in Maude-NPA

Antonio González-Burgueño[1], Damián Aparicio[2], Santiago Escobar[2], Catherine Meadows[3], José Meseguer[2]

[1] University of Oslo, Norway
antonigo@ifi.uio.no

[2] VRAIN, Universitat Politècnica de València, Spain
{daapsnc, sescobar}@dsic.upv.es

[3] Naval Research Laboratory, Washington DC, USA
meadows@itd.nrl.navy.mil

[4] University of Illinois at Urbana-Champaign, USA
meseguer@illinois.edu

Abstract We perform an automated analysis of two devices developed by Yubico: *YubiKey*, designed to authenticate a user to network-based services, and *YubiHSM*, Yubico's hardware security module. Both are analyzed using the Maude-NPA cryptographic protocol analyzer. Although previous work has been done applying formal tools to these devices devices, there has not been any completely automated analysis. This is not surprising, because both YubiKey and YubiHSM, which make use of cryptographic APIs, involve a number of complex features: (i) discrete time in the form of *Lamport clocks*, (ii) a mutable memory for storing previously seen keys or nonces, (iii) event-based properties that require an analysis of sequences of actions, and (iv) reasoning modulo exclusive-or.

Partially supported by the EU (FEDER) and the Spanish MINECO under grant TIN 2015-69175-C4-1-R, by the Generalitat Valenciana under grant PROMETEOII/2015/013, by the US Air Force Oce of Scientific Research under award number FA9550-17-1-0286, and by NRL under contract number N00173-17-1-G002

Maude-NPA has provided support for exclusive-or for years but has not provided support for the other three features, which we show can also be supported by using constraints on natural numbers, protocol composition and reasoning modulo associativity. In this work, we have been able to automatically prove security properties of YubiKey and find the known attacks on the YubiHSM, in both cases beyond the capabilities of previous work using the Tamarin Prover due to the need of auxiliary user-defined lemmas and limited support for exclusive-or. Tamarin has recently been endowed with exclusive-orand we have rewritten the original specification of YubiHSM in Tamarin to use exclusive-or confirming that both attacks on YubiHSM can be carried out by this recent version of Tamarin.

2.1 Introduction

Nowadays there exist several security tokens having the form of a smartcard or an USB device, which are designed for protecting cryptographic values from an intruder, e.g, hosting service, email, e-commerce, online banks, etc. They are also used to ease authentication for the authorized users of a service, e.g., if you are using a service that verifies your Personal Identification Number (PIN), the same service should not be used for checking your flights, reading your emails, etc. By using an Application Programming Interface (API) to separate the service from the authentication system, such problems can be prevented.

Yubico is a leading company on open authentication standards and has developed two core inventions: the *YubiKey*, a small USB designed to authenticate a user against network-based services, and the *YubiHSM*, Yubico's hardware security module (HSM). The YubiKey allows for the secure authentication of a user against network-based services by considering different methods: one-time password (OTP), public key encryption, public key authentication, and the Universal 2nd Factor (U2F) protocol [9]. YubiKey works by using a secret value (i.e., a running counter) and some random values, all encrypted using a 128 bit Advanced Encryption Standard (AES). An important feature of YubiKey is that it is independent of the operating system and does not require any installation, because it works with the USB system drivers. YubiHSM is intended to operate in conjunction with a host application. It supports several modes of operation, but the key concept is a symmetric scheme where one device at one location can generate a secure data element in a secure environment. Although the main application area is for securing YubiKey's

OTP authentication/validation operations, the use of several generic cryptographic primitives allows a wider range of applications. The increasing success of YubiKey and YubiHSM has led to its use by governments, universities and companies like Google, Facebook, Dropbox, CERN, Bank of America etc., including more than 30,000 customers [7].

Cryptographic Application Programmer Interfaces (Crypto APIs) are commonly used to secure interaction between applications and hardware security module (HSMs), and are both used in YubiKey and YubiHSM. However, many crypto APIs have been subjected to intruder manipulation to disclose relevant information, as is the case for YubiHSM. In [74, 75], Künnemann and Steel show two kinds of attacks on the first released version YubiHSM API: (i) if the intruder had access to the server running YubiKey, where AES keys are generated, then it was able to obtain plaintext in the clear; (ii) even if the intruder had no access to the server running YubiKey, it could use previous nonces to obtain AES keys. However, there has not been any completely automated analysis of these two attacks to date. This is not surprising, because both YubiKey and YubiHSM, which make use of cryptographic APIs, involve a number of complex features: (1) discrete time in the form of *Lamport clocks*, (2) a mutable memory for storing previously seen keys or nonces, (3) event-based properties that require an analysis of sequences of actions, and (4) reasoning modulo exclusive-or. Maude-NPA has provided support for exclusive-or for years but has not provided support for the other three features, which we show can also be supported by using protocol composition and reasoning modulo associativity.

This paper is the third in a series using Maude-NPA to analyze cryptographic APIs; earlier work appeared in [64,65]. We find this problem area one of particular interest for two reasons. First, these APIs often use exclusive-or and this gives us the opportunity to explore how well Maude-NPA can be applied to protocols that use exclusive-or. Secondly, cryptographic APIs offer a number of other challenging features and this allows us to explore how Maude-NPA can handle them.

In this work we use Maude-NPA [2] for analyzing both YubiKey and YubiHSM. Our analysis was carried out on generation 2 of YubiKey and version 0.9.8 beta of the YubiHSM, as was the analysis of [74]. In order to facilitate comparison with earlier work, our formal specifications of YubiKey and YubiHSM follow those of [74] as closely as possible.

Contributions

1. We automatically prove the secrecy and authentication properties of YubiKey and find both attacks on YubiHSM, beyond the capabilities of Tamarin [21] in the earlier analysis in [74, 75], which was only able to find one attack due to limited support for exclusive-or. Tamarin has recently been endowed with exclusive-or in [45]. In Section 2.6.1 we have rewritten the original specification of YubiHSM in Tamarin to use exclusive-or and checked that both attacks on YubiHSM can now be carried out by Tamarin.

2. Our analysis was completely automatic and either found an attack or terminated with a finite search graph, showing that no attack of that kind exists. That is, Maude-NPA did not need any human guiding or auxiliary lemmas. However, both the earlier analysis in [74, 75] and our own analysis using the latest version of Tamarin in Section 2.6.1 involved some additional user-defined lemmas in order to prove properties of YubiKey and YubiHSM.

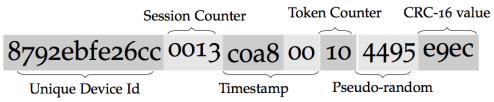
3. We show in Section 2.4 how we implemented Lamport clocks, mutable memory, and event-based properties in Maude-NPA, even though the tool does not support these natively, by using constraints on natural numbers, protocol composition and reasoning modulo associativity. These techniques should be applicable to protocols with similar properties.

Plan of the paper. In Sections 2.2 and 2.3 we give an overview of the YubiKey and YubiHSM, respectively. In Section 2.4 we give a high-level summary of Maude-NPA. In Section 2.5 we describe how we specified YubiKey and YubiHSM in Maude-NPA. In Section 2.6 we describe our experiments. Finally, in Section 2.7 we discuss related work, and we conclude in Section 2.8.

2.2 The YubiKey Device

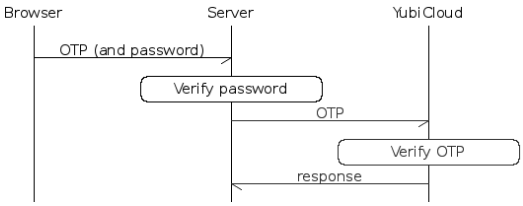
The YubiKey USB device [116] is an authentication device capable of generating One Time Passwords (OTPs). The YubiKey connects to a USB port and identifies itself as a standard USB keyboard, which allows it to be used in most computing environments using the system's native drivers.

We will focus on the YubiKey OTP mode, a mode that uses a button physically located on the YubiKey. When this button is pressed, it emits a string that can be verified only once against a server in order to receive the permission to access a service. Furthermore, a request for a new authentication token is triggered also by touching the YubiKey button. As a result of this request, some counters that are stored on the device are incremented and some random values are generated in order to create a fresh 16-byte plaintext. An OTP has the following concatenated fields [112]:



Each OTP is sent encrypted using an AES key and, thus, the YubiKey authentication server accepts an OTP only if it decrypts under the apropriate AES key and the token counter stored in the OTP is larger than the token counter stored in the last OTP received by the server. The token counter is used as a *Lamport clock* [78], i.e., it is used to determine the order of events in a distributed concurrent system by using a counter that both has a minimum value (e.g. 0) and has a minimum *tick* (increment of the counter) between every two possible events.

The authentication protocol of YubiKey involves three roles: (i) the user, (ii) the service, and (iii) the verification server. The user can have access to the service if it provides its own valid OTP generated by the YubiKey; its validity is verified by the verification server as explained before. The following example shows the user (Browser), the service (YubiCloud), and the verification server running the YubiKey API.



Since both the YubiKey and the server need to store information, e.g. the last received token counter, different predicates are defined in [74]: (i) `SharedKey(pid,k)` to represent the key `k` that is shared with the Yubikey

public ID `pid`, (ii) `Y(pid,sid)` that stores the corresponding secret ID `sid` associated to the Yubikey public ID `pid`, (iii) `Server(pid,sid,token counter)` that links the Yubikey public ID `pid` with the secret ID `sid` and the value of the last received counter `token counter`, and (iv) `YubiCounter(pid,token counter)` that represents that the current counter value `token counter` is stored on the Yubikey.

The YubiKey OTP generation scheme can be described by the following interaction.

1. The initialization of the YubiKey device takes place. A fresh public ID (*pid*), secret ID (*sid*) and YubiKey key (k) are generated. Any interaction between the YubiKey and the server will involve all three elements *pid*, *sid* and k. There are also two token counters, one stored on the Server and another stored on the YubiKey.

2. The YubiKey is plugged in. Every time the YubiKey is plugged in, the YubiKey token counter must be increased. However, we consider the compromised scenario of [74] in which the attacker has temporary access to the authentication server and it can produce all counter values, thus adding a new token counter as an input to the command and checking that it must be bigger than the old stored token counter. Figure 2.1 shows a graphical representation of the plugin event, including the input, output, and saved information.

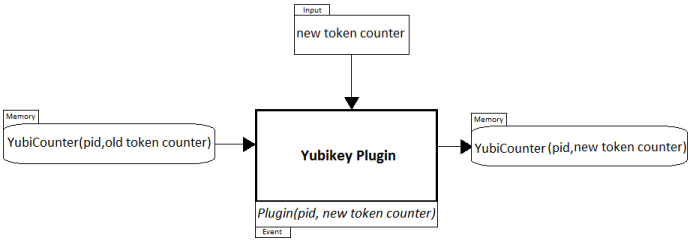


Figure 2.1: YubiKey Plugin API Command

3. The user pushes the YubiKey OTP generation button and generates a byte string formed by the *sid*, the YubiKey token counter, and a random number. The byte string is encrypted using a symmetric encryption operator and the saved key k. The YubiKey token counter is also increased. According to the compromised scenario, the YubiKey token

counter must be provided as input. Figure 2.2 shows a graphical representation of the button-pressing event, including the input, output, and saved information.

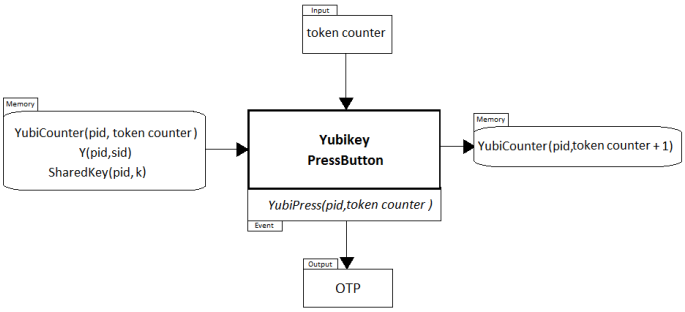


Figure 2.2: YubiKey Press Button Command

4. Upon reception of the generated OTP string, the basic verification steps are:

   4.1 The byte string is decrypted, and if it is not valid the OTP is rejected.

   4.2 The token counter stored in the OTP is compared with the server token counter. If smaller than or equal to the server token counter, the received OTP is rejected as a replay. According to the compromised scenario, the server token counter must be provided as input.

   4.3 A successful login must have been preceded by a button press for the same counter value, and there is not a second distinct login for this counter value. In this paper we omit this check and show that this property is always guaranteed, assuming that the checks on the byte string and token counter succeed.

   4.4 If all the checks succeed, the token counter stored in the OTP is stored as the server token counter and the OTP is accepted as valid.

   Figure 2.3 shows a graphical representation of the login event, including the input, output, and saved information.

In [74, 75], Künnemann and Steel were able to prove several properties:

(a) Absence of replay attacks, i.e., there are no two distinct logins that accept the same counter.

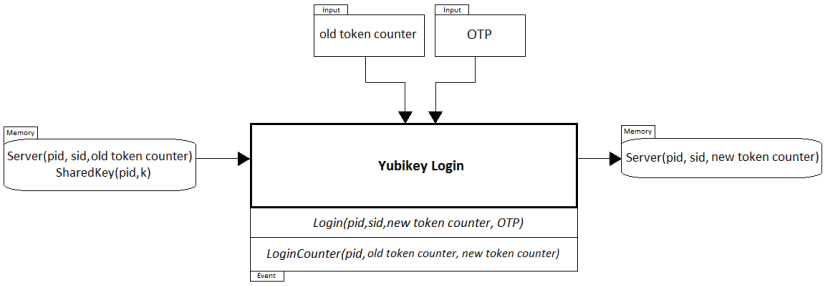


Figure 2.3: YubiKey Login Command

(b) Correspondence between pressing the button on a YubiKey and a successful login. In other words, a successful login must have been preceded by a button pressed for the same counter value. Furthermore, there is no second distinct login for this counter value.

(c) Counter values are different over time, i.e., the counter values associated to logins are monotonically increasing in time. Therefore, if one login has a smaller counter than the other, then it must have occurred earlier.

Note that the verification of properties (b) and (c) in [75] using Tamarin involved additional user-defined lemmas (see Section 2.6.1).

2.3 The YubiHSM Device

Yubico also distributes a USB device that works as an application-specific Hardware Security Module (HSM) to protect the YubiKey AES keys. The YubiHSM [117] stores a very limited number of AES keys so that the server can use them to perform cryptographic operations without the key values ever appearing in the server's memory. The YubiHSM is designed to protect the YubiKey AES keys when an authentication server is compromised by encrypting the AES keys using a master key stored inside the YubiHSM.

In addition, the YubiHSM can decrypt an indefinite number of YubiKey's OTP's with secure storage of the AES keys on the host computer. The AES keys are only readable to the YubiHSM through the use of *Authenticated Encryption with Associated Data (AEAD)*. The AEAD uses a cryptographic method that provides both confidentiality and authenticity. An AEAD consists of two parts: (i) the encryption of a message using the counter mode cryptographic mode of operation, and (ii) a *message authentication code (MAC)*

taken over the encrypted message. In order to construct, decrypt or verify an AEAD, a symmetrical cryptographic key and a piece of associated data are required. This associated data, called a *nonce* in the rest of the paper, can either be a uniquely generated handle or something that is uniquely related to the AEAD.

To encrypt a message using counter mode, one first divides it into blocks of equal length, each suitable for input to the block cipher AES, e.g. $data_1, \ldots, data_n$. The sequence $counter_1, \ldots, counter_n$ is then computed, where $counter_i = nonce \oplus i$ modulo 2^η and η is the length of a block in bits. The encrypted message is then $senc(counter_1, k) \oplus data_1; \ldots; senc(counter_n, k) \oplus data_n$, where $senc$ is the encryption function and k the symmetrical cryptographic key, and $senc(counter_1, k); \ldots; senc(counter_n, k)$ is called the *keystream*. Finally, the MAC is computed over the encrypted message and appended to obtain $(senc(counter_1, k) \oplus data_1; \ldots; senc(counter_n, k) \oplus data_n); MAC$. The MAC is of fixed length, so it is possible to predict where it starts in an AEAD. However, since the two attacks considered below do not involve most of the details about block cipher AES, we follow the generalization of [74] and consider just messages of the form $senc(cmode(nonce), k) \oplus data; mac(data, k)$.

In [74,75], Künnemann and Steel reported two kinds of attacks on version 0.9.8 beta of YubiHSM API: (a) if the intruder has access to the server running YubiKey, where AES keys are generated, then it is able to obtain plaintext in the clear; (b) even if the intruder has no access to the server running YubiKey, it can use previous nonces to obtain AES keys. However, they were only able to find the first attack in Tamarin due to the limited support for exclusive-or in Tamarin at that time (see Section 2.6.1).

The first attack involves the YubiHSM API command depicted in Figure 2.4, which takes a handle to an AES key and the nonce and applies the raw block cipher.

In order to perform this attack the intruder compromises the server to learn a *AEAD* and the key-handle used to produce it. Then, using the Block Encrypt command shown in Figure 2.4, an intruder is able to decrypt an *AEAD* by recreating the blocks of the key-stream: inputting $counter_i$ (the nonce) to the YubiHSM Block Encrypt API command. The intruder exclusive-ors the result with the *AEAD* truncated by the length of the *MAC* and obtains the plaintext. Note that the verification of this attack in [75] using Tamarin involved additional user-defined lemmas (see Section 2.6.1).

The second attack involves the YubiHSM command depicted in Figure 2.5 that takes a nonce, a handle to an AES key and some data and outputs an

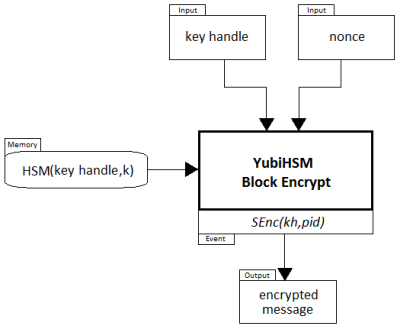


Figure 2.4: YubiHSM Block Encrypt API Command

AEAD. An intruder can produce an *AEAD* for the same handle *kh* and a value *nonce* that was previously used to generated another *AEAD*. An intruder can recover the keystream directly by using the AEAD-Generate command to encrypt a string of zeros and discarding the *MAC*. The result will be the exclusive-or of the keystream with a string of zeros, which is equal to the keystream itself. This attack is worse than the first one, because this command cannot be avoided or restricted (see [74]).

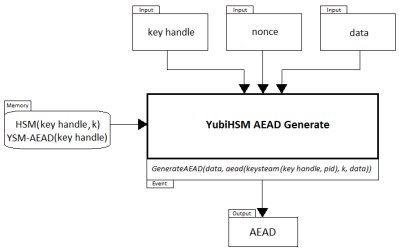


Figure 2.5: YubiHSM AEAD Generate API Command

2.4 Maude-NPA

In Maude-NPA, as in most formal analysis tools for cryptographic protocols, a protocol is modeled as a set of rules that describe the actions of honest principals communication across a network controlled by an intruder. Given a protocol $\mathscr{P}$, states in Maude-NPA are modeled as elements of an initial algebra $\mathscr{T}_{\Sigma_{\mathscr{P}}/E_{\mathscr{P}}}$, where $\Sigma_{\mathscr{P}} = \Sigma_{SS} \cup \Sigma_{\mathscr{C}}$ is the signature defining the sorts and

function symbols ($\Sigma_{\mathscr{C}}$ for the cryptographic functions and Σ_{SS} for all the state constructor symbols), $E_{\mathscr{P}} = E_{\mathscr{C}} \cup E_{SS}$ is a set of equations where $E_{\mathscr{C}}$ specifies the *algebraic properties* of the cryptographic functions and E_{SS} denotes properties of state constructors. The set of equations $E_{\mathscr{C}}$ may vary depending on different protocols, but the set of equations E_{SS} is always the same for all protocols. Therefore, a state is an $E_{\mathscr{P}}$-equivalence class $[t]_{E_{\mathscr{P}}} \in T_{\Sigma_{\mathscr{P}}/E_{\mathscr{P}}}$ with t a ground $\Sigma_{\mathscr{P}}$-term, i.e. a term without variables.

In Maude-NPA a *state pattern* for a protocol P is a term t of sort State which has the form $\{S_1 \,\&\, \cdots \,\&\, S_n \,\&\, \{IK\}\}$, where & is an infix associative-commutative union operator with identity symbol $\emptyset$. Each element in the set is either a *strand* S_i or the *intruder knowledge* $\{IK\}$ at that state.

The *intruder knowledge* $\{IK\}$ belongs to the state and is represented as a set of facts using comma as an infix associative-commutative union operator with identity element *empty*. There are two kinds of intruder facts: *positive* knowledge facts (the intruder knows m, i.e., $m \in \mathscr{I}$), and *negative* knowledge facts (the intruder *does not yet know* m but *will know it in a future state*, i.e., $m \notin \mathscr{I}$), where m is a message expression.

A *strand* [61] specifies the sequence of messages sent and received by a principal executing the protocol and is represented as a sequence

$[msg_1^{\pm}, msg_2^{\pm}, msg_3^{\pm}, \ldots, msg_{k-1}^{\pm}, msg_k^{\pm}]$ with $msg_i^{\pm}$ either msg_i^{-} (also written $-msg_i$) representing an input message, or msg_i^{+} (also written $+msg_i$) representing an output message. Note that each msg_i is a term of a special sort Msg. Variables of a special sort Fresh are used to represent pseudo-random values (nonces). Maude-NPA ensures that two distinct fresh variables will never be merged. Strands are extended with all the fresh variables $f_1, \ldots, f_k$ created by that strand, i.e., $:: f_1, \ldots, f_k :: [msg_1^{\pm}, msg_2^{\pm}, \ldots, msg_k^{\pm}]$.

Strands are used to represent both the actions of honest principals (with a strand specified for each protocol role) and the actions of an intruder (with a strand for each action an intruder is able to perform on messages). In Maude-NPA strands evolve over time; the symbol | is used to divide past and future. That is, given a strand [$msg_1^{\pm}$, …, $msg_i^{\pm}$ | $msg_{i+1}^{\pm}$, …, $msg_k^{\pm}$], messages $msg_1^{\pm}, \ldots, msg_i^{\pm}$ are the *past messages*, and messages $msg_{i+1}^{\pm}, \ldots, msg_k^{\pm}$ are the *future messages* ($msg_{i+1}^{\pm}$ is the immediate future message). A strand $[msg_1^{\pm}, \ldots, msg_k^{\pm}]$ is shorthand for $[nil \mid msg_1^{\pm}, \ldots, msg_k^{\pm}, nil]$. An *initial state* is a state where the bar is at the beginning for all strands in the state, and the intruder knowledge has no fact of the form $m \in \mathscr{I}$. A *final state* is a state where the bar is at the end for all strands in the state and there is no intruder fact of the form $m \notin \mathscr{I}$.

Since the number of states in $T_{\Sigma_{\mathscr{P}}/E_{\mathscr{P}}}$ is in general infinite, rather than exploring concrete protocol states $[t]_{E_{\mathscr{P}}} \in T_{\Sigma_{\mathscr{P}}/E_{\mathscr{P}}}$, Maude-NPA explores *symbolic state patterns* $[t(x_1,\ldots,x_n)]_{E_{\mathscr{P}}} \in T_{\Sigma_{\mathscr{P}}/E_{\mathscr{P}}}(\mathscr{X})$ on the free $(\Sigma_{\mathscr{P}}, E_{\mathscr{P}})$-algebra over a set of variables $\mathscr{X}$. In this way, a state pattern $[t(x_1,\ldots,x_n)]_{E_{\mathscr{P}}}$ represents not a single concrete state (i.e., an $E_{\mathscr{P}}$-equivalence class) but a possibly infinite set of states (i.e., an infinite set of $E_{\mathscr{P}}$-equivalence classes), namely all the *instances* of the pattern $[t(x_1,\ldots,x_n)]_{E_{\mathscr{P}}}$ where the variables $x_1,\ldots,x_n$ have been instantiated by concrete ground terms.

The semantics of Maude-NPA is expressed in terms of *rewrite rules* that describe how a protocol transitions from one state to another via the intruder's interaction with it. One uses Maude-NPA to find an attack by specifying an insecure state pattern called an *attack pattern*. Maude-NPA attempts to find a path from an initial state to the attack pattern via backwards narrowing (narrowing using the rewrite rules with the orientation reversed). That is, a narrowing sequence from an initial state to an attack state is searched *in reverse* as a *backwards path* from the attack state to the initial state. Maude-NPA attempts to find paths until it can no longer form any backwards narrowing step, at which point it terminates. If at that point it has not found an initial state, the attack pattern is shown to be *unreachable* modulo the equations $E_{\mathscr{P}}$. Note that Maude-NPA places *no bounds on the number of sessions*, so reachability is undecidable in general. Note also that Maude-NPA does not perform any data abstraction such as a bounded number of nonces. However, the tool makes use of various sound and complete state space reduction techniques that help to identify unreachable and redundant states, and thus make termination more likely.

2.4.1 Modeling Mutable Memory by means of Maude-NPA Strand Composition

Strands can be extended with *synchronization messages* [105] of the form $\{Role_1 \rightarrow Role_2 \,;;\, mode \,;;\, w\}$ where $Role_1, Role_2$ are constants of sort Role provided by the user, *mode* can be either `1-1` or `1-*` representing a one-to-one or one-to-many synchronization (whether an output message can synchronize with one or many input messages), and w is a term representing the information passed along in the synchronization messages. Synchronization messages are limited to the beginning and/or end of a strand. Although originally intended for a different use, they are very useful for representing a strand of unspecified length as a concatenation of different fixed-length strands. For ex-

ample, consider a module that receives i pieces of data, and then exclusive-ors them, i.e., $[-(M_1),\ldots,-(M_i),+(M_1 \oplus \cdots \oplus M_i)]$ for $i \geq 1$. This can be specified using three strands using synchronization messages:

1. $[\quad -(M_1), \{role_{\oplus} \rightarrow role_{\oplus}$;; `1-1` ;; $M_1\} \quad]$
2. $[\{role_{\oplus} \rightarrow role_{\oplus}$;; `1-1` ;; $M\}, -(M_2), \{role_{\oplus} \rightarrow role_{\oplus}$;; `1-1` ;; $(M \oplus M_2)\} \quad]$
3. $[\{role_{\oplus} \rightarrow role_{\oplus}$;; `1-1` ;; $M\}, +(M) \quad]$

Composition is then performed by unifying output synchronization messages with input synchronization messages of instances of strands.

For the YubiKey and YubiHSM APIs, if each event is represented by a strand, then an execution (e.g., Plugin followed by Press followed by Login) can be represented by the concatenation of the strands associated to the execution. However, the YubiKey and YubiHSM APIs also require different information to be stored from one API command to the next. Some information is read-only, but other information is updated, such as the `YubiCounter(pid,counter)`. Maude-NPA, unlike Tamarin, does not natively support mutable memory; but it can be modeled using synchronization messages. That is, the old data will appear in the input synchronization message of an API strand, and the new information will appear in the output synchronization message of that strand, which will then become the input synchronization message of the next API strand.

We model the mutable memory used by YubiKey as a multiset of predicates, where we define a new multiset union symbol `@`, which is an infix associative-commutative symbol with an identity symbol `empty`. Thus, for the strand describing the YubiKey button press, the input synchronization message is as follows:

```
{yubikey -> yubikey ;; 1-1 ;; Y(pid,sid) @ YubiCounter(pid,c1) @
   Server(pid,sid,c2) @ SharedKey(pid,k)}
```

Updating the counter of the YubiKey after a button press is represented by updating the second argument of the `YubiCounter(pid,c1)` predicate in the multiset. This updated multiset becomes the output synchronization of the strand.

2.4.2 Modeling Event Lists by means of Mutable Memory

The YubiKey and YubiHSM APIs also keep a rigid control of the ordering of events, where an event is a state transition in the system, and a proper analysis of actions is mandatory. Maude-NPA, unlike Tamarin, does not natively support the representation and analysis of event sequences; but we have implemented it by storing event sequences in the synchronization messages. This is helped by the fact that Maude-NPA, via the Maude language, has recently been endowed with built-in lists (using any associative symbol). We have defined a new infix associative symbol `++` with an identity symbol `nil` to represent an event list and also a new auxiliary infix symbol `|>` where the left-hand side contains the mutable memory and the right-hand side contains the event list. The input synchronization message for the button press strand has now the form:

```
{yubikey -> yubikey ;; 1-1 ;; Y(pid,sid) @ YubiCounter(pid,c1) @
Server(pid,sid,c2) @ SharedKey(pid,k) |> Plugin(pid,c3) ++ Press(pid,c4)}
```

Every time a new event occurs, it is inserted as a new element at the end of the event list. The leftmost elements are the oldest ones, whereas the rightmost elements are the newest. Thus, if we want to say that event e_1 must occur before event e_2, we can express this with the event list L_1 `++` e_1 `++` L_2 `++` e_2 `++` L_3, where any of the L_i could be empty.

2.4.3 Modeling Lamport Clocks in Maude-NPA Using Constraints

Lamport clocks require the testing of constraints: that is, whether one counter is smaller than another. This is simple to do when the counters have concrete values. However, since Maude-NPA does not consider concrete protocol states but symbolic state patterns (terms with logical variables), the equality and disequality constraints handled by Maude-NPA are predicates defined over variables, whose domain, in the case of Lamport clocks, is the natural numbers.

In Maude-NPA strands can be extended with equality and disequality constraints [57] of the form *Term1* `eq` *Term2* and *Term1* `neq` *Term2*. Whenever an equality constraint is found during the execution of a strand, the two terms in the equality constraint are unified modulo the set $E_{\mathscr{P}}$ of equations of the protocol and a new state is created for each possible unifier. Whenever a disequality constraint is found during the execution of a strand, it is simply stored in an internal repository of disequality constraints associated to each protocol

state; but every time a new state is going to be generated during the state space exploration, all the disequality constraints in the internal repository are tested for satisfiability [57]. That is, for each state, if there is a disequality constraint of the form *Term1* `neq` *Term2* such that *Term1* and *Term2* are equal modulo $E_{\mathscr{P}}$ then the state is discarded.

We deal with Lamport clocks symbolically by representing the relations between clocks as constraints in *Presburger Arithmetic*. Although various *Satisfiability modulo theories (SMT)* [96] solvers such as CVC4[1], Yices[2], and Microsoft Z3[3] could be used for this purpose, we decided to avoid the complexities of invoking an external tool while executing Maude-NPA. Instead, we have used the variant-based decision procedure for Presburger Arithmetic already available in Maude [91]; but considered only positive numbers without zero.

Adding two natural numbers i and j is written as i + j. Checking whether a natural number i is smaller than another natural number j is represented in Maude-NPA by a constraint of the form j `eq` i + k, where k is a new variable.

2.5 Formal Specifications in Maude-NPA

2.5.1 Formal Specifications of YubiKey in Maude-NPA

In our specification, each command of the YubiKey API (Figures 2.1, 2.2, and 2.3) plus the initialization is specified in Maude-NPA as a strand.

The initialization strand is defined as follows. Three new Fresh values are defined: a YubiKey public ID (`rpid`), a secret ID (`rsid`), and a key '`rk`' shared with the server. Variables of sort Fresh are wrapped by symbol `Fr` as in [74].

```
:: rk,rpid,rsid ::
[ +(init),
  {yubikey -> yubikey ;; 1-1 ;;
   YubiCounter(Fr(rpid), 1) @ Server(Fr(rpid),Fr(rsid),1) @
   Y(Fr(rpid),Fr(rsid)) @ SharedKey(Fr(rpid),Fr(rk))
   |> Init(Fr(rpid),Fr(rk)) ++ ExtendedInit(Fr(rpid),Fr(rsid),Fr(rk))}]
```

The API command represented in Figure 2.1 shows what happens when a YubiKey is being plugged in. This command checks that the new re-

[1] Available at `http://cvc4.cs.stanford.edu/web/`.

[2] Available at `http://yices.csl.sri.com`.

[3] Available at `https://github.com/Z3Prover/z3`.

ceived counter is smaller than the previous one and updates the predicate YubiCounter.

```
 :: nil ::
[{yubikey -> yubikey ;; 1-1 ;; YubiCounter(pid,otc) @ mem |> EL },
   -(tc), (tc eq (otc + extra)),
 {yubikey -> yubikey ;; 1-1 ;; YubiCounter(pid,tc) @ mem |> EL ++ Plugin(pid,tc)}]
```

Note that the parameter mem denotes the rest of the mutable memory and the parameter EL denotes the previous event list. The variable extra is an auxiliary variable used just for testing the numerical constraint.

The command shown in Figure 2.2 represents what happens when the YubiKey button is pressed and the OTP is sent. The OTP is represented by message senc(sid ; tc ; Fr(rnpr),k) where senc denotes symmetric encryption using key k and symbol ; denotes message concatenation[4].

```
:: rnpr,rnonce ::
[{yubikey -> yubikey ;; 1-1 ;;
   YubiCounter(pid,tc) @ Y(pid,sid) @ SharedKey(pid,k) @ mem |> EL },
   -(tc),
   +(pid ; Fr(rnonce) ; senc(sid ; tc ; Fr(rnpr),k)),
 {yubikey -> yubikey ;; 1-1 ;; YubiCounter(pid,tc + 1) @ Y(pid,sid)
    @ SharedKey(pid,k) @ mem |> EL ++ YubiPress(pid,tc)}]
```

Finally, the command shown in Figure 2.3 represents what happens when the server receives a login request. This request is accepted if the counter inside the encryption is larger than the last counter stored on the server.

```
:: nil ::
[{yubikey -> yubikey ;; 1-1 ;; Server(pid,sid,otc) @ SharedKey(pid,k) @ mem |> EL},
   -(pid ; nonce ; senc(sid ; tc ; pr, k)),
   -(otc),  (tc eq (otc + extra)),
  {yubikey -> yubikey ;; 1-1 ;; Server(pid,sid,tc) @ SharedKey(pid,k) @ mem
  |> EL ++ Login(pid,sid,tc,senc(sid ; tc ; pr, k)) ++ LoginCounter(pid,otc,tc)}]
```

2.5.2 Formal Specification of YubiHSM in Maude-NPA

We consider only the two commands shown in Figures 2.4 and 2.5. Each command is specified in Maude-NPA as a strand. YubiHSM makes extensive

[4]Note that ; is not an associative symbol and it is used as “message cons" symbol using Maude label “gather (e E)" that concatenates a single element to the left of a list.

use of exclusive-or, denoted by the symbol $*$, which satisfies the following equations:

$$\begin{aligned} x*(y*z) &= (x*y)*z && \text{(associativity)} \\ x*y &= y*x && \text{(commutativity)} \\ x*null &= x && \text{(identity element)} \\ x*x &= null && \text{(self-cancellation)} \end{aligned}$$

The YubiHSM command of Figure 2.4 is defined as follows.

```
:: nil ::
[ {YubiHSM -> YubiHSM ;; 1-1 ;; HSM(kh,k) @ mem |> EL },
   -(kh), -(nonce),
   +(senc(cmode(nonce),k)),
  {YubiHSM -> YubiHSM ;; 1-1 ;; HSM(kh,k) @ mem |> EL ++ SEnc(kh,nonce) } ]
```

We use two alternative definitions of the YubiHSM command of Figure 2.5, one to represent what happens when the command processes plaintext from the intruder, and another to represent what happens when the command processes plaintext from a legitimate principal. This is possible because, unlike in the traditional Dolev-Yao model, honest principals communicate with the YubiHSM devices directly, not through the intruder. This means that we can represent an honest principal's input data as internal to the system. Moreover, in this instance such a representation is necessary, since we are asking whether the intruder can learn the input data. We maximize the intruder's advantage, however, by giving it control over the other input data.

The following strand represents the intruder learning an honest principal's input plaintext data. We assume that the plaintext data is a Fresh value. In this way, we can later ask whether the intruder is able to learn that Fresh value. We use the following macro: `aead(n,k,d) = (senc(cmode(n),k) * d) ; mac(d,k)`.

```
:: data ::
[ {YubiHSM -> YubiHSM ;; 1-1 ;; HSM(kh,k) @ mem |> EL },
   -(kh), -(nonce),
   +(aead(nonce,k,Fr(data))),
   {YubiHSM -> YubiHSM ;; 1-1 ;; HSM(kh,k) @ mem
      |> EL ++ GenerateAEAD(Fr(data),aead(nonce,k,Fr(data)))}]
```

In the second strand we represent the Fresh value (`data`) associated to the plaintext data by an input from the intruder.

```
:: nil ::
[ {YubiHSM -> YubiHSM ;; 1-1 ;; HSM(kh,k) @ mem |> EL },
   -(data), -(kh), -(nonce),
   +(aead(nonce,k,data)),
   {YubiHSM -> YubiHSM ;; 1-1 ;; HSM(kh,k) @ mem
      |> EL ++ GenerateAEAD(data,aead(nonce,k,data)) }]
```

2.6 Experiments

We have been able to automatically prove secrecy and authentication properties (a,b,c) below of YubiKey and to find both attacks (d,e) below on YubiHSM:

(a) Absence of replay attacks in YubiKey, i.e., there are no two distinct logins that accept the same counter.

(b) Correspondence between pressing the button on a YubiKey and a successful login. In other words, a successful login must have been preceded by a button pressed for the same counter.

(c) Counter values of YubiKey are different over time, where a successful login invalidates previous OTPs.

(d) If the intruder has access to the server running YubiKey, where the YubiHSM AES keys are generated, then it is able to obtain plaintext in the clear.

(e) If the intruder has no access to the server running YubiKey, it can use previous YubiHSM nonces to obtain AES keys.

Table 2.1 summarizes the result of the analyses of the YubiKey and YubiHSM APIs specified in Maude-NPA showing the number of generated nodes in each step. The notation "(1)" represents that the tool found 1 solution to the question asked by the attack pattern. When the number of generated nodes is 0, the attack pattern is *unreachable*.

The complete paper with the appendix provides the specific attack patterns are available at `https://arxiv.org/abs/1806.07209`. All the details on how the attack patterns are specified and which was the output returned by Maude-NPA are available at `http://personales.upv.es/sanesro/Maude-NPA-YubiKey-YubiHSM/`. The analyses were completely automatic and we obtained finite search graphs for all the attack patterns. This was achieved

Attack Pattern	Depth								
	1	2	3	4	5	6	7	8	9
YubiKey (a)	4	4	9	21	88	160	0		
YubiKey (b)	4	7	16	14	2	2	5	0	
YubiKey (c)	4	4	6	18	55	80	0		
YubiKey Login	1	1	2	1	1	1	1	1	1(1)
YubiHSM (d)	1	2	3	4	7	13	24	40	76(1)
YubiHSM (e)	4	6	11	26(1)					

Table 2.1: Output YubiKey and YubiHSM Experiments

thanks to Maude's associative unification (i.e., event list expressions are included within the attack patterns) and the variant-based SMT solving for Lamport clocks (i.e., specific counter constraints are included). Note that Maude-NPA uses a full specification of exclusive-or, an unbounded session model, and an active Dolev-Yao intruder model. Moreover, it does not perform any data abstraction such as a bounded number of nonces, so there are no false positives or negatives.

2.6.1 Experiments using Tamarin

In [74, 75], the authors needed some user-defined lemmas to prove properties (b) and (c) of YubiKey and property (d) of YubiHSM, and they could not find the attack of property (e) due to the limited support for exclusive-or in Tamarin at that time. However, Tamarin has recently been endowed with exclusive-or in [45]. In this section, we report on some experiments that we have performed with it. The Tamarin core team has kindly provided this last version to us. In summary, nothing has changed for properties (b) and (c), and property (e) can *now* be carried out by Tamarin using a lemma. However, our automated analysis [63] was done before [45] appeared.

The latest version of Tamarin with exclusive-or (version 1.4.0) is *now* available at `https://github.com/tamarin-prover/tamarin-prover`.
Both YubiKey and YubiHSM specifications are also available at path "`examples/related_work/YubiSecure_KS_STM12`".
Property (b) of YubiKey is specified as follows:

```
lemma one_count_foreach_login[reuse,use_induction]:
       "∀ pid sid x otp  #t2 . Login(pid,sid,x,otp)@#t2  →
         ( ∃  #t1 . YubiPress(pid,x)@#t1 ∧  #t1<#t2 )"
```

whereas property (c) of YubiKey is specified as follows:

```
lemma Login_invalidates_smaller_counters:
       "∀ pid otc1 tc1 otc2 tc2 #t1 #t2 #t3 .
            LoginCounter(pid,otc1,tc1)@#t1 ∧ LoginCounter(pid,otc2,tc2)@#t2
     ∧  Smaller(tc1,tc2)@#t3 →  #t1<#t2 "
```

Both properties (b) and (c) use a constraint `Smaller(tc1,tc2)` where `tc1` and `tc2` are token counters; for property (b) the constraint is not written explicitly but it is also necessary. In order for Tamarin to prove these two properties the following user-defined lemmas are necessary (called axioms in Tamarin).

```
axiom smaller:
        "∀ #i a b. Smaller(a,b)@#i →  Ex z. a+z=b"
axiom transitivity:
        "∀ #t1 #t2 a b c. IsSmaller(a,b)@#t1 ∧  IsSmaller(b,c)@#t2
        →  ∃  #t3 . IsSmaller(a,c)@#t3 "
axiom smaller_implies_unequal:
        " ¬ (∃  a #t . IsSmaller(a,a)@#t)"
```

Since these properties do not require exclusive-or, nothing changes from the earlier version of Tamarin to the latest one and, when proving properties (b) and (c) without these axioms either Tamarin is not able to terminate or terminates but without finding a proof.

For properties (d) and (e) of YubiHSM, we have rewritten the original specification to use exclusive-or following the examples published in [45]. The following axioms were necessary to find the attack of property (d) in [75], and they are still necessary when using the new specification of YubiHSM and the latest version of Tamarin.

```
axiom theory_before_protocol:
    "∀ #i #j. Theory() @ i & Protocol() @ j ==> i < j"
axiom onetime:
    "∀ #t3 #t4 . OneTime()@#t3 & OneTime()@t4 ==> #t3=#t4"
```

We encoded property (e) as follows:

```
lemma auth_intruder_obtain_AES[use_induction]: exists-trace
   "∃ data ks k mac #t1 #t2 .
   GenerateAEAD(data,<senc(ks,k),mac>)@#t1 ∧  K(senc(ks,k))@#t2  ∧  #t1<#t2"
```

We checked that the latest version of Tamarin was able to find the corresponding attack of the new specification of propery (e), though our automated analysis in Maude-NPA was done [63] before [45] appeared.

2.7 Related Work

There is a vast amount of research on the formal analysis of APIs, so in this related work section we will concentrate on the work that is closest to ours, namely, the formal analysis of the YubiKey and YubiKey-like systems. Further related work on APIs and exclusive-or can be found in [64, 65].

Besides the work on formalizing and verifying YubiKey that we have already discussed, there has been further work focused on building tools for analyzing policies for YubiKey and YubiKey-like systems.

In [6], Yubico presents some security arguments on their website. An independent analysis was given by blogger Fredrik Björck in 2009 [23, 24], raising issues that Yubico responded to in a subsequent post. Oswald, Richter, et al. [99] analyze the YubiKey, generation 2, for side-channel attacks. They show that non-invasive measurements of the power consumption of the device allow retrieving the AES-key within approximately one hour of access. The authors mentioned a more recent version of the YubiKey, the YubiKey Neo which employs a certified smart-card controller that was designed with regard to implementation attacks and is supposed to be more resilient to power consumption analysis.

Künnemann et al. [74] performed a deep analysis of the different properties of YubiKey, but unlike our analysis using the Maude-NPA tool, they needed to use different lemmas to check some properties that cannot be done automatically by the Tamarin prover, whereas these properties can be checked out in an automatic way by the Maude-NPA tool. Some properties were not proved due to limited support for exclusive-or.

Mutable global state memory can be used in protocols that provide end-to-end encryption for instant messaging [37] as well as at the Trusted Platform Module (TPM) [43] that is a hardware chip designed to enable commodity computers to achieve greater levels of security than is possible by software alone.

2.8 Conclusions

The main contributions of this paper are to both prove properties of YubiKey generation 2 and find the known attacks on version 0.9.8 of YubiHSM in a completely automated way beyond the capabilities of previous work in the literature. This allowed us to perform the analysis of these APIs in a fully-unbounded session model making no abstraction or approximation of fresh

values, and with no extra assumptions. These APIs involve several challenges: (1) handling of Lamport clocks, (2) modeling of mutable memory, (3) handling of constraints on the ordering of events, and (4) support for symbolic reasoning modulo exclusive or.

The main goal of this work has been to investigate whether Maude-NPA could complement and extend the formal modeling and analysis results about YubiKey and YubiHSM obtained in [74]. This is a non-obvious question: on the one hand, Maude-NPA has provided support for exclusive-or for years, so it is well-suited for meeting Challenge (4). But, on the other hand, previous applications of Maude-NPA have not addressed Challenges (1)-(3). The main upshot of the results we present can be summarized as follows: Challenge (2) can by met by expressing mutable memory in terms of *synchronization messages*, a notion used in Maude-NPA to specify protocol compositions [105], Challenge (3) can by met by the recently added unification modulo associativity, allowing an easy treatment of lists, and Challenge (1) can be met by a slight extension of Maude-NPA's current support for equality and disequality constraints [57], namely, by adding also support for constraints in Presburger Arithmetic. In this way, we show how challenges (1)-(4) can all be met by Maude-NPA, and how these results in automated formal analyses of YubiKey and YubiHSM that substantially extend previous analyses. Very few tools are well equipped to simultaneously handle all of the challenges.

What remains to be seen is how generally applicable these tools are to YubiKey and similar APIs. We note that previous work on analyzing API protocols in Maude-NPA did not achieve termination of the search space: the IBM CCA API in [64] and the PKCS#11 in [65]. In this work we have been able to achieve termination of many properties thanks to the use of Lamport clocks, mutable memory, and event lists. But more secure API case studies are needed to further test and advance the techniques presented here.

CHAPTER 3

An Optimizing Protocol Transformation for Constructor Finite Variant Theories in Maude-NPA

Damián Aparicio-Sánchez[1], Santiago Escobar[1], Raúl Gutiérrez[2], Julia Sapiña[1]

[1] VRAIN, Universitat Politècnica de València, Spain
{daapsnc,sescobar,jsapina}@upv.es

[2] Universidad Politécnica de Madrid, Spain
r.gutierrez@upm.es

Abstract Maude-NPA is an analysis tool for cryptographic security protocols that takes into account the algebraic properties of the cryptosystem. Maude-NPA can reason about a wide range of cryptographic properties. However, some algebraic properties, and protocols using them, have been beyond Maude-NPA capabilities, either because the cryptographic properties cannot be expressed using its equational unification features or because the state space is unmanageable. In this paper, we provide a protocol transformation that can *safely* get rid of cryptographic properties under some conditions. The time and space difference between verifying the protocol with all the crypto properties and verifying the protocol with a minimal set of the crypto properties is remarkable. We

Partially supported by the EU (FEDER) and the Spanish MCIU under grant RTI2018-094403-B-C32, by the Spanish Generalitat Valenciana under grant PROMETEO/2019/098, and by the US Air Force Office of Scientific Research under award number FA9550-17-1-0286. Julia Sapiña has been supported by the Generalitat Valenciana APOSTD/2019/127 grant.

also provide, for the first time, an encoding of the theory of bilinear pairing into Maude-NPA that goes beyond the encoding of bilinear pairing available in the Tamarin tool.

3.1 Introduction

Maude-NPA [55] is an analysis tool for cryptographic security protocols that takes into account the algebraic properties of the cryptosystem. Sometimes algebraic properties can uncover weaknesses of cryptosystems and, in other cases, they are part of the protocol security assumptions. Maude-NPA uses an approach similar to its predecessor, the NRL Protocol Analyzer (NPA) [82], i.e., it is based on unification and performs backward search from an attack state pattern to determine whether or not it is reachable. However, unlike the original NPA, it has a theoretical basis on *rewriting logic* [53] and *narrowing* [35], and while NPA could only be used to reason about equational theories involving a fixed set of rewrite rules, Maude-NPA can be used to reason about a wide range of cryptographic properties [2, 55], including cancellation of encryption and decryption, Diffie-Hellman exponentiation [54], exclusive-or [106], and some approximations of homomorphic encryption [52, 113].

However, some algebraic properties and protocols using them have been beyond Maude-NPA capabilities, either because the cryptographic properties cannot be expressed using its equational unification features or because the state space is unmanageable. We provide a protocol transformation that can substantially reduce the search space, i.e., given some cryptographic properties, expressed using the equational unification features of Maude-NPA, and a protocol, we are able to transform the protocol in such a way that some cryptographic properties are no longer necessary, and thus can be safely removed. The time and space difference between verifying the protocol with all the crypto properties and verifying the protocol with a minimal set of the crypto properties is remarkable. We also provide, for the first time, an encoding of the theory of bilinear pairing into Maude-NPA that goes beyond the encoding of bilinear pairing available in Tamarin [108], the only crypto tool with such an equational theory.

Our protocol transformation relies on a program transformation from [92] for rewrite theories in Maude that we have improved by relaxing some of its applicability conditions. Such program transformation relies on *constructor term variants* [91], which is an extension of *term variants* [38,60]. Nowadays, several

crypto analysis tools rely on the variant-based equational unification capabilities of Maude, such as Maude-NPA but also Tamarin [44] and AKISS [17]. These tools may be benefited from our protocol transformation and, furthermore, from our encoding of the theory of bilinear pairing. Our contributions may even be useful for other tools with more limited crypto properties such as ProVerif [25], Scyther [39] or Scyther-proof [84].

The main contributions of this work are: (i) we provide a non-trivial protocol transformation based on [92]; (ii) since the protocols of Section 3.5 do not satisfy the conditions of [92], we provide a more powerful protocol transformation that we implemented, made available online, and pays off in practice; (iii) we provide an encoding of bilinear pairing that can handle all the protocols of Section 3.5 that Tamarin cannot handle; (iv) we implemented the algorithm of [110] for the computation of constructor variants [91] from scratch; and (v) there was no implementation of the program transformation of [92] and we implemented it.

After some preliminaries on Section 3.2, we present how Maude-NPA works in Section 3.3. We introduce our protocol transformation in Section 3.4. Section 3.5 presents several increasingly complex case studies: Diffie-Hellman protocol in Section 3.5.1, STR protocol in Section 3.5.2, Joux protocol in Section 3.5.3, and TAK protocols in Section 3.5.4. Our experiments are presented in Section 3.6 and we conclude in Section 3.7.

3.2 Preliminaries

We follow the classical notation and terminology for term rewriting [18], and for rewriting logic and order-sorted notions [86]. We assume an order-sorted signature Σ with a poset of sorts $(\mathsf{S}, \leq)$. We also assume an S-sorted family $\mathcal{X} = \{\mathcal{X}_\mathsf{s}\}_{\mathsf{s}\in\mathsf{S}}$ of disjoint variable sets with each $\mathcal{X}_\mathsf{s}$ countably infinite. $\mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ is the set of terms of sort s, and $\mathcal{T}_{\Sigma,\mathsf{s}}$ is the set of ground terms of sort s. We write $\mathcal{T}_\Sigma(\mathcal{X})$ and $\mathcal{T}_\Sigma$ for the corresponding order-sorted term algebras. For a term t, $\mathcal{V}ar(t)$ denotes the set of variables in t. Throughout this paper, Σ is assumed to be *preregular*, so each term t has a least sort, denoted $ls(t)$.

A *substitution* $\sigma \in \mathcal{S}ubst(\Sigma, \mathcal{X})$ is a sorted mapping from a finite subset of $\mathcal{X}$ to $\mathcal{T}_\Sigma(\mathcal{X})$. Substitutions are written as $\sigma = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$, where the domain of σ is $Dom(\sigma) = \{X_1, \ldots, X_n\}$ and the set of variables introduced by terms $t_1, \ldots, t_n$ is written $Ran(\sigma)$. The identity substitution is denoted id. Substitutions are homomorphically extended to $\mathcal{T}_\Sigma(\mathcal{X})$. The application of a

substitution σ to a term t is denoted by $t\sigma$ or $\sigma(t)$. The restriction of σ to a set of variables V is $\sigma|_V$. Composition of two substitutions σ and σ' is written $\sigma \circ \sigma'$.

A Σ-*equation* is an unoriented pair $t = t'$, where $t, t' \in \mathscr{T}_\Sigma(\mathscr{X})_\mathsf{s}$ for some sort $\mathsf{s} \in \mathsf{S}$. Given Σ and a set E of Σ-equations, order-sorted equational logic induces a congruence relation $=_E$ on terms $t, t' \in \mathscr{T}_\Sigma(\mathscr{X})$. The E-equivalence class of a term t is denoted by $[t]_E$ and $\mathscr{T}_{\Sigma/E}(\mathscr{X})$ and $\mathscr{T}_{\Sigma/E}$ denote the corresponding order-sorted term algebras modulo E. Throughout this paper we assume that $\mathscr{T}_{\Sigma,\mathsf{s}} \neq \emptyset$ for every sort s, because this affords a simpler deduction system. An *equational theory* (Σ, E) is a pair with Σ an order-sorted signature and E a set of Σ-equations.

An E-*unifier* for a Σ-equation $t = t'$ is a substitution σ such that $t\sigma =_E t'\sigma$. A set of substitutions $CSU_E(t = t')$ is said to be a *complete* set of unifiers for the equality $t = t'$ modulo E iff: (i) each $\sigma \in CSU_E(t = t')$ is an E-unifier of $t = t'$; (ii) for any E-unifier ρ of $t = t'$ there is $\sigma \in CSU_E(t = t')$ and τ s.t. $\sigma\tau =_E \rho$; (iii) for all $\sigma \in CSU_E(t = t')$, $Dom(\sigma) \subseteq (\mathscr{V}ar(t) \cup \mathscr{V}ar(t'))$. An E-unification algorithm is *complete* if for any equation $t = t'$ it generates a complete set of E-unifiers. A unification algorithm is said to be *finitary* and complete if it always terminates after generating a finite and complete set of solutions.

A *rewrite rule* is an oriented pair $l \to r$, where $l \notin \mathscr{X}$ and $l, r \in \mathscr{T}_\Sigma(\mathscr{X})_\mathsf{s}$ for some sort $\mathsf{s} \in \mathsf{S}$. An *(unconditional) order-sorted rewrite theory* is a triple (Σ, E, R) with Σ an order-sorted signature, E a set of Σ-equations, and R a set of rewrite rules. The relation $\to_{R,E}$ on $\mathscr{T}_\Sigma(\mathscr{X})$ is defined as: $t \to_{p,R,E} t'$ (or just $t \to_{R,E} t'$) iff there exist $p \in Pos_\Sigma(t)$, a rule $l \to r$ in R, and a substitution σ such that $t|_p =_E l\sigma$ and $t' = t[r\sigma]_p$. The transitive (resp. transitive and reflexive) closure of $\to_{R,E}$ is denoted by $\to^+_{R,E}$ (resp. $\to^*_{R,E}$). A term t is (R,E)-irreducible if there is no t' s.t. $t \to_{R,E} t'$. The R,E-*narrowing* relation on $\mathscr{T}_\Sigma(\mathscr{X})$ is defined as $t \leadsto_{p,\sigma,R,E} t'$ ($\leadsto_\sigma$ if R,E are understood, and $\leadsto$ if σ is also understood) if there is a non-variable position $p \in Pos_\Sigma(t)$, a rule $l \to r \in R$ standardized apart (i.e., contains no variable previously met during any previous computation) and a unifier $\sigma \in CSU_E(t|_p = l)$, such that $t' = (t[r]_p)\sigma$. The transitive (resp. transitive and reflexive) closure of $\leadsto$ is denoted by $\leadsto^+$ (resp. $\leadsto^*$).

3.3 The Maude-NPA

Given a protocol $\mathscr{P}$ to be specified, protocol states are modeled as elements of an initial algebra $\mathscr{T}_{\Sigma_\mathscr{P}/E_\mathscr{P}}$, i.e., each state is an equivalence class $[t]_{E_\mathscr{P}} \in T_{\Sigma_\mathscr{P}/E_\mathscr{P}}$

where $\Sigma_{\mathscr{P}}$ is the set of symbols defining the protocol $\mathscr{P}$, and $E_{\mathscr{P}}$ specifies the *algebraic properties* of the cryptographic functions $\Sigma_{\mathscr{P}}$. The cryptographic properties $E_{\mathscr{P}}$ may vary depending on different protocols.

The signature $\Sigma_{\mathscr{P}}$ incorporates some predefined symbols for protocol infrastructure. A state is a term of the form $\{S_1 \,\&\, \cdots \,\&\, S_n \,\&\, \{IK\}\}$ where & is an associative-commutative union operator with identity symbol $\emptyset$.

The *intruder knowledge IK* of a state $\{S_1 \,\&\, \cdots \,\&\, S_n \,\&\, \{IK\}\}$ is defined as a set of facts using the comma as an associative-commutative union operator with identity element $\emptyset$. There are two kinds of intruder facts: *positive* knowledge facts (the intruder knows m, i.e., $m \in \mathscr{I}$), and *negative* knowledge facts (the intruder *does not yet know* m but *will know it in a future state*, i.e., $m \notin \mathscr{I}$), where m is a message expression.

Each S_i of a state $\{S_1 \,\&\, \cdots \,\&\, S_n \,\&\, \{IK\}\}$ is called a strand and specifies the sequence of messages sent and received by a principal executing the protocol. *Strands* [61] are represented as a sequence of messages $[msg_1^{\pm}, msg_2^{\pm}, msg_3^{\pm}, \ldots, msg_{k-1}^{\pm}, msg_k^{\pm}]$ with $msg_i^{\pm}$ either msg_i^{-} (also written $-msg_i$) representing an input message, or msg_i^{+} (also written $+msg_i$) representing an output message. Note that each msg_i is a term of a special sort Msg; this sort is extended by the user to allow any user-definable protocol syntax. Variables of a special sort Fresh are used to represent pseudo-random values (nonces) and Maude-NPA ensures that two distinct fresh variables will never be merged. Strands are extended with all the fresh variables created by that strand, i.e., $:: f_1, \ldots, f_k :: [msg_1^{\pm}, msg_2^{\pm}, \ldots, msg_k^{\pm}]$. Section 3.5 includes several examples of honest and Dolev-Yao strands.

Strands are used to represent both the actions of honest principals (with a strand specified for each protocol role) and the actions of an intruder (with a strand for each action an intruder is able to perform on messages). In Maude-NPA strands evolve over time; the symbol | is used to divide past and future. That is, given a strand $[\ msg_1^{\pm},\ \ldots,\ msg_i^{\pm}\ |\ msg_{i+1}^{\pm},\ \ldots,\ msg_k^{\pm}\ ]$, messages $msg_1^{\pm}, \ldots, msg_i^{\pm}$ are the *past messages*, and messages $msg_{i+1}^{\pm}, \ldots, msg_k^{\pm}$ are the *future messages* ($msg_{i+1}^{\pm}$ is the immediate future message). A strand $[msg_1^{\pm}, \ldots, msg_k^{\pm}]$ is shorthand for $[nil \mid msg_1^{\pm}, \ldots, msg_k^{\pm}, nil]$. An *initial state* is a state where the bar is at the beginning for all strands in the state, and the intruder knowledge has no fact of the form $m \in \mathscr{I}$. A *final state* is a state where the bar is at the end for all strands in the state and there is no intruder fact of the form $m \notin \mathscr{I}$.

Since the number of states $T_{\Sigma_{\mathscr{P}}/E_{\mathscr{P}}}$ is in general infinite, rather than exploring concrete protocol states $[t]_{E_{\mathscr{P}}} \in T_{\Sigma_{\mathscr{P}}/E_{\mathscr{P}}}$ Maude-NPA explores *state patterns*

$[t(x_1,\ldots,x_n)]_{E_{\mathscr{P}}} \in T_{\Sigma_{\mathscr{P}}/E_{\mathscr{P}}}(\mathscr{X})$ on the free $(\Sigma_{\mathscr{P}},E_{\mathscr{P}})$-algebra over a set of variables $\mathscr{X}$. In this way, a state pattern $[t(x_1,\ldots,x_n)]_{E_{\mathscr{P}}}$ represents not a single concrete state but a possibly infinite set of such states, namely all the *instances* of the pattern $[t(x_1,\ldots,x_n)]_{E_{\mathscr{P}}}$ where the variables $x_1,\ldots,x_n$ have been instantiated by concrete ground terms.

The semantics of Maude-NPA is expressed in terms of a Maude rewrite theory, including *rewrite rules* that describe how a protocol moves from one state to another via the intruder's interaction with it [55]. One uses Maude-NPA to find an attack by specifying an insecure state pattern called an *attack pattern*. Maude-NPA attempts to find a path from an initial state to the attack pattern via *backwards narrowing* (using the narrowing capabilities of Maude [35] but with the reversed orientation of the rewrite rules). That is, a sequence from an initial state to an attack state is searched *in reverse* as a *backwards path* from an attack state pattern to an initial state. Maude-NPA attempts to find paths until it can no longer form any backwards narrowing steps, at which point it terminates. If at that point it has not found an initial state, the attack pattern is judged *unreachable*; providing a proof of security rather than finding attacks. However, note that Maude-NPA places *no bound on the number of sessions*, so reachability is undecidable in general. Maude-NPA does not achieve termination by any data abstraction, e.g. a bounded number of nonces. Instead, the tool makes use of a number of sound and complete state space reduction techniques that help to identify unreachable and redundant states [56], and thus make termination more likely.

3.4 Protocol Transformation

Maude-NPA relies on equational unification to perform each backwards narrowing step. Some cryptographic properties often involve the development of dedicated algorithms (see [15]). Maude-NPA provides built-in support for theories involving symbols with any combination of associativity (A), commutativity (C), and identity (U) axioms. Furthermore, by relying on the variant-based equational unification [35,60], Maude-NPA allows users to augment the basic set of equational axioms supported with rewrite rules such as cancellation of encryption and decryption, Diffie-Hellman exponentiation [54], exclusive-or [106], and some approximations of homomorphic encryption [52,113].

3.4.1 Finite Variant Theories

An equational theory $(\Sigma, \mathscr{E})$ is often decomposed into a disjoint union $\mathscr{E} = E \uplus B$, where B is a set of algebraic axioms (which are implicitly expressed in Maude as operator attributes `assoc`, `comm`, and `id:` keywords) and E consists of variant equations that are implicitly oriented from left to right as a set $\vec{E}$ of rewrite rules (and operationally used as simplification rules modulo B).

Definition 1 (Decomposition [60])**.** *Let $(\Sigma, \mathscr{E})$ be an order-sorted equational theory. We call $(\Sigma, B, \vec{E})$ a* decomposition *of $(\Sigma, \mathscr{E})$ if $\mathscr{E} = E \uplus B$ and $(\Sigma, B, \vec{E})$ is an order-sorted rewrite theory satisfying the following properties:*

1. *B is* regular*, i.e., for each $t = t'$ in B, we have $\mathscr{V}ar(t) = \mathscr{V}ar(t')$, and* linear*, i.e., for each $t = t'$ in B, each variable occurs only once in t and in t'.*
2. *B is* sort-preserving*, i.e., for each $t = t'$ in B and substitution σ, we have $t\sigma \in \mathscr{T}_\Sigma(\mathscr{X})_\mathsf{s}$ iff $t'\sigma \in \mathscr{T}_\Sigma(\mathscr{X})_\mathsf{s}$. Furthermore, for each equation $t = t'$ in B, all variables in $\mathscr{V}ar(t)$ and $\mathscr{V}ar(t')$ have a common top sort.*
3. *B has a finitary and complete unification algorithm.*
4. *The rewrite rules in $\vec{E}$ are* convergent*, i.e., confluent, terminating, and coherent modulo B, and* sort-decreasing*.*

In a decomposition, for each term $t \in \mathscr{T}_\Sigma(\mathscr{X})$, there is a unique (up to B-equivalence) $(\vec{E}, B)$-irreducible term that can be obtained by rewriting t to its *normal* form, which is denoted by $t\downarrow_{\vec{E},B}$. We often abuse notation and say that $(\Sigma, B, \vec{E})$ is a decomposition of an order-sorted equational theory $(\Sigma, \mathscr{E})$ even if $\mathscr{E} \neq E \uplus B$ but E is instead the explicitly extended B-coherent completion of a set E' such that $\mathscr{E} = E' \uplus B$ (see [60]).

Example 1. *The property associated to Diffie-Hellman exponentiation is described using the following equational theory in Maude, including an auxiliary associative-commutative symbol $*$ for exponents so that $(z^x)^y = (z^y)^x = z^{x*y}$.*

```
fmod DH-FVP is
  sorts Exp Nonce NeNonceSet Gen .
  subsort Nonce < NeNonceSet . subsort Gen < Exp .
  op exp : Exp NeNonceSet -> Exp .
  op _*_ : NeNonceSet NeNonceSet -> NeNonceSet [assoc comm] .
  var X : Exp . vars Y Z : NeNonceSet .
  eq exp(exp(X,Y),Z) = exp(X,Y * Z) [variant] .
endfm
```

Note that X admits any exponentiation and Y and Z are restricted to non-empty multisets of nonces. For an arbitrary term g of sort Gen *and three arbitrary terms* n_A, n_B, n_C *of sort* Nonce*,* $t = exp(exp(exp(g,n_A),n_B),n_C)$ *is simplified into* $t{\downarrow_{\vec{E},B}} = exp(g, n_A * n_B * n_C)$.

In order to provide a finitary and complete unification algorithm for a decomposition $(\Sigma, B, \vec{E})$, the *folding variant narrowing* strategy is defined in [60]. Intuitively, an $(\vec{E},B)$-*variant* of a term t is the $(\vec{E},B)$-irreducible form of an *instance* $t\sigma$ of t. That is, the variants of t are all of the possible $(\vec{E},B)$-irreducible terms to which instances of t evaluate.

Definition 2 (Term Variant [38, 60])**.** *Given a term t and a decomposition* $(\Sigma, B, \vec{E})$*, we say that* (t', θ) *is a* variant *of t if* $t' =_B (t\theta){\downarrow_{\vec{E},B}}$*, where* $Dom(\theta) \subseteq \mathscr{V}ar(t)$ *and* $Ran(\theta) \cap \mathscr{V}ar(t) = \emptyset$.

Example 2. *Following Example 1, the set of variants for the term* $exp(X,Y)$ *is infinite, since we have* $(exp(X', Y * Y'), \{X \mapsto exp(X', Y')\})$, $(exp(X'', Y * Y' * Y''), \{X \mapsto exp(exp(X'', Y''), Y')\})$, ….

It is possible to compute a complete and finite set of variants for some equational theories.

Definition 3 (Complete set of Variants [60])**.** *Given a decomposition* $(\Sigma, B, \vec{E})$ *and a term t, we write* $[\![t]\!]_{\vec{E},B}$ *for a* complete *set of variants of t, i.e., for any variant* (t_2, θ_2) *of t, there is a variant* $(t_1, \theta_1) \in [\![t]\!]_{\vec{E},B}$ *such that* $(t_1, \theta_1) \leq_{\vec{E},B} (t_2, \theta_2)$*, where* $(t_1, \theta_1) \leq_{\vec{E},B} (t_2, \theta_2)$ *iff there is a substitution* ρ *such that* $|_{(\theta_1 \rho)}\mathscr{V}ar(t) =_B |_{(\theta_2 {\downarrow_{\vec{E},B}})}\mathscr{V}ar(t)$ *and* $t_1\rho =_B t_2$*. An equational theory has the* finite variant property *(FVP) (also called* finite variant theory*) iff for all* $t \in \mathscr{T}_\Sigma(\mathscr{X})$*,* $[\![t]\!]_{\vec{E},B}$ *is a finite set.*

Example 3. *Following Example 2, there exists a complete and finite set of variants for the term* $exp(X,Y)$*: the variant* $(exp(X,Y), id)$ *and the variant* $(exp(X', Y * Y'), \{X \mapsto exp(X', Y')\})$*. Any other variant includes a substitution not in irreducible form.*

ariant-based unification is defined as follows (see [35, 60] for details).

Definition 4 (Variant-based Unification [60])**.** *Let* $(\Sigma, B, \vec{E})$ *be a decomposition of an equational theory. Let* t_1, t_2 *be two* Σ*-terms. Then,* ρ *is an unifier of* t_1 *and* t_2 *iff* $\exists (t', \rho) \in [\![t_1]\!]_{\vec{E},B} \cap [\![t_2]\!]_{\vec{E},B}$.

3.4.2 Constructor Finite Variant Theories

Quite often, the signature Σ of a decomposition $(\Sigma, B, \vec{E})$, on which $\mathscr{T}_{\Sigma/B}$ is defined, has a natural subsignature of *constructor* symbols Ω. The elements of the *canonical algebra* $C_{\Sigma/\vec{E},B} = \{[t\downarrow_{\vec{E},B}]_B \mid t \in \mathscr{T}_\Sigma\}$, i.e., the B-equivalence classes computed by $\vec{E}, B$-simplification, are Ω-terms, whereas the other symbols are viewed as functions which are simplified into constructor symbols.

Proverif [25] already incorporated this distinction between what they called *destructor and constructor symbols* time ago in contrast to other crypto tools such as AKISS [17], Maude-NPA [2], OFMC [93], Scyther [39], Scyther-proof [84], and Tamarin [21]. In the rest of the paper, we exploit this distinction in Maude-NPA without altering the tool.

A decomposition $(\Sigma, B, \vec{E})$ *protects* a *constructor decomposition* $(\Omega, B_\Omega, \vec{E}_\Omega)$ iff $\Omega \subseteq \Sigma$, $B_\Omega \subseteq B$, and $\vec{E}_\Omega \subseteq \vec{E}$, and for all $t, t' \in \mathscr{T}_\Omega(\mathscr{X})$ we have: (i) $t =_{B_\Omega} t' \iff t =_B t'$, (ii) $t = t\downarrow_{\vec{E}_\Omega,B_\Omega} \iff t = t\downarrow_{\vec{E},B}$, and (iii) $C_{\Omega/\vec{E}_\Omega,B_\Omega} = C_{\Sigma/\vec{E},B}|_\Omega$. A *constructor decomposition* $(\Omega, B_\Omega, \emptyset)$ is called *free*. A decomposition $(\Sigma, B, \vec{E})$ is called *sufficiently complete* with respect to a free constructor decomposition $(\Omega, B_\Omega, \emptyset)$ iff for each $t \in \mathscr{T}_\Sigma$ we have: (i) $t\downarrow_{\vec{E},B} \in \mathscr{T}_\Omega$, and (ii) if $u \in \mathscr{T}_\Omega$ and $u =_B v$, then $v \in \mathscr{T}_\Omega$. This ensures that if any element in an equivalent class is a constructor term, all the other elements are also constructor.

Example 4. *We can extend the equational theory of Example 1 to protect a constructor subsignature*[1] *by overloading symbol* `exp` *to use the former*[2] *sorts* Exp *and* Gen.

```
fmod DH-CFVP is
  sorts Exp Nonce NeNonceSet Gen .
  subsort Nonce < NeNonceSet .
  op exp : Gen NeNonceSet -> Exp [ctor] .
  op exp : Exp NeNonceSet -> Exp .
  op _*_ : NeNonceSet NeNonceSet -> NeNonceSet [assoc comm ctor] .
  var X : Gen . vars Y Z : NeNonceSet .
  eq exp(exp(X,Y),Z) = exp(X,Y * Z) [variant] .
endfm
```

For an arbitrary term g of sort Gen *and three arbitrary terms n_A, n_B, n_C of sort*

[1] Operator declarations labeled `ctor`, their associated sorts, and no equation.

[2] This equational theory, as well as all the ones in Section 3.5, should be parametric on sorts `Gen`, `GenP` and `Nonce` but we omit such more general-purpose definitions for simplicity (see [35] for details on parametric equational theories).

Nonce, $t = exp(exp(exp(g,n_A),n_B),n_C)$ *is simplified into the constructor term* $t{\downarrow_{\vec{E},B}} = exp(g,n_A * n_B * n_C)$.

The notion of a constructor variant, rather than a variant, is defined in [91].

Definition 5 (Constructor Variant [91])**.** *Given a decomposition* $(\Sigma,B,\vec{E})$ *protecting a constructor decomposition* $(\Omega,B_\Omega,\vec{E}_\Omega)$ *and a* Σ*-term* t*, we say that a variant* (t',θ) *of* t *is a* constructor variant *if* $t' \in \mathscr{T}_\Omega(\mathscr{X})$.

Example 5. *Following Example 4, the set of constructor variants for the term* $exp(X,Y)$ *is infinite, as in Example 2, since we have* $(exp(X',Y*Y'),\{X \mapsto exp(X',Y')\})$, $(exp(X'',Y*Y'*Y''),\{X \mapsto exp(exp(X'',Y''),Y')\})$, ….

Definition 6 (Complete set of Constructor Variants [91])**.** *Given a decomposition* $(\Sigma,B,\vec{E})$ *protecting a constructor decomposition* $(\Omega,B_\Omega,\vec{E}_\Omega)$ *and a* Σ*-term* t*, we write* $[\![t]\!]^\Omega_{\vec{E},B}$ *for a* complete *set of constructor variants of* t*, i.e., for any constructor variant* (t_2,θ_2) *of* t*, there is a constructor variant* $(t_1,\theta_1) \in [\![t]\!]^\Omega_{\vec{E},B}$ *such that* $(t_1,\theta_1) \leq_{\vec{E},B} (t_2,\theta_2)$. *A decomposition* $(\Sigma,B,\vec{E})$ *has the* constructor finite variant property *(CFVP) (or it is called a* constructor finite variant theory*) iff for all* $t \in \mathscr{T}_\Sigma(\mathscr{X})$, $[\![t]\!]^\Omega_{\vec{E},B}$ *is a finite set.*

Example 6. *Following Example 5, there exists a finite and complete set of constructor variants for the term* $exp(X,Y)$ *where* X *is of sort* Exp*, since we have* $(exp(XG,Y*Y'),\{X \mapsto exp(XG,Y')\})$ *where* XG *is a new variable of sort* Gen *instead of sort* Exp.

An algorithm for computing $[\![t]\!]^\Omega_{\vec{E},B}$ is provided in [110] for equational theories that are FVP. This algorithm assumes an extra condition called *preregular below*, i.e., a term cannot have a constructor typing above a non-constructor typing.

Definition 7 (Preregular below [91])**.** *Given a decomposition* $(\Sigma,B,\vec{E})$ *protecting a constructor decomposition* $(\Omega,B_\Omega,\vec{E}_\Omega)$*, the (preregular) order-sorted signature* $(\Sigma,<)$ *is called* preregular below *iff* $\forall t \in \mathscr{T}_\Sigma(\mathscr{X})$, $ls_\Omega(t) = ls_\Sigma(t)$.

Example 7. *Consider the following equational theory*

```
fmod DH-NoPreregularBelow is
  sorts Nonce NeNonceSet GenSub Gen ExpSub Exp .
  subsort GenSub < Gen . subsort ExpSub < Exp .
  subsort Nonce < NeNonceSet .
  op gSub : -> GenSub [ctor] .
```

```
  op g : -> Gen [ctor] .
  op exp : GenSub NeNonceSet -> ExpSub .
  op exp : Gen NeNonceSet -> Exp [ctor] .
  op exp : Exp NeNonceSet -> Exp .
endfm
```

The signature is not preregular below since, given an arbitrary term n_A of sort Nonce*, the least sort of the term $exp(gSub, n_A)$ is* ExpSub *in the original signature but* Exp *in the constructor subsignature.*

The set of constructor variants of the form $[\![\langle l, r\rangle]\!]^{\Omega}_{\vec{E},B}$, where l and r are, respectively, the lefthand and righthand sides of a rewrite rule in a rewrite theory, play a crucial role in the following theory transformation $\mathscr{R} \mapsto \mathscr{R}^{\Omega}_{l,r}$ from [92].

Definition 8 ($\mathscr{R} \mapsto \mathscr{R}^{\Omega}_{l,r}$ [92])**.** *Given a rewrite theory $(\Sigma, B \uplus E, R)$ such that $(\Sigma, B, \vec{E})$ is CFVP and preregular below, the rewrite theory $(\Sigma, B \uplus E, R^{\Omega}_{l,r})$ is defined as $R^{\Omega}_{l,r} = \{l' \to r' \mid l \to r \in R \wedge (\langle l', r'\rangle, \sigma) \in [\![\langle l, r\rangle]\!]^{\Omega}_{\vec{E},B}\}$.*

Section 3.5 shows how several protocols are transformed using a protocol transformation that relies on this program transformation.

Example 8. *Any expression of the form $exp(X, Y)$, where X is of sort* Exp *and Y is of sort* NeNonceSet*, occurring in any lefthand or righthand side of a rule in a rewrite theory will be replaced by the constructor variant shown in Example 6.*

Theorem 1 ([92, Theo. 7])**.** *Given a rewrite theory $(\Sigma, B \uplus E, R)$ such that $(\Sigma, B, \vec{E})$ is a decomposition protecting a free constructor decomposition $(\Omega, B_{\Omega}, \emptyset)$, it is CFVP, it is sufficiently complete with respect to $(\Omega, B_{\Omega}, \emptyset)$, and Σ is preregular below, then the rewrite theory $(\Sigma, B_{\Omega}, R^{\Omega}_{l,r})$ is ground semantically equivalent to $(\Sigma, B \uplus E, R)$.*

The equational theory for Diffie-Hellman of Example 4 is sufficiently complete w.r.t. its constructor subsignature, since any ground term rooted by symbol `exp` is either already using the constructor typing or can be simplified into the constructor typing of `exp`. However, some other theories of interest are not.

Example 9. *Consider the cancellation of encryption and decryption.*

```
fmod DE is
  sorts Msg Key .
```

```
  op enc : Key Msg -> Msg [ctor] .
  op dec : Key Msg -> Msg .
  var K : Key . vars X : Msg .
  eq dec(K,enc(K,X)) = X [variant] .
endfm
```

Given arbitrary keys k_1, k_2 and an arbitrary term a, the term $dec(k_1, enc(k_2, a))$ cannot be reduced.

Terms that cannot be simplified into a constructor term are understood as an erroneous expression and discarded. This is the behaviour of *destructor symbols* in Proverif [25], i.e., functions that may fail. In the rest of the paper, we relax the condition on sufficiently completeness of Theorem 1 and follow the spirit of Proverif's approach[3]: a NF rewrite theory below ensures that erroneous expressions cannot occur in the righthand sides of rewrite rules or in equations, preventing any function to capture that any of its arguments fails. Typical security protocols do however not satisfy the conditions of [92], and in particular all protocols studied in Section 3.5 did not.

Definition 9 (NF Rewrite Theory)**.** *Given a rewrite theory $(\Sigma, B \uplus E, R)$ such that $(\Sigma, B, \vec{E})$ is a decomposition protecting a free constructor decomposition $(\Omega, B_\Omega, \emptyset)$,* erroneous terms *are defined as $\mathscr{E}rr_\bot = \{t \in \mathscr{T}_\Sigma(\mathscr{X}) \mid \nexists \sigma : (t\sigma)\downarrow_{\vec{E},B} \in \mathscr{T}_\Omega(\mathscr{X})\}$ whereas* possibly erroneous terms *are defined as $\mathscr{E}rr_\top = \{t \in \mathscr{T}_\Sigma(\mathscr{X}) \mid \exists \sigma : (t\sigma)\downarrow_{\vec{E},B} \notin \mathscr{T}_\Omega(\mathscr{X})\}$. We say the rewrite theory is* NF *if, for each $l = r \in E$, $l, r \notin \mathscr{E}rr_\bot$ and, for each $l \to r \in R$, $r|_p \in \mathscr{E}rr_\top \implies \exists q : l|_q =_B r|_p$.*

Theorem 2. *Given a NF rewrite theory $(\Sigma, B \uplus E, R)$ such that $(\Sigma, B, \vec{E})$ is a decomposition protecting a free constructor decomposition $(\Omega, B_\Omega, \emptyset)$ and it is CFVP, then any term reachable from a constructor term is also constructor.*

Proof 1. *By induction on the length of the narrowing sequence $t_0 \leadsto^n t_n$. If $n = 0$, then $t_n = t_0$ and t_0 is constructor. If $n > 0$, then $t_0 \leadsto_\sigma t_1 \leadsto^{n-1} t_n$ s.t. $\sigma \in CSU_{E \uplus B}(t_0|_p = l)$ and $t_1 = (t_0[r]_p)\sigma$. Since t_0 is a constructor term, there is no equation applicable to t_0, i.e., $(t_0|_p)\sigma =_B (l\sigma)\downarrow_{\vec{E},B}$. Since t_0 is a constructor term, the bindings in $|_\sigma \mathscr{V}ar(t_0)$ contain only constructor terms. Since erroneous expressions do not appear in the equations E, $|_\sigma \mathscr{V}ar(l)$ contains also constructor terms. Since r does not contain any extra possible erroneous expression, t_1 is constructor. The conclusion follows by the induction hypothesis.* □

[3] A detailed comparison is outside the scope of this paper.

Corollary 1. *Given a NF rewrite theory* $(\Sigma, B \uplus E, R)$ *such that* $(\Sigma, B, \vec{E})$ *is a decomposition protecting a free constructor decomposition* $(\Omega, B_\Omega, \emptyset)$*, it is CFVP, and* Σ *is preregular below, then the rewrite theory* $(\Sigma, B_\Omega, R^\Omega_{l,r})$ *is ground semantically equivalent to* $(\Sigma, B \uplus E, R)$.

We have implemented both the algorithm for computing $[\![t]\!]^\Omega_{\vec{E},B}$ provided in [110] for equational theories that are FVP and the rewrite theory transformation $\mathscr{R} \mapsto \mathscr{R}^\Omega_{l,r}$ from [92]. As far as we know, there was no implementation available of the rewrite theory transformation $\mathscr{R} \mapsto \mathscr{R}^\Omega_{l,r}$ of [92]. We have used $[\![t]\!]^\Omega_{\vec{E},B}$ and $\mathscr{R} \mapsto \mathscr{R}^\Omega_{l,r}$ to create a *protocol* transformation available online at `http://safe-tools.dsic.upv.es/cvtool`. This web page accepts a protocol specification, using the Maude-NPA syntax, and returns the transformed version, including strands and attack patterns. The proof of soundness and completeness of the protocol transformation is omitted but relies on Theorem 1 and Corollary 1. Informally speaking, Maude-NPA internally transforms a protocol specification into a rewrite theory (see Section 3.3). This transformed rewrite theory is then transformed using the program transformation $\mathscr{R} \mapsto \mathscr{R}^\Omega_{l,r}$. And, finally, this resulting rewrite theory is mapped back into a protocol specification. Note that the web page assumes that the conditions of Corollary 1 are satisfied without enforcing them. All the protocols presented in the next section need the relaxed conditions of application of Corollary 1 to safely apply the protocol transformation. These relaxed conditions allow us to deal with more complex protocol specifications efficiently.

3.5 Case studies

This section presents several increasingly complex case studies: Diffie-Hellman protocol in Section 3.5.1, STR protocol in Section 3.5.2, Joux protocol in Section 3.5.3, and TAK protocols in Section 3.5.4. The Joux and TAK protocols use bilinear pairing but TAK4 requires properties beyond the encoding of bilinear pairings available in Tamarin, the only crypto tool with such an equational theory.

3.5.1 The Diffie-Hellman Protocol

In this section, we describe the analysis performed on the Diffie-Hellman (DH) protocol. This protocol was already analysed using Maude-NPA in [54]. DH

uses exponentiation to share a secret between two parties. The description of the protocol using an Alice & Bob notation is given in Figure 3.1.

Alice and Bob agree on a common generator g. Alice sends the generator g raised to the power of a new nonce generated by her. Bob sends the generator g raised to the power of a new nonce generated by him. Both Alice and Bob take the received nonce and raised it to the power of their own respective nonce. The cryptographic property here allows $(g^{N_A})^{N_B} = (g^{N_B})^{N_A} = g^{N_A * N_B}$. This cryptographic property is represented using the equational theory of Example 4.

The informal description of Figure 3.1 is specified using strands as follows. We represent an exponentiation x^y as $\mathtt{exp}(x,y)$. We represent a nonce N_A as $n(A,f)$ where f is a Fresh variable. We have added the identifiers of the participants in the message exchange for clarity. And we have appended a final encryption of some random secret using the generated key to make explicit the different keys used by the honest participants before and after the transformation.

$$\begin{aligned}(\textbf{Alice}) :: f_a, f ::[&+(A;B;exp(g,n(A,f_a))), -(B;A;X),\\ &+(enc(exp(X,n(A,f_a)),sec(A,f)))]\\ (\textbf{Bob}) :: f_b ::[&-(A;B;Y), +(B;A;exp(g,n(B,f_b))),\\ &-(enc(exp(Y,n(B,f_b)),Sr))]\end{aligned}$$

After applying the protocol transformation, we obtain

$$\begin{aligned}(\textbf{Alice}) :: f_a, f ::[&+(A;B;exp(g,n(A,f_a))), -(B;A;exp(G,N)),\\ &+(enc(exp(G,n(A,f_a)*N),sec(A,f)))]\\ (\textbf{Bob}) :: f_b ::[&-(A;B;exp(G,N)), +(B;A;exp(g,n(B,f_b))),\\ &-(enc(exp(G,N*n(B,f_b)),Sr))]\end{aligned}$$

As explained in Example 6, the expression $exp(X{:}Exp, n(A,f_a))$ has only one constructor variant using substitution $X{:}Exp \mapsto exp(G{:}Gen, N{:}NeNonceSet)$. Similarly for $exp(Y{:}Exp, n(B,f_b))$. The duplication of symbols in one defined and one constructor, the coincidence that each defined symbol has only one equation, and the use of associativity and commutativity, makes each strand of the protocols of this paper is replaced by just one strand. This may not always be the case and a strand may be replaced by several new strands (see [92, Example 7]). The Dolev-Yao capabilities for exponentiation are as follows.

$$(\textbf{DY\_exp\_ctor})[-(G{:}Gen),-(N{:}NeNonceSet),+(exp(G{:}Gen,N{:}NeNonceSet)]$$
$$(\textbf{DY\_exp\_func})[-(E{:}Exp),-(N{:}NeNonceSet),+(exp(E{:}Exp,N{:}NeNonceSet)]$$

The second one is transformed as follows

$$(\textbf{DY\_exp\_cvar})[-(exp(G{:}Gen,X{:}NeNonceSet)),-(N{:}NeNonceSet),\\ +(exp(G{:}Gen,X{:}NeNonceSet*N{:}NeNonceSet))]$$

3.5.2 The STR protocol

One extension of the Diffie-Hellman protocol is to consider that every time a new member is joined the exchange key is repeated, allowing for an unbounded number of participants a priori. We consider the three-party group key agreement protocol *STR* from [73], where *STR* is a short name for **S**kinny **TR**ee. The description of the protocol using an informal Alice & Bob notation is given in Figure 3.2. The only difference between the cryptographic properties of STR and DH is that we can have an exponentiation as an exponent, where DH could not. Therefore, the only difference to the equational theory of Example 4 is "`subsort Nonce Exp < NeNonceSet`". The equational theory still satisfies all the conditions of Corollary 1. The informal description of Figure 3.2 is specified using strands as follows, we remove the identifiers of the participants for simplicity.

$$(\textbf{Alice}) :: f_a,f ::[+(exp(g,n(A,f_a))),-(XB),-(XC),\\ +(enc(exp(XC,exp(XB,n(A,f_a))),sec(A,f)))]$$
$$(\textbf{Bob}) :: f_b ::[-(XA),+(exp(g,n(B,f_b))),+(exp(g,exp(XA,n(B,f_b)))),\\ -(XC),-(enc(exp(XC,exp(XA,n(B,f_b))),Sr))]$$
$$(\textbf{Carol}) :: f_c ::[-(XAB),+(exp(g,n(C,f_c))),-(enc(exp(XAB,n(C,f_c)),Sr))]$$

After applying the protocol transformation, we obtain

$$(\textbf{Alice}) :: f_a,f ::[+(exp(g,n(A,f_a))),-(exp(G1,NB)),-(exp(G2,NC)),\\ +(enc(exp(G2,exp(G1,n(A,f_a)*NB)*NC),sec(A,f)))]$$
$$(\textbf{Bob}) :: f_b ::[-(exp(G,NA)),+(exp(g,n(B,f_b))),\\ +(exp(g,exp(G,n(B,f_b)*NA))),-(exp(G,NC)),\\ -(enc(exp(G,exp(G,n(B,f_b)*NA)*NC),Sr))]$$
$$(\textbf{Carol}) :: f_c ::[-(exp(G,NAB)),+(exp(g,n(C,f_c))),\\ -(enc(exp(G,NAB*n(C,f_c)),Sr))]$$

3.5.3 The Joux Protocol

When you want to keep the spirit of the Diffie-Hellman protocol, where no extra sharing is necessary apart of the initial broadcast information, an interesting alternative for three participants is the Joux protocol [70], which relies on bilinear pairing. The description of the protocol using an informal Alice & Bob notation is given in Figure 3.3.

Pairing-based cryptography makes use of a pairing function $\hat{e} : G_1 \times G_2 \rightarrow G_T$ of two cryptographic groups G_1 and G_2 into a third group G_T. Typically, $G_1 = G_2$ and it will be a subgroup of the group of points on an elliptic curve over a finite field, and G_T will be a subgroup of the multiplicative group of a related finite field and the map $\hat{e}$ will be derived from either the Weil or Tate pairing on the elliptic curve. When $G = G_1 = G_2$, the pairing is called *symmetric* and the pairing function $\hat{e}$ is commutative, i.e., if the participants agree on a generator $g \in G$, for any P, Q in G there exist integers i, j s.t. $P = g^i$, $Q = g^j$, $\hat{e}(P,Q) = \hat{e}(g^i, g^j) = \hat{e}(g,g)^{i*j} = \hat{e}(g^j, g^i) = \hat{e}(Q,P)$. In Figure 3.3, we follow the syntax of [70] and use letter P as the agreed generator. We write aP instead of P^a for P added to itself a times, also called scalar multiplication of P by a. Note that we write `[a]P` in the equational theory below for clarification. The bilinear pairing is specified as follows.

```
fmod BP-CFVP is
  sorts Nonce NeNonceSet Gen GenP Exp ExpP .
  subsort Nonce < NeNonceSet .
  op exp : Gen NeNonceSet -> Exp [ctor] .
  op exp : Exp NeNonceSet -> Exp .
  op _*_ : NeNonceSet NeNonceSet -> NeNonceSet [assoc comm ctor] .
  op p : -> GenP [ctor] .
  op em : GenP GenP -> Gen [ctor comm] .
  op em : ExpP ExpP -> Exp [comm] .
  op [_]_ : NeNonceSet GenP -> ExpP [ctor] .
  op [_]_ : NeNonceSet ExpP -> ExpP .
  var X : Gen . vars Y Z : NeNonceSet . vars P Q : GenP .
  eq exp(exp(X,Y),Z) = exp(X,Y * Z) [variant] .
  eq [Z]([Y]P) = [Z * Y]P [variant] .
  eq em([Y]P, [Z]Q) = exp(em(P,Q),Y * Z) [variant] .
endfm
```

We adapted the built-in theory of bilinear pairing of Tamarin [21, 108] to satisfy[4] the conditions of Corollary 1. The informal description of Figure 3.3

[4]Confluence is proved by the absence of critical pairs between the lefthand sides of the three equa-

is specified using strands as follows.

$$\begin{aligned}(\textbf{Alice}) :: f_a, f ::[&+([n(A,f_a)]p), -(XB), -(XC),\\ &+(enc(exp(em(XB,XC),n(A,f_a)),sec(A,f))]\\ (\textbf{Bob}) :: f_b ::[&-(XA), +([n(B,f_b)]p), -(XC),\\ &-(enc(exp(em(XA,XC),n(B,f_b)),Sr)]\\ (\textbf{Carol}) :: f_c ::[&-(XA), -(XB), +([n(C,f_c)]p),\\ &-(enc(exp(em(XA,XB),n(C,f_c)),Sr)]\end{aligned}$$

After applying the protocol transformation, we obtain

$$\begin{aligned}(\textbf{Alice}) :: f_a, f ::[&+([n(A,f_a)]p), -([NB]PB), -([NC]PC),\\ &+(enc(exp(em(PB,PC),n(A,f_a)*NB*NC),sec(A,f))]\\ (\textbf{Bob}) :: f_b ::[&-([NA]PA), +([n(B,f_b)]p), -([NC]PC),\\ &+(enc(exp(em(PA,PC),n(B,f_b)*NA*NC),Sr)]\\ (\textbf{Carol}) :: f_c ::[&-([NA]PA), -([NB]PB), +([n(C,f_c)]p),\\ &+(enc(exp(em(PA,PB),n(C,f_c)*NA*NB),Sr)]\end{aligned}$$

3.5.4 The TAK Group Protocols

The Tripartite Authenticated Key group protocols [10] is a set of authenticated key agreement protocols that still require only one round of communication. It is an improvement of the Joux protocol. The four versions of TAK share the same exchanged message but the computation key is different for each version. The description of the TAK protocol using an informal Alice & Bob notation is given in Figure 3.4. However, the four different ways of computing the keys are given in Figures 3.4, 3.5, 3.6, and 3.7. These four protocols use the bilinear pairing cryptographic properties explained in Section 3.5.3 plus a hash function h and the following additive property (and its symmetric version, since $\hat{e}$ is commutative)

$$\hat{e}(Q, W+Z) = \hat{e}(Q,W)\cdot\hat{e}(Q,Z) \tag{3.1}$$

where $+$ is the additive symbol for the group G and $\cdot$ is the additive symbol for the group G_T given $\hat{e} : G \times G \to G_T$. These properties are specified[5] as follows.

tions. Termination and FVP are proved by strongly right-irreducibility [60], i.e., righthand sides do not unify with any lefthand side. CFVP is proved because it is preregular below.

[5]The additive property (3.1) is not supported by the bilinear pairing of Tamarin [21, 108].

```
fmod BPAdd-CFVP is
  sorts Nonce NeNonceSet Gen GenP Exp ExpP ExpT .
  subsort Nonce < NeNonceSet . subsort Exp < ExpT .
  op exp : Gen NeNonceSet -> Exp [ctor] .
  op exp : Exp NeNonceSet -> Exp .
  op _*_ : NeNonceSet NeNonceSet -> NeNonceSet [ctor assoc comm] .
  op p : -> GenP [ctor] .
  op em : GenP GenP -> Gen [ctor comm] .
  op em : ExpP ExpP -> Exp [comm] .
  op [_]_ : NeNonceSet GenP -> ExpP [ctor] .
  op [_]_ : NeNonceSet ExpP -> ExpP .
  op _+_ : NeNonceSet NeNonceSet -> NeNonceSet [ctor assoc comm] .
  op _+_ : ExpP ExpP -> ExpP .
  op _·_ : ExpT ExpT -> ExpT [ctor assoc comm] .
  var X : Gen . vars Y Z : NeNonceSet . vars P Q : GenP .
  eq exp(exp(X,Y),Z) = exp(X,Y * Z) [variant] .
  eq [Z]([Y]P) = [Z * Y]P [variant] .
  eq em([Y]P, [Z]Q) = exp(em(P,Q),Y * Z) [variant] .
  eq ([Y]P) + ([Z]P) = [Y + Z]P [variant] .
endfm
```

Note that Property 3.1 does not appear explicitly in the equational theory above and it is transformed as follows. The addition symbol + is split into two versions, one of them being an associative-commutative constructor and the other one being a defined symbol. A new equation relating these two versions of + is added. And symbol · is simply represented as an associative-commutative constructor. The last, new equation denotes a homomorphic addition and it is easily handled by variant-based unification because it is defined on disconnected sorts ExpP and NeNonceSet (see [113] for approximations of homomorphism following the same idea). For example, the key generated by Alice in TAK4

$$K_A = exp(em([b]p + [h([b]p;[y]p) * y]p, [c]p + [h([c]p;[z]p) * z]p), \\ a + (h([a]p;[x]p) * x))$$

is transformed into the common key

$$K_{ABC} = exp(em(p,p),(a + (h([a]p;[x]p) * x)) * \\ (b + (h([b]p;[y]p) * y)) * (c + (h([c]p;[z]p) * z)))$$

by applying the last equation two times, followed by the third and the first equations (we underline the replaced subterm)

$$
\begin{aligned}
&exp(em(\underline{[b]p + [h([b]p;[y]p) * y]p}, \underline{[c]p + [h([c]p;[z]p) * z]p}),\\
&\quad a + (h([a]p;[x]p) * x)) =\\
&exp(\underline{em([b + (h([b]p;[y]p) * y)]p, [c + (h([c]p;[z]p) * z)]p)},\\
&\quad a + (h([a]p;[x]p) * x))) =\\
&\underline{exp(exp(em(p,p), a + (h([a]p;[x]p) * x),}\\
&\quad \underline{(b + h([b]p;[y]p) * y) * (c + h([c]p;[z]p) * z))} =\\
&exp(em(p,p), (a + (h([a]p;[x]p) * x)) *\\
&\quad (b + (h([b]p;[y]p) * y)) * (c + (h([c]p;[z]p) * z)))
\end{aligned}
$$

If the non-constructor version of + becomes associative-commutative, then the theory is not FVP. This equational theory works for all the TAK protocols even if it is not the most general possible; it is left for future work whether Property 3.1 can be encoded directly. This equational theory satisfies the conditions of Corollary 1. The original and transformed versions of TAK1, TAK2, and TAK3 are omitted but are available online. The informal description of the TAK4 protocol given in Figure 3.7 is specified using strands as follows.

$$
\begin{aligned}
(\textbf{Alice}) :: f_a, f_x, f :: [&+([n(A,f_a)]p), +([n(A,f_x)]p), -(BP), -(YP), -(CP), -(ZP),\\
&+(enc(exp(\hat{e}(BP + [h(BP;YP)]YP, CP + [h(CP;ZP)]ZP),\\
&\quad f_a + h([n(A,f_a)]p; [n(A,f_x)]p * f_x)), sec(A,f)))]
\end{aligned}
$$

$$
\begin{aligned}
(\textbf{Bob}) :: f_b, f_y :: [&-(AP), -(XP), +([n(B,f_b)]p), +([n(B,f_y)]p), -(CP), -(ZP),\\
&-(enc(exp(\hat{e}(AP + [h(AP;XP)]XP, CP + [h(CP;ZP)]ZP),\\
&\quad f_b + h([n(B,f_b)]p; [n(B,f_y)]p * f_y)), Sr))]
\end{aligned}
$$

$$
\begin{aligned}
(\textbf{Carol}) :: f_c, f_z :: [&-(AP), -(XP), -(BP), -(YP), +([n(C,f_c)]p), +([n(C,f_z)]p),\\
&-(enc(exp(\hat{e}(AP + [h(AP;XP)]XP, BP + [h(BP;YP)]YP),\\
&\quad f_c + h([n(C,f_c)]p; [n(C,f_c)]p * f_c)), Sr))]
\end{aligned}
$$

After applying the protocol transformation, we obtain

$$
\begin{aligned}
(\textbf{Alice}) :: f_a, f_x, f :: [&+([n(A,f_a)]p), +([n(A,f_x)]p),\\
&-([NB]PB), -([NY]PB), -([NC]PC), -([NZ]PC),\\
&+(enc(exp(\hat{e}(PB,PC), (NB + (h([NB]PB;[NY]PB) * NY))\\
&\quad * (NC + (h([NC]PC;[NZ]PC) * NZ))\\
&\quad * (f_a + h([n(A,f_a)]p; [n(A,f_x)]p * f_x))), sec(A,f)))]
\end{aligned}
$$

		Before Transformation		*After Transformation*			
Protocol	*Property*	*States*	*Time (ms)*	*States*	*Time (ms)*	*States (%)*	*Speedup*
DH	auth	137	308,066	111	132,756	81.02	2.32
	secrecy	138	322,731	104	142,015	75.36	2.27
STR	auth	34	43,144	31	16,010	91.18	2.69
	secrecy	250	1,016,469	117	408,960	46.80	2.49
Joux	auth	38	85,579	37	30,012	97.37	2.85
	secrecy	55	247,712	58	78,384	105.45	3.16
TAK1	secrecy	25	259,619	20	126,998	80.00	2.04
TAK2	secrecy	67	365,797	46	152,842	68.66	2.39
TAK3	secrecy	117	670,775	67	216,350	57.26	3.10
TAK4	secrecy	57	371,770	48	181,850	84.21	2.04

Table 3.1: Experimental results for the transformed protocols.

$$
\begin{aligned}
(\textbf{Bob}) :: f_b, f_y :: [& -([NA]PA), -([NX]PA), +([n(B,f_b)]p), +([n(B,f_b)]p), \\
& -([NC]PC), -([NZ]PC), \\
& -(enc(exp(\hat{e}(PA,PC), (NA + (h([NA]PA; [NX]PA) * NX)) \\
& \qquad * (NC + (h([NC]PC; [NZ]PC) * NZ)) \\
& \qquad * (f_b + h([n(B,f_b)]p; [n(B,f_y)]p * f_y))), Sr))]
\end{aligned}
$$

$$
\begin{aligned}
(\textbf{Carol}) :: f_c, f_z :: [& -([NA]PA), -([NX]PA), -([NB]PB), -([NB]PB), \\
& +([n(C,f_c)]p), +([n(C,f_c)]p), \\
& -(enc(exp(\hat{e}(PA,PB), (NA + (h([NA]PA; [NX]PA) * NX)) \\
& \qquad * (NB + (h([NB]PB; [NY]PB) * NY)) \\
& \qquad * (f_c + h([n(C,f_c)]p; [n(C,f_z)]p * f_z))), Sr))]
\end{aligned}
$$

3.6 Experiments

We have evaluated all the protocols of Section 3.5, both before and after the transformation. For DH, STR and Joux, we consider two general attack patterns, one for authentication and another for secrecy of the session key. For TAKs we consider only a secrecy attack pattern. Both properties of DH are insecure, authentication of STR is insecure but secrecy is secure [73], both properties of Joux are insecure [70], and TAK1, TAK2, TAK3, and TAK4 are secure [10].

In Table 3.1, we report both the number of states and the generation time of the search space associated to each attack pattern. The transformation itself

is almost immediate, since the equational theories in these examples are not so complex. The time and space difference is shown in columns *States (%)* and *Speedup*. These columns demonstrate that the difference between verifying the protocol with all the crypto properties and verifying the protocol with a minimal set of the crypto properties is remarkable in three different aspects. First, for the STR protocol, the transformed protocol produces only 46.80% of the total number of states of the untransformed version. Second, for the TAK3 protocol, the execution time of the transformed protocol is three times faster than the untransformed version. Third, for the Joux protocol, even if the analysis of the transformed protocol produces more states than the analysis of the untransformed protocol, the execution time is three times faster.

All the experiments were conducted on a PC with a 3.3GHz Intel Xeon E5-1660 and 64GB RAM. We used Maude v3.0 [35] and Maude-NPA v3.1.4 [2]. The protocol specifications of both before and after the transformation and the output of each analysis are available at `http://safe-tools.dsic.upv.es/cvtool`.

3.7 Conclusions

Our first contribution is a protocol transformation that can safely get rid of cryptographic properties under some mild conditions. We have demonstrated with experiments that the time and space difference between verifying the protocol with all the crypto properties and verifying the protocol with a minimal set of the crypto properties is remarkable (an average speedup of 2.54). A similar idea is presented in [77] for XOR and in [76] for DH. These works are however not comparable to ours, since they are not protocol transformations but classes of protocols where the analysis using Proverif is sound. In [66], protocol transformations are studied. However the goal is not to optimize the verification, but to ensure that a transformed protocol satisfies some security goals, when the source protocol did, focusing on incremental protocol construction. Our second contribution is an encoding of the theory of bilinear pairing into Maude-NPA. This encoding goes beyond the encoding of bilinear pairing available in the Tamarin tool, the only crypto tool with such an equational theory. Since Tamarin [44] and AKISS [17] use term variants, they could be adapted to use both our protocol transformation and our encoding of the theory of bilinear pairing. They may even be useful for other crypto tools with more limited crypto properties such as ProVerif [25], OFMC [22],

Scyther [40] or Scyther-proof [3]. Specially, since Proverif [25] already incorporated the notion of destructors and constructors time ago. As future work, we plan to study how the protocol transformation applies to other families of protocols and crypto properties such as homomorphisms [113].

$$A \longrightarrow B : g^{N_a}$$
$$B \longrightarrow A : g^{N_b}$$
$$K_A = (g^{N_b})^{N_a}$$
$$K_B = (g^{N_a})^{N_b}$$
$$K_{AB} = g^{N_a * N_b}$$

Figure 3.1: DH

$$A \longrightarrow B : g^{N_a}$$
$$B \longrightarrow A : g^{N_b}$$
$$B \longrightarrow C : g^{((g^{N_b})^{N_a})}$$
$$C \longrightarrow A, B : g^{N_c}$$
$$K_A = (g^{N_c})^{(g^{((g^{N_b})^{N_a})})}$$
$$K_B = (g^{N_c})^{(g^{((g^{N_a})^{N_b})})}$$
$$K_C = (g^{((g^{N_b})^{N_a})})^{N_c}$$
$$K_{ABC} = g^{\underbrace{g^{N_a * N_b}}_{} * N_c}$$

Figure 3.2: STR

$$A \longrightarrow B, C : aP$$
$$B \longrightarrow A, C : bP$$
$$C \longrightarrow A, B : cP$$
$$K_A = \hat{e}(bP, cP)^a$$
$$K_B = \hat{e}(aP, cP)^b$$
$$K_C = \hat{e}(aP, bP)^c$$
$$K_{ABC} = \hat{e}(P, P)^{a*b*c}$$

Figure 3.3: Joux

$$A \longrightarrow B, C : \ aP; \{xP\}_A$$
$$B \longrightarrow A, C : \ bP; \{yP\}_B$$
$$C \longrightarrow A, B : \ cP; \{zP\}_C$$

$$K_A = h(\hat{e}(bP, cP)^a; \hat{e}(yP, zP)^x)$$
$$K_B = h(\hat{e}(aP, cP)^b; \hat{e}(xP, zP)^y)$$
$$K_C = h(\hat{e}(aP, bP)^c; \hat{e}(xP, yP)^z)$$
$$K_{ABC} = h(\hat{e}(P, P)^{a*b*c}; \hat{e}(P, P)^{x*y*z})$$

Figure 3.4: TAK1

$$K_A = \hat{e}(bP, zP)^a \cdot \hat{e}(yP, cP)^a \cdot \hat{e}(bP, cP)^x$$
$$K_B = \hat{e}(aP, zP)^b \cdot \hat{e}(xP, cP)^b \cdot \hat{e}(aP, cP)^y$$
$$K_C = \hat{e}(aP, yP)^c \cdot \hat{e}(xP, bP)^c \cdot \hat{e}(aP, bP)^z$$
$$K_{ABC} = \hat{e}(P, P)^{a*y*c} \cdot \hat{e}(P, P)^{x*b*c} \cdot \hat{e}(P, P)^{a*b*z}$$

Figure 3.5: TAK2

$$K_A = \hat{e}(yP, cP)^x \cdot \hat{e}(bP, zP)^x \cdot \hat{e}(yP, zP)^a$$
$$K_B = \hat{e}(aP, zP)^y \cdot \hat{e}(xP, cP)^y \cdot \hat{e}(xP, zP)^b$$
$$K_C = \hat{e}(aP, yP)^z \cdot \hat{e}(xP, bP)^z \cdot \hat{e}(xP, yP)^c$$
$$K_{ABC} = \hat{e}(P, P)^{x*y*c} \cdot \hat{e}(P, P)^{x*b*z} \cdot \hat{e}(P, P)^{a*y*z}$$

Figure 3.6: TAK3

$$K_A = \hat{e}(bP + h(bP; yP)yP, cP + h(cP; zP)zP)^{a+(h(aP;xP)*x)}$$
$$K_B = \hat{e}(aP + h(aP; xP)xP, cP + h(cP; zP)zP)^{b+(h(bP;yP)*y)}$$
$$K_C = \hat{e}(aP + h(bP; yP)yP, bP + h(bP; yP)yP)^{a+(h(cP;cP)*c)}$$
$$K_{ABC} = \hat{e}(P, P)^{(a+(h(aP;xP)*x))*(b+(h(bP;yP)*y))*(c+(h(cP;zP)*z))}$$

Figure 3.7: TAK4

CHAPTER 4

Protocol Analysis with Time

Damián Aparicio-Sánchez[1], Santiago Escobar[1], Catherine Meadows[2]
José Meseguer[3], Julia Sapiña[1]

[1] VRAIN, Universitat Politècnica de València, Spain
{daapsnc,sescobar,jsapina}@upv.es

[2] Naval Research Laboratory, Washington DC USA
meadows@itd.nrl.navy.mil

[3] University of Illinois at Urbana-Champaign, USA
meseguer@illinois.edu

Abstract We present a framework suited to the analysis of cryptographic protocols that make use of time in their execution. We provide a process algebra syntax that makes time information available to processes, and a transition semantics that takes account of fundamental properties of time. Additional properties can be added by the user if desirable. This timed protocol framework can be implemented either as a simulation tool or as a symbolic analysis tool in which time references are represented by logical variables, and in which the properties of time are implemented as constraints on those time logical variables. These constraints are carried along the symbolic execution of the protocol. The satisfiability of these constraints can be evaluated as the analysis proceeds, so attacks that

This paper was partially supported by the EU (FEDER) and the Spanish MCIU under grant RTI2018-094403-B-C32, by the Spanish Generalitat Valenciana under grant PROMETEO/2019/098 and APOSTD/2019/127, by the US Air Force Office of Scientific Research under award number FA9550-17-1-0286, and by ONR Code 311

violate the laws of physics can be rejected as impossible. We demonstrate the feasibility of our approach by using the Maude-NPA protocol analyzer together with an SMT solver that is used to evaluate the satisfiability of timing constraints. We provide a sound and complete protocol transformation from our timed process algebra to the Maude-NPA syntax and semantics, and we prove its soundness and completeness. We then use the tool to analyze Mafia fraud and distance hijacking attacks on a suite of distance-bounding protocols.

4.1 Introduction

Time is an important aspect of many cryptographic protocols, and there has been increasing interest in the formal analysis of protocols that use time. Model checking of protocols that use time can be done using either an explicit time model, or by using an untimed model and showing it is sound and complete with respect to a timed model. The former is more intuitive for the user, but the latter is often chosen because not all cryptographic protocol analysis tools support reasoning about time. In this paper we describe a solution that combines the advantages of both approaches. An explicit timed specification language is developed with a timed syntax and semantics, and is automatically translated to an existing untimed language. The user however writes protocol specifications and queries in the timed language. In this paper we describe how such an approach has been applied to the Maude-NPA tool by taking advantage of its built-in support for constraints. We believe that this approach can be applied to other tools that support constraint handling as well.

There are a number of security protocols that make use of time. In general, there are two types: those that make use of assumptions about time, most often assuming some sort of loose synchronization, and those that guarantee these assumptions. The first kind includes protocols such as Kerberos [95], which uses timestamps to defend against replay attacks, the TESLA protocol [101], which relies on loose synchronization to amortize digital signatures, and blockchain protocols, which use timestamps to order blocks in the chain. The other kind provides guarantees based on physical properties of time: for example, distance bounding, which guarantees that a prover is within a certain distance of a verifier, and secure time synchronization, which guarantees that the clocks of two different nodes are synchronized within a certain margin of error. In this paper, we concentrate on protocols using distance bounding,

both because it has been well-studied, and because the timing constraints are relatively simple.

A number of approaches have been applied to the analysis of distance bounding protocols. In [83], an epistemic logic for distance bounding analysis is presented where timing is captured by means of *timed channels*, which are described axiomatically. Time of sending and receiving messages can be deduced by using these timed channel axioms. In [20], Basin et al. define a formal model for reasoning about physical properties of security protocols, including timing and location, which they formalize in Isabelle/HOL and use it to analyze several distance bounding protocols, by applying a technique similar to Paulson's inductive approach [100]. In [41], Debant et al. develop a timing model for AKiSS, a tool for verifying protocol equivalence in the bounded session model, and use it to analyze distance bounding protocols. In [98], Nigam et al. develop a model of timing side channels in terms of constraints and use it to define a timed version of observational equivalence for protocols. They have developed a tool for verifying observational equivalence that relies on SMT solvers. Other work concentrates on simplifying the problem so it can be more easily analyzed by a model checker, but proving that the simple problem is sound and complete with respect to the original problem so that the analysis is useful. In this regard, Nigam et al. [97] and Debant et al. [42] show that it is safe to limit the size and complexity of the topologies, and Mauw et al. [81] and Chothia et al. [34] develop timed and untimed models and show that analysis in the untimed model is sound and complete with respect to the timed model.

In this paper we illustrate our approach by developing a timed protocol semantics suitable for the analysis of protocols that use constraints on time and distance, such as distance bounding, and that can be implemented as either a simulation tool for generating and checking concrete configurations, or as a symbolic analysis tool that allows the exploration of all relevant configurations. We realize the timed semantics by translating it into the semantics of the Maude-NPA protocol analysis tool, in which timing properties are expressed as constraints. The constraints generated during the Maude-NPA search are then checked using an SMT solver.

There are several things that help us. One is that we consider a metric space with distance constraints. Many tools support constraint handling, e.g., Maude-NPA [57] and Tamarin [85]. Another is that time can be naturally added to a process algebra. Many tools support processes, e.g., Maude-NPA [115] and AKISS [41].

The rest of this paper is organized as follows. In Section 4.2, we recall the Brands-Chaum protocol, which is used as the running example throughout the paper. In Section 4.3, we present the timed process algebra with its intended semantics. In Section 4.4, we present a sound and complete protocol transformation from our timed process algebra to an untimed process algebra. In Section 4.5, we show how our timed process algebra can be transformed into Maude-NPA strand notation. In Section 4.6, we present our experiments. We conclude in Section 4.7.

4.2 The Brands-Chaum distance bounding protocol

In the following, we recall the Brands-Chaum distance bounding protocol of [27], which we will use as the running example for the whole paper.

Example 10. *The Brands-Chaum protocol specifies communication between a verifier V and a prover P. P needs to authenticate itself to V, and also needs to prove that it is within a distance "d" of it. X;Y denotes concatenation of two messages X and Y, commit(N,Sr) denotes commitment of secret Sr with a nonce N, open(N,Sr,C) denotes opening a commitment C using the nonce N and checking whether it carries the secret Sr, $\oplus$ is the exclusive-or operator, and sign(A,M) denotes A signing message M. A typical interaction between the prover and the verifier is as follows:*

$P \to V : commit(N_P, S_P)$
//The prover sends his name and a commitment

$V \to P : N_V$
//The verifier sends a nonce
//and records the time when this message was sent

$P \to V : N_P \oplus N_V$
//The verifier checks the answer of this exclusive-or
//message arrives within two times a fixed distance

$P \to V : S_P$
//The prover sends the committed secret
//and the verifier checks $open(N_P, S_P, commit(N_P, S_P))$

$P \to V : sign_P(N_V; N_P \oplus N_V)$
//The prover signs the two rapid exchange messages

The previous informal Alice&Bob notation can be naturally extended to include time. We consider wireless communication between the participants located at an arbitrary given topology (participants do not move from their assigned locations) with distance constraints, where time and distance are

equivalent for simplification and are represented by a real number. We assume a metric space with a distance function $d : A \times A \to \mathit{Real}$ from a set A of participants such that $d(A,A) = 0$, $d(A,B) = d(B,A)$, and $d(A,B) \leq d(A,C) + d(C,B)$. Then, time information is added to the protocol. First, we add the time when a message was sent or received as a subindex $P_{t_1} \to V_{t_2}$. Second, time constraints associated to the metric space are added: (i) the sending and receiving times of a message differ by the distance between them and (ii) the time difference between two consecutive actions of a participant must be greater or equal to zero. Third, the distance bounding constraint of the verifier is represented as an arbitrary distance d. Time constraints are written using quantifier-free formulas in linear real arithmetic. For convenience, in linear equalities and inequalities (with $<$, $\leq$, $>$ or $\geq$), we allow both $2 * x = x + x$ and the monus function $x \dot{-} y = \mathit{if}\ y < x\ \mathit{then}\ x - y\ \mathit{else}\ 0$ as definitional extensions.

In the following timed sequence of actions, a vertical bar is included to differentiate between the process and some constraints associated to the metric space. We remove the constraint $\mathit{open}(N_P, S_P, \mathit{commit}(N_P, S_P))$ for simplification.

$$
\begin{array}{rll}
P_{t_1} \to V_{t'_1} : \mathit{commit}(N_P, S_P) & \mid t'_1 = t_1 + d(P,V) \\
V_{t_2} \to P_{t'_2} : N_V & \mid t'_2 = t_2 + d(P,V) \wedge t'_1 \geq t'_1 \\
P_{t_3} \to V_{t'_3} : N_P \oplus N_V & \mid t'_3 = t_3 + d(P,V) \wedge t_3 \geq t'_2 \\
V : t'_3 \dot{-} t_2 \leq 2 * d & \\
P_{t_4} \to V_{t'_4} : S_P & \mid t'_4 = t_4 + d(P,V) \wedge t_4 \geq t_3 \wedge t'_4 \geq t'_3 \\
P_{t_5} \to V_{t'_5} : \mathit{sign}_P(N_V; N_P \oplus N_V) & \mid t'_5 = t_5 + d(P,V) \wedge t_5 \geq t_4 \wedge t'_5 \geq t'_4
\end{array}
$$

The Brands-Chaum protocol is designed to defend against mafia frauds, where an honest prover is outside the neighborhood of the verifier (i.e., $d(P,V) > d$) but an intruder is inside (i.e., $d(I,V) \leq d$), pretending to be the honest prover. The following is an example of an *attempted* mafia fraud, in which the intruder simply forwards messages back and forth between the prover and the verifier. We write $I(P)$ to denote an intruder pretending to be an honest prover P.

$$
\begin{array}{rll}
P_{t_1} \to I_{t_2} & : \mathit{commit}(N_P, S_P) & \mid t_2 = t_1 + d(P,I) \\
I(P)_{t_2} \to V_{t_3} & : \mathit{commit}(N_P, S_P) & \mid t_3 = t_2 + d(V,I) \\
V_{t_3} \to I(P)_{t_4} & : N_V & \mid t_4 = t_3 + d(V,I) \\
I_{t_4} \to P_{t_5} & : N_V & \mid t_5 = t_4 + d(P,I) \\
P_{t_5} \to I_{t_6} & : N_P \oplus N_V & \mid t_6 = t_5 + d(P,I) \\
I(P)_{t_6} \to V_{t_7} & : N_P \oplus N_V & \mid t_7 = t_6 + d(V,I) \\
V & : t_7 \dot{-} t_3 \leq 2 * d & \\
P_{t_8} \to I_{t_9} & : S_P & \mid t_9 = t_8 + d(P,I) \wedge t_8 \geq t_5 \\
I(P)_{t_{10}} \to V_{t_{11}} & : S_P & \mid t_{11} = t_{10} + d(V,I) \wedge t_{11} \geq t_7 \\
I(P)_{t_{12}} \to V_{t_{13}} & : \mathit{sign}_P(N_V; N_P \oplus N_V) & \mid t_{13} = t_{12} + d(V,I) \wedge t_{13} \geq t_{11}
\end{array}
$$

Note that, in order for this trace to be consistent with the metric space, it would require that $2*d(V,I)+2*d(P,I) \leq 2*d$, which is unsatisfiable by $d(V,P) > d > 0$ and the triangular inequality $d(V,P) \leq d(V,I)+d(P,I)$, which implies that the attack is not possible.

However, a distance hijacking attack is possible (i.e., the time and distance constraints are satisfiable) where an intruder located outside the neighborhood of the verifier (i.e., $d(V,I) > d$) succeeds in convincing the verifier that he is inside the neighborhood by exploiting the presence of an honest prover in the neighborhood (i.e., $d(V,P) \leq d$) to achieve his goal. The following is an example of a *successful* distance hijacking, in which the intruder listens to the exchanges messages between the prover and the verifier but builds the last message.

$$\begin{array}{rll}
P_{t_1} \rightarrow V_{t_2} & : commit(N_P, S_P) & \mid t_2 = t_1 + d(P,V) \\
V_{t_2} \rightarrow P_{t_3}, I_{t'_3} & : N_V & \mid t_3 = t_2 + d(P,V) \wedge t'_3 = t_2 + d(I,V) \\
P_{t_3} \rightarrow V_{t_4}, I_{t'_4} & : N_P \oplus N_V & \mid t_4 = t_3 + d(P,V) \wedge t'_4 = t_3 + d(I,V) \\
V & : t_4 \dot{-} t_2 \leq 2*d & \\
P_{t_5} \rightarrow V_{t_6} & : S_P & \mid t_6 = t_5 + d(P,V) \wedge t_5 \geq t_3 \wedge t_6 \geq t_4 \\
I(P)_{t_7} \rightarrow V_{t_8} & : sign_I(N_V; N_P \oplus N_V) & \mid t_8 = t_7 + d(I,V) \wedge t_7 \geq t'_4 \wedge t_8 \geq t_6
\end{array}$$

4.3 A Timed Process Algebra

In this section, we present our timed process algebra and its intended semantics. We restrict ourselves to a semantics that can be used to reason about time and distance. We discuss how this could be extended in Section 4.7. To illustrate our approach, we use Maude-NPA's process algebra and semantics described in [115], extending it with a global clock and time information.

4.3.1 New Syntax for Time

In our timed protocol process algebra, the behaviors of both honest principals and the intruders are represented by *labeled processes*. Therefore, a protocol is specified as a set of labeled processes. Each process performs a sequence of actions, namely sending $(+m)$ or receiving $(-m)$ a message m, but without knowing who actually sent or received it. Each process may also perform deterministic or non-deterministic choices. We define a protocol $\mathscr{P}$ in the timed protocol process algebra, written $\mathscr{P}_{TPA}$, as a pair of the form $\mathscr{P}_{TPA} = ((\Sigma_{TPA_{\mathscr{P}}}, E_{TPA_{\mathscr{P}}}), P_{TPA})$, where $(\Sigma_{TPA_{\mathscr{P}}}, E_{TPA_{\mathscr{P}}})$ is the equational theory

specifying the equational properties of the cryptographic functions and the state structure, and P_{TPA} is a $\Sigma_{TPA/\mathscr{E}}$-term denoting a *well-formed* timed process. The timed protocol process algebra's syntax Σ_{TPA} is parameterized by a sort Msg of messages. Moreover, time is represented by a new sort Real, since we allow conditional expressions on time using linear arithmetic for the reals.

Similar to [115], processes support four different kinds of choice: (i) a process expression P ? Q supports *explicit non-deterministic choice* between P and Q; (ii) a choice variable $X_?$ appearing in a send message expression +m supports *implicit non-deterministic choice* of its value, which can furthermore be an *unbounded* non-deterministic choice if $X_?$ ranges over an infinite set; (iii) a conditional *if* C *then* P *else* Q supports *explicit deterministic choice* between P and Q determined by the result of its condition C; and (iv) a receive message expression $-m(X_1,...,X_n)$ supports *implicit deterministic* choice about accepting or rejecting a received message, depending on whether or not it matches the pattern $m(X_1,...,X_n)$. This deterministic choice is implicit, but it could be made explicit by replacing $-m(X_1,...,X_n) \cdot P$ by the semantically equivalent conditional expression $-X.$ *if* $X = m(X_1,...,X_n)$ *then* P *else* $nilP \cdot P$, where X is a variable of sort Msg, which therefore accepts any message.

The timed process algebra has the following syntax, also similar to that of [115] plus the addition of the suffix *@Real* to the sending and receiving actions:

$$\begin{aligned} \mathit{ProcConf} &::= \mathit{LProc} \mid \mathit{ProcConf}\ \&\ \mathit{ProcConf} \mid \emptyset \\ \mathit{ProcId} &::= (\mathit{Role}, \mathit{Nat}) \\ \mathit{LProc} &::= (\mathit{ProcId}, \mathit{Nat})\ \mathit{Proc} \\ \mathit{Proc} &::= \mathit{nilP} \mid +(\mathit{Msg}@\mathit{Real}) \mid -(\mathit{Msg}@\mathit{Real}) \mid \mathit{Proc} \cdot \mathit{Proc} \mid \\ &\quad \mathit{Proc}\ ?\ \mathit{Proc} \mid \mathit{if}\ \mathit{Cond}\ \mathit{then}\ \mathit{Proc}\ \mathit{else}\ \mathit{Proc} \end{aligned}$$

- *ProcConf* stands for a *process configuration*, i.e., a set of labeled processes, where the symbol & is used to denote set union for sets of labeled processes.

- *ProcId* stands for a *process identifier*, where *Role* refers to the role of the process in the protocol (e.g., prover or verifier) and *Nat* is a natural number denoting the identity of the process, which distinguishes different instances (sessions) of a process specification.

- *LProc* stands for a *labeled process*, i.e., a process *Proc* with a label (ProcId, J). For convenience, we sometimes write (Role, I, J), where J

indicates that the action at stage J of the process $(Role, I)$ will be the next one to be executed, i.e., the first $J-1$ actions of the process for role *Role* have already been executed. Note that the I and J of a process $(Role, I, J)$ are omitted in a protocol specification.

- *Proc* defines the actions that can be executed within a process, where $+Msg@T$, and $-Msg@T$ respectively denote sending out a message or receiving a message *Msg*. Note that T must be a variable where the underlying metric space determines the exact sending or receiving time, which can be used later in the process. Moreover, "$Proc \cdot Proc$" denotes *sequential composition* of processes, where symbol _._ is associative and has the empty process *nilP* as identity. Finally, "$Proc \ ? \ Proc$" denotes an explicit *nondeterministic choice*, whereas "*if Cond then Proc else Proc*" denotes an explicit *deterministic choice*, whose continuation depends on the satisfaction of the constraint *Cond*. Note that choice is explicitly represented by either a non-deterministic choice between $P_1 \ ? \ P_2$ or by the deterministic evaluation of a conditional expression *if Cond then* P_1 *else* P_2, but it is also implicitly represented by the instantiation of a variable in different runs.

In all process specifications we assume four disjoint kinds of variables, similar to the variables of [115] plus time variables:

- ***fresh variables***: each one of these variables receives a *distinct constant value* from a data type $\mathsf{V}_{\mathsf{fresh}}$, denoting unguessable values such as nonces. Throughout this paper we will denote this kind of variables as $f, f_1, f_2, \ldots$.

- ***choice variables***: variables first appearing in a *sent message* $+M$, which can be substituted by any value arbitrarily chosen from a possibly infinite domain. A choice variable indicates an *implicit non-deterministic choice*. Given a protocol with choice variables, each possible substitution of these variables denotes a possible run of the protocol. We always denote choice variables by letters postfixed with the symbol "?" as a subscript, e.g., $A_?, B_?, \ldots$.

- ***pattern variables***: variables first appearing in a *received message* $-M$. These variables will be instantiated when matching sent and received messages. *Implicit deterministic choices* are indicated by terms containing pattern variables, since failing to match a pattern term leads to the rejection of a message. A pattern term plays the implicit role of

a guard, so that, depending on the different ways of matching it, the protocol can have different continuations. Pattern variables are written with uppercase letters, e.g., $A, B, N_A, \ldots$.

- ***time variables***: a process cannot access the global clock, which implies that a time variable T of a reception or sending action $+(M@T)$ can never appear in M but can appear in the remaining part of the process. Also, given a receiving action $-(M_1@t_1)$ and a sending action $+(M_2@t_2)$ in a process of the form $P_1 \cdot -(M_1@t_1) \cdot P_2 \cdot +(M_2@t_2) \cdot P_3$, the assumption that timed actions are performed from left to right forces the constraint $t_1 \leq t_2$. Time variables are always written with a (subscripted) t, e.g., $t_1, t_1', t_2, t_2', \ldots$.

These conditions about variables are formalized by the function $wf : Proc \rightarrow Bool$ defined in Figure 4.1, for *well-formed* processes. The definition of wf uses an auxiliary function $shVar : Proc \rightarrow VarSet$, which is defined in Figure 4.2.

$$
\begin{aligned}
&wf(P \cdot +(M@T)) = wf(P) \\
&\quad \text{if } (\mathscr{V}ar(M) \cap \mathscr{V}ar(P)) \subseteq shVar(P) \wedge T \notin \mathscr{V}ar(M) \cup \mathscr{V}ar(P) \\
&wf(P \cdot -(M@T)) = wf(P) \\
&\quad \text{if } (\mathscr{V}ar(M) \cap \mathscr{V}ar(P)) \subseteq shVar(P) \wedge T \notin \mathscr{V}ar(M) \cup \mathscr{V}ar(P) \\
&wf(P \cdot (\text{if } T \text{ then } Q \text{ else } R)) = wf(P \cdot Q) \wedge wf(P \cdot R) \\
&\quad \text{if } P \neq nilP \text{ and } Q \neq nilP \text{ and } \mathscr{V}ar(T) \subseteq shVar(P) \\
&wf(P \cdot (Q \,?\, R)) = wf(P \cdot Q) \wedge wf(P \cdot R) \quad \text{if } Q \neq nilP \text{ or} R \neq nilP \\
&wf(P \cdot \ nilP) = wf(P) \\
&wf(nilP) = True.
\end{aligned}
$$

Figure 4.1: The well-formed function

$$
\begin{aligned}
&shVar(+(M@T) \ \cdot P) = \mathscr{V}ar(M) \cup shVar(P) \\
&shVar(-(M@T) \ \cdot P) = \mathscr{V}ar(M) \cup shVar(P) \\
&shVar((\text{if } T \text{ then } P \text{ else } Q) \ \cdot R) = \mathscr{V}ar(T) \cup (shVar(P) \cap shVar(Q)) \cup shVar(R) \\
&shVar((P \,?\, Q) \ \cdot R) = (shVar(P) \cap shVar(Q)) \cup shVar(R) \\
&shVar(nilP) = \emptyset
\end{aligned}
$$

Figure 4.2: The shared variables auxiliary function

Example 11. *Let us specify the Brands and Chaum protocol of Example 19, where variables are distinct between processes. A nonce is represented as $n(A_?, f)$, whereas a secret value is represented as $s(A_?, f)$. The identifier of each process is represented by a choice variable $A_?$. Recall that there is an arbitrary distance $d > 0$.*

$$
\begin{aligned}
(\mathit{Verifier}) : & -(\mathit{Commit}@t_1) \cdot \\
& +(n(V_?, f_1)@t_2) \cdot \\
& -((n(V_?, f_1) \oplus N_P)@t_3) \cdot \\
& \mathit{if}\ t_3 \dot{-} t_2 \leq 2 * d \\
& \mathit{then}\ -(S_P@t_4) \cdot \\
& \qquad \mathit{if}\ \mathit{open}(N_P, S_P, \mathit{Commit}) \\
& \qquad \mathit{then}\ -(\mathit{sign}(P, n(V_?, f_1); N_P \oplus n(V_?, f_1))@t_5)\ \mathit{else}\ \mathit{nilP} \\
& \mathit{else}\ \mathit{nilP} \\
(\mathit{Prover}) : & +(\mathit{commit}(n(P_?, f_1), s(P_?, f_2))@t_1) \cdot \\
& -(N_V@t_2) \cdot \\
& +((N_V \oplus n(P_?, f_1))@t_3) \cdot \\
& +(s(P_?, f_2)@t_4) \cdot \\
& +(\mathit{sign}(P_?, N_V; n(P_?, f_2) \oplus N_V)@t_5)
\end{aligned}
$$

4.3.2 Timed Intruder Model

The active Dolev-Yao intruder model is followed, which implies an intruder can intercept, forward, or create messages from received messages. However, intruders are *located*. Therefore, they cannot change the physics of the metric space, e.g., cannot send messages from a different location or intercept a message that it is not within range.

In our timed intruder model, we consider several located intruders, modeled by the distance function $d : \mathit{ProcId} \times \mathit{ProcId} \to \mathit{Real}$, each with a family of capabilities (concatenation, deconcatenation, encryption, decryption, etc.), and each capability may have arbitrarily many instances. The combined actions of two intruders requires time, i.e., their distance; but a single intruder can perform many actions in zero time. Adding time cost to single-intruder actions could be done with additional time constraints, but is outside the scope of this paper. Note that, unlike in the standard Dolev-Yao model, we cannot assume just one intruder, since the time required for a principal to communicate with a given intruder is an observable characteristic of that intruder. Thus, although the Mafia fraud and distance hijacking attacks considered in the experiments presented in this paper only require configurations with just one prover, one verifier and one intruder, the framework itself allows general participant configurations with multiple intruders.

Example 12. *In our timed process algebra, the family of capabilities associated to an intruder k are also described as processes. For instance, concatenating two received messages is represented by the process (where time variables t_1, t_2, t_3 are not actually*

used by the process)

$$(k.Conc):\ -(X@t_1)\ \cdot -(Y@t_2)\ \cdot +(X;Y@t_3)$$

and extracting one of them from a concatenation is described by the process

$$(k.Deconc):\ -(X;Y@t_1)\ \cdot +(X@t_2)$$

Roles of intruder capabilities include the identifier of the intruder, and it is possible to combine several intruder capabilities from the same or from different intruders. For example, we may say that the $+(X;Y@T)$ *of a process I1.Conc associated to an intruder* $I1$ *may be synchronized with the* $-(X;Y@T')$ *of a process I2.Deconc associated to an intruder* $I2$*. The metric space fixes* $T' = T + d(I1,I2)$*, where* $d(I1,I2) > 0$ *if* $I1 \neq I2$ *and* $d(I1,I2) = 0$ *if* $I1 = I2$*.*

A special forwarding *intruder capability, not considered in the standard Dolev-Yao model, has to be included in order to take into account the time travelled by a message from an honest participant to the intruder and later to another participant, probably an intruder again.*

$$(k.Forward):\ -(X@t_1)\ \cdot +(X@t_2)$$

4.3.3 Timed Process Semantics

A *state* of a protocol $\mathscr{P}$ consists of a set of (possibly partially executed) *labeled processes*, a set of terms in the network $\{Net\}$, and the global clock. That is, a state is a term of the form $\{LP_1 \,\&\, \cdots \,\&\, LP_n \mid \{Net\} \mid \bar{t}\}$. In the timed process algebra, the only time information available to a process is the variable T associated to input and output messages $M@T$. However, once these messages have been sent or received, we include them in the network Net with extra information. When a message $M@T$ is sent, we store $M\ @\ (A : t \to \emptyset)$ denoting that message M was sent by process A at the global time clock t, and propagate $T \mapsto t$ within the process A. When this message is received by an action $M'@T'$ of process B (honest participant or intruder) at the global clock time t', M is matched against M' modulo the cryptographic functions, $T' \mapsto t'$ is propagated within the process B, and $B : t'$ is added to the stored message, following the general pattern $M\ @\ (A : t \to (B_1 : t_1 \cdots B_n : t_n))$.

The rewrite theory $(\Sigma_{TPA_{\mathscr{P}}+State}, E_{TPA_{\mathscr{P}}}, R_{TPA_{\mathscr{P}}})$ characterizes the behavior of a protocol $\mathscr{P}$, where $\Sigma_{TPA_{\mathscr{P}}+State}$ extends $\Sigma_{TPA_{\mathscr{P}}}$, by adding state constructor symbols. We assume that a protocol run begins with an empty state, i.e., a

state with an empty set of labeled processes, an empty network, and at time zero. Therefore, the initial empty state is always of the form $\{\emptyset \mid \{\emptyset\} \mid 0.0\}$. Note that, in a specific run, all the distances are provided a priori according to the metric space and a chosen topology, whereas in a symbolic analysis, they will simply be variables, probably occurring within time constraints.

State changes are defined by a set $R_{TPA_{\mathscr{D}}}$ of *rewrite rules* given below. Each transition rule in $R_{TPA_{\mathscr{D}}}$ is labeled with a tuple (ro, i, j, a, n, t), where:

- ro is the role of the labeled process being executed in the transition.
- i denotes the instance of the same role being executed in the transition.
- j denotes the process' step number since its beginning.
- a is a ground term identifying the action that is being performed in the transition. It has different possible values: "$+m$" or "$-m$" if the message m was sent (and added to the network) or received, respectively; "m" if the message m was sent but did not increase the network, "?" if the transition performs an explicit non-deterministic choice, "T" if the transition performs an explicit deterministic choice, "*Time*" when the global clock is incremented, or "*New*" when a new process is added.
- n is a number that, if the action that is being executed is an explicit choice, indicates which branch has been chosen as the process continuation. In this case n takes the value of either 1 or 2. If the transition does not perform any explicit choice, then $n = 0$.
- t is the global clock at each transition step.

Note that in the transition rules $R_{TPA_{\mathscr{D}}}$ shown below, *Net* denotes the network, represented by a set of messages of the form M @ $(A : t \to (B_1 : t_1 \cdots B_n : t_n))$, P denotes the rest of the process being executed and PS denotes the rest of labeled processes of the state (which can be the empty set $\emptyset$).

- *Sending a message* is represented by the two transition rules below, depending on whether the message M is stored, (TPA++), or just discarded, (TPA+). In (TPA++), we store the sent message with its sending information, $(ro, i) : \bar{t}$, and add an empty set for those who will be receiving the message in the future $(M\sigma' @ (ro, i) : \bar{t} \to \emptyset)$.

$$\{(ro,i,j)\ (+M@t\cdot P)\ \&\ PS \mid \{Net\} \mid \bar{t}\}$$
$$\longrightarrow_{(ro,i,j,+(M\sigma'),0,\bar{t})}$$
$$\{(ro,i,j+1)\ P\sigma'\ \&\ PS \mid \{(M\sigma'@(ro,i):\bar{t}\to\emptyset),Net\} \mid \bar{t}\}$$
$$\textit{if } (M\sigma':(ro,i):\bar{t}\to\emptyset)\notin Net$$
where σ *is a ground substitution binding choice variables in* M
and $\sigma' = \sigma \uplus \{t \mapsto \bar{t}\}$ (TPA++)

$$\{(ro,i,j)\ (+M@t\cdot P)\ \&\ PS \mid \{Net\} \mid \bar{t}\}$$
$$\longrightarrow_{(ro,i,j,M\sigma',0,\bar{t})} \{(ro,i,j+1)\ P\sigma'\ \&\ PS \mid \{Net\} \mid \bar{t}\}$$
where σ *is a ground substitution binding choice variables in* M
and $\sigma' = \sigma \uplus \{t \mapsto \bar{t}\}$ (TPA+)

- *Receiving a message* is represented by the transition rule below. We add the reception information to the stored message, i.e., we replace $(M'@((ro',k):t'\to AS))$ by $(M'@((ro',k):t'\to(AS\uplus(ro,i):\bar{t}))$.

$$\{(ro,i,j)\ (-(M@t)\cdot P)\ \&\ PS \mid \{(M'@((ro',k):t'\to AS)),Net\} \mid \bar{t}\}$$
$$\longrightarrow_{(ro,i,j,-(M\sigma'),0,\bar{t})}$$
$$\{(ro,i,j+1)\ P\sigma'\ \&\ PS \mid \{(M'@((ro',k):t'\to(AS\uplus(ro,i):\bar{t})),Net\} \mid \bar{t}\}$$
$$\mathsf{IF}\ \exists\sigma : M' =_{E_{\mathscr{D}}} M\sigma, \bar{t} = t' + d((ro',k),(ro,i)), \sigma' = \sigma \uplus \{t \mapsto \hat{t}\}$$ (TPA-)

- An *explicit deterministic choice* is defined as follows. More specifically, the rule (TPAif1) describes the *then* case, i.e., if the constraint T is satisfied, then the process continues as P, whereas rule (TPAif2) describes the *else* case, that is, if the constraint T is *not* satisfied, the process continues as Q.

$$\{(ro,i,j)\ ((\textit{if } T \textit{ then } P \textit{ else } Q)\cdot R)\ \&\ PS \mid \{Net\} \mid \bar{t}\}$$
$$\longrightarrow_{(ro,i,j,T,1,\bar{t})} \{(ro,i,j+1)\ (P\cdot R)\&PS \mid \{Net\} \mid \bar{t}\}\mathsf{IF}\ T$$ (TPAif1)
$$\{(ro,i,j)\ ((\textit{if } T \textit{ then } P \textit{ else } Q)\cdot R)\ \&\ PS \mid \{Net\} \mid \bar{t}\}$$
$$\longrightarrow_{(ro,i,j,T,2,\bar{t})} \{(ro,i,j+1)\ (Q\cdot R)\&PS \mid \{Net\} \mid \bar{t}\}\mathsf{IF}\neg T$$ (TPAif2)

- An *explicit non-deterministic choice* is defined as follows. The process can continue either as P, denoted by rule (TPA?1), or as Q, denoted by rule (TPA?2).

$$\{(ro,i,j)\ ((P\ ?\ Q)\cdot R)\ \&\ PS \mid \{Net\} \mid \bar{t}\} \longrightarrow_{(ro,i,j,?,1,\bar{t})} \{(ro,i,j+1)\ (P\cdot R)\ \&\ PS \mid \{Net\} \mid \bar{t}\} \quad \text{(TPA?1)}$$

$$\{(ro,i,j)\ ((P\ ?\ Q)\cdot R)\ \&\ PS \mid \{Net\} \mid \bar{t}\} \longrightarrow_{(ro,i,j,?,2,\bar{t})} \{(ro,i,j+1)(Q\cdot R)\ \&\ PS \mid \{Net\} \mid \bar{t}\} \quad \text{(TPA?2)}$$

- *Global Time advancement* is represented by the transition rule below that increments the global clock enough to make one sent message arrive to its closest destination.

$$\{PS \mid \{Net\} \mid \bar{t}\} \longrightarrow_{(\perp,\perp,\perp,Time,0,\bar{t}+t')} \{PS \mid \{Net\} \mid \bar{t}+t'\}$$
$$\mathsf{IF}\ t' = mte(PS,Net,\bar{t}) \wedge t' \neq 0 \quad \text{(PTime)}$$

where the function *mte* is defined as follows:

$$mte(\emptyset,Net,\bar{t}) = \infty$$
$$mte(P\&PS,Net,\bar{t}) = min(mte(P,Net,\bar{t}),mte(PS,Net,\bar{t}))$$
$$mte((ro,i,j)\ nilP,Net,\bar{t}) = \infty$$
$$mte((ro,i,j)\ +(M@t)\cdot P,Net,\bar{t}) = 0$$
$$mte((ro,i,j)\ -(M@t)\cdot P,Net,\bar{t}) =$$
$$min\left(\left\{\begin{array}{c} d((ro,i),(ro',i')) \mid (M'@(ro',i') : t_0 \rightarrow AS) \in Net \\ \wedge \exists \sigma : M\sigma =_B M' \end{array}\right\}\right)$$
$$mte((ro,i,j)\ (if\ T\ then\ P\ else\ Q)\cdot R,Net,\bar{t}) = 0$$
$$mte((ro,i,j)\ P_1?P_2,Net,\bar{t}) = 0$$

Note that the function *mte* evaluates to 0 if some instantaneous action by the previous rules can be performed. Otherwise, *mte* computes the smallest non-zero time increment required for some already sent message (existing in the network) to be received by some process (by matching with such an existing message in the network).

Remark The timed process semantics assumes a metric space with a distance function $d : ProcId \times ProcId \rightarrow Real$ such that (i) $d(A,A) = 0$, (ii) $d(A,B) = d(B,A)$, and (iii) $d(A,B) \leq d(A,C) + d(C,B)$. For every message M @ $(A : t \rightarrow (B_1 : t_1 \cdots B_n : t_n))$ stored in the network *Net*, our semantics assumes that (iv) $t_i = t + d(A,B_i)$, $\forall 1 \leq i \leq n$. Furthermore, according to our wireless communication model, our semantics assumes (v) a *time sequence monotonicity* property, i.e., there is no other process C such that $d(A,C) \leq d(A,B_i)$ for some i, $1 \leq i \leq n$, and C is not included in the set of recipients of the message M. Also, for each class of attacks such as the Mafia fraud or the hijacking attack, (vi) some extra topology constraints may be necessary. However, in Section 4.4, timed processes are transformed into untimed processes with time constraints and the transformation takes care only of conditions (i), (ii), and (iv). For a fixed number of participants, all the instances of the triangle inequality (iii) as well as constraints (vi) should be added by the user. In the general case, conditions (iii), (v), and (vi) can be partially specified and fully checked on a successful trace.

- New processes can be added as follows.

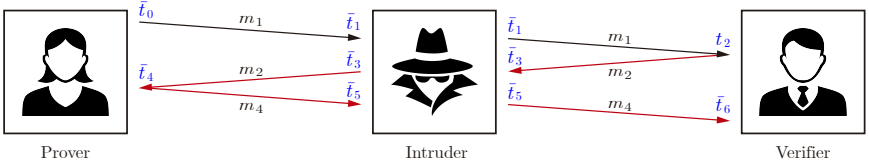
$$\left\{\begin{array}{l} \forall\ (ro)\ P_k \in P_{PA} \\ \{PS \mid \{Net\} \mid \bar{t}\} \\ \longrightarrow_{(ro,i+1,1,New,0,\bar{t})} \\ \{(ro, i+1, 1, x_?\sigma, y_?\sigma)\ P_k\sigma\rho_{ro,i+1}\ \&\ PS \mid \{Net\} \mid \bar{t}\} \\ \textit{where } \rho_{ro,i+1} \textit{ is a fresh substitution,} \\ \quad \sigma \textit{ is a ground substitution binding } x_? \textit{ and } y_?, \text{ and } i = id(PS, ro) \end{array}\right\} \quad \text{(TPA\&)}$$

The auxiliary function id counts the instances of a role

$$id(\emptyset, ro) = 0$$
$$id((ro', i, j)P \& PS, ro) = \begin{cases} max(id(PS, ro), i) & \textit{if } ro = ro' \\ id(PS, ro) & \textit{if } ro \neq ro' \end{cases}$$

where PS denotes a process configuration, P a process, and ro, ro' role names.

Therefore, the behavior of a timed protocol in the process algebra is defined by the set of transition rules $R_{TPA_{\mathscr{P}}} = \{\text{(TPA++)}, \text{(TPA+)}, \text{(PhyTime)}, \text{(TPA-)}, \text{(TPAif1)}, \text{(TPAif2)}, \text{(TPA?1)}, \text{(TPA?2)}\} \cup \text{(TPA\&)}$.



$$\{\emptyset|\{\emptyset\}|0.0\} \rightarrow_{p,0,1,New} \{(p,0,1):+(m_1@t_1)\cdots \mid \{\emptyset\} \mid \bar{t}_0 = 0.0\} \;\; m_1 = commit(n(p,f_1),s(p,f_2))$$

$$\rightarrow_{i,0,1,New} \left\{ \begin{array}{l} (p,0,1):+(m_1@t_1)\cdots \\ (i.F,0,1):-(X@t'_1)\cdot+(X@t'_2) \end{array} \right| \{\emptyset\} \mid \bar{t}_0\}$$

$$\rightarrow_{p,0,1,+(m_1)} \left\{ \begin{array}{l} (p,0,2):-(N_V@t_2)\cdots \\ (i.F,0,1):-(X@t'_1)\cdot+(X@t'_2) \end{array} \right| \{(m_1@(p,0):\bar{t}_0 \rightarrow \emptyset)\} \mid \bar{t}_0\}$$

$$\rightarrow_{Time} \left\{ \begin{array}{l} (p,0,2):-(N_V@t_2)\cdots \\ (i.F,0,1):-(X@t'_1)\cdot+(X@t'_2) \end{array} \right| \{(m_1@(p,0):\bar{t}_0 \rightarrow \emptyset)\} \mid \bar{t}_1 = 1.0\}$$

$$\rightarrow_{i,0,1,-(m_1)} \left\{ \begin{array}{l} (p,0,2):-(N_V@t_2)\cdots \\ (i.F,0,2):+(m_1@t'_2) \end{array} \right| \{(m_1@(p,0):\bar{t}_0 \rightarrow (i.F,0):\bar{t}_1)\} \mid \bar{t}_1\}$$

$$\rightarrow_{i,0,3,+(m_1)} \left\{ \begin{array}{l} (p,0,2):-(N_V@t_2)\cdots \\ (i.F,0,2):nilP \end{array} \right| \{(m_1@(i.F,0):\bar{t}_1 \rightarrow \emptyset\} \mid \bar{t}_1\}$$

$$\rightarrow_{v,0,1,New} \left\{ \begin{array}{l} (p,0,2):-(N_V@t_2)\cdots \\ (v,0,1):-(Commit@t''_1)\cdots \end{array} \right| \{(m_1@(i.F,0):\bar{t}_1 \rightarrow \emptyset)\} \mid \bar{t}_1\}$$

$$\rightarrow_{Time} \left\{ \begin{array}{l} (p,0,2):-(N_V@t_2)\cdots \\ (v,0,1):-(Commit@t''_1)\cdots \end{array} \right| \{(m_1@(i.F,0):\bar{t}_1 \rightarrow \emptyset)\} \mid \bar{t}_2 = 2.0\}$$

$$\rightarrow_{v,0,1,-(m_1)} \left\{ \begin{array}{l} (p,0,2):-(N_V@t_2)\cdots \\ (v,0,2):+(m_2@t''_2)\cdots \end{array} \right| \{(m_1@(i.F,0):\bar{t}_1 \rightarrow (v,0):\bar{t}_2)\} \mid \bar{t}_2\} \;\; m_2 = n(v,f_3)$$

$$\rightarrow_{v,0,2,+(m_2)} \left\{ \begin{array}{l} (p,0,2):-(N_V@t_2)\cdots \\ (v,0,3):-(m_3@t''_3)\cdots \end{array} \right| \{(m_2@(v,0):\bar{t}_2 \rightarrow \emptyset)\} \mid \bar{t}_2\} \;\; m_3 = (m_2 \oplus N_P)$$

$$\rightarrow_{i,1,1,New} \left\{ \begin{array}{l} (p,0,2):-(N_V@t_2)\cdots \\ (v,0,3):-(m_3@t''_3)\cdots \\ (i.F,1,1):-(X'@t'''_1)\cdot+(X'@t'''_2) \end{array} \right| \{(m_2@(v,0):\bar{t}_2 \rightarrow \emptyset)\} \mid \bar{t}_2\}$$

$$\rightarrow_{Time} \left\{ \begin{array}{l} (p,0,2):-(N_V@t_2)\cdots \\ (v,0,3):-(m_3@t''_3)\cdots \\ (i.F,1,1):-(X'@t'''_1)\cdot+(X'@t'''_2) \end{array} \right| \{(m_2@(v,0):\bar{t}_2 \rightarrow \emptyset)\} \mid \bar{t}_3 = 3.0\}$$

$$\rightarrow_{i,1,1,-(m_2)} \left\{ \begin{array}{l} (p,0,2):-(N_V@t_2)\cdots \\ (v,0,3):-(m_3@t''_3)\cdots \\ (i.F,1,2):+(m_2@t'''_2) \end{array} \right| \{(m_2@(v,0):\bar{t}_2 \rightarrow (i.F,1):\bar{t}_3)\} \mid \bar{t}_3\}$$

$$\rightarrow_{i,1,2,+(m_2)} \left\{ \begin{array}{l} (p,0,2):-(N_V@t_2)\cdots \\ (v,0,3):-(m_3@t''_3)\cdots \\ (i.F,1,2):nilP \end{array} \right| \{(m_2@(i.F,1):\bar{t}_3 \rightarrow \emptyset)\} \mid \bar{t}_3\}$$

$$\rightarrow_{Time} \left\{ \begin{array}{l} (p,0,2):-(N_V@t_2)\cdots \\ (v,0,3):-(m_3@t''_3)\cdots \end{array} \right| \{(m_2@(i.F,1):\bar{t}_3 \rightarrow \emptyset)\} \mid \bar{t}_4 = 4.0\}$$

$$\rightarrow_{p,0,2,-(m_2)} \left\{ \begin{array}{l} (p,0,3):+(m_4@t_3)\cdots \\ (v,0,3):-(m_3@t''_3)\cdots \end{array} \right| \{(m_2@(i.F,1):\bar{t}_3 \rightarrow (p,0):\bar{t}_4)\} \mid \bar{t}_4\} \;\; m_4 = (m_2 \oplus n(p,f_1))$$

$$\rightarrow_{p,0,3,+(m_4)} \left\{ \begin{array}{l} (p,0,4):+(m_5@t_4)\cdots \\ (v,0,3):-(m_3@t''_3)\cdots \end{array} \right| \{(m_4@(p,0):\bar{t}_4 \rightarrow \emptyset)\} \mid \bar{t}_4\} \;\; m_5 = s(p,f_2)$$

$$\rightarrow_{i,2,1,New} \left\{ \begin{array}{l} (v,0,3):-(m_3@t''_3)\cdots \\ (i.F,2,1):-(X''@t''''_1)\cdot+(X''@t''''_2) \end{array} \right| \{(m_4@(p,0):\bar{t}_4 \rightarrow \emptyset)\} \mid \bar{t}_4\}$$

$$\rightarrow_{Time} \left\{ \begin{array}{l} (v,0,3):-(m_3@t''_3)\cdots \\ (i.F,2,1):-(X''@t''''_1)\cdot+(X''@t''''_2) \end{array} \right| \{(m_4@(p,0):\bar{t}_4 \rightarrow \emptyset)\} \mid \bar{t}_5 = 5.0\}$$

$$\rightarrow_{i,3,1,-(m_4)} \left\{ \begin{array}{l} (v,0,3):-(m_3@t''_3)\cdots \\ (i.F,2,1):+(m_4@t''''_2) \end{array} \right| \{(m_4@(p,0):\bar{t}_4 \rightarrow (i.F,2):\bar{t}_5)\} \mid \bar{t}_5\}$$

$$\rightarrow_{i,3,2,+(m_4)} \left\{ \begin{array}{l} (v,0,3):-(m_3@t''_3)\cdots \\ (i.F,2,1):\; nil \end{array} \right| \{(m_4@(i.F,2):\bar{t}_5 \rightarrow \emptyset)\} \mid \bar{t}_5\}$$

$$\rightarrow_{Time} \{(v,0,3):-(m_3@t''_3)\cdots \mid \{(m_4@(i.F,2):\bar{t}_5 \rightarrow \emptyset)\} \mid \bar{t}_6 = 6.0\}$$

$$\rightarrow_{v,0,3,-(m_4)} \{(v,0,4):-(S_P@t_4)\cdots \mid \{(m_4@(i.F,2):\bar{t}_5 \rightarrow (v,0):\bar{t}_6)\} \mid \bar{t}_6\}$$

Figure 4.3: Brand and Chaum execution for a prover, an intruder, and a verifier

Example 13. *Continuing Example 21, a possible run of the protocol is represented in Figure 4.3 for a prover p, an intruder i, and a verifier v. A simpler, graphical representation of the same run is included at the top of the figure. There, the neighborhood distance is* $d = 1.0$*, the distance between the prover and the verifier is* $d(p,v) = 2.0$*, but the distance between the prover and the intruder as well as the distance between the verifier and the intruder are* $d(v,i) = d(p,i) = 1.0$*, i.e., the honest prover p is outside v's neighborhood,* $d(v,p) > d$*, where* $d(v,p) = d(v,i) + d(p,i)$*. Only the first part of the rapid message exchange sequence is represented and the forwarding action of the intruder is denoted by i.F.*

The prover sends the commitment $m_1 = commit(n(p,f_1), s(p,f_2))$ *at instant* $\bar{t}_0 = 0.0$ *and is received by the intruder at instant* $\bar{t}_1 = 1.0$*. The intruder forwards* m_1 *at instant* $\bar{t}_1$ *and is received by the verifier at instant* $\bar{t}_2 = 2.0$*. Then, the verifier sends* $m_2 = n(v,f_3)$ *at instant* $\bar{t}_2$*, which is received by the intruder at instant* $\bar{t}_3 = 3.0$*. The intruder forwards* m_2 *at instant* $\bar{t}_3$*, which is received by the prover at instant* $\bar{t}_4 = 4.0$*. Then, the prover sends* $m_4 = (m_2 \oplus n(p,f_1))$ *at instant* $\bar{t}_4$ *and is received by the intruder at instant* $\bar{t}_5 = 5.0$*. Finally, the intruder forwards* m_4 *at instant* $\bar{t}_5$ *and is received by the verifier at instant* $\bar{t}_6 = 6.0$*. Thus, the verifier sent* m_2 *at time* $\bar{t}_2 = 2.0$ *and received* m_4 *at time* $\bar{t}_6 = 6.0$*. But the protocol cannot complete the run, since* $\bar{t}_6 - \bar{t}_2 = 4.0 < 2 * d = 2.0$ *is unsatisfiable.*

Our time protocol semantics can already be implemented straightforwardly as a simulation tool. For instance, [83] describes distance bounding protocols using an authentication logic, which describes the evolution of the protocol, [97] provides a strand-based framework for distance bounding protocols based on simulation with time constraints, and [41] defines distance bounding protocol using some applied-pi calculus. Note, however, that, since the number of metric space configurations is infinite, model checking a protocol for a *concrete configuration* with a simulation tool is very limited, since it cannot prove the *absence* of an attack for *all* configurations. For this reason, we follow a symbolic approach that can explore *all* relevant configurations.

In the following section, we provide a sound and complete protocol transformation from our timed process algebra to the untimed process algebra of the Maude-NPA tool. In order to do this, we make use of an approach introduced by Nigam et al. [97] in which properties of time, which can include both those following from physics and those checked by principals, are represented by linear constraints on the reals. As a path is built, an SMT solver can be used to check that the constraints are satisfiable, as is done in [98].

4.4 Timed Process Algebra into Untimed Process Algebra with Time Variables and Timing Constraints

In this section, we consider a more general constraint satisfiability approach, where all possible (not only some) runs are symbolically analyzed. This provides both a trace-based insecure statement, i.e., a run leading to an insecure secrecy or authentication property is discovered given enough resources, and an unsatisfiability-based secure statement, i.e., there is no run leading to an insecure secrecy or authentication property due to time constraint unsatisfiability.

Example 14. *Consider again the run of the Brands-Chaum protocol given in Figure 4.3. All the terms of sort* Real*, written in blue color, are indeed variables that get an assignment during the run based on the distance function. Then, it is possible to obtain a symbolic trace from the run of Figure 4.3, where the following time constraints are accumulated:*

$$\bar{t}_1 = \bar{t}_0 + d((p,0),(i.F,0)),\ d((p,0),(i.F,0)) \geq 0$$

$$\bar{t}_2 = \bar{t}_1 + d((v,0),(i.F,0)),\ d((v,0),(i.F,0)) \geq 0$$

$$\bar{t}_3 = \bar{t}_2 + d((v,0),(i.F,1)),\ d((v,0),(i.F,1)) \geq 0$$

$$\bar{t}_4 = \bar{t}_3 + d((p,0),(i.F,1)),\ d((p,0),(i.F,1)) \geq 0$$

$$\bar{t}_5 = \bar{t}_4 + d((p,0),(i.F,2)),\ d((p,0),(i.F,2)) \geq 0$$

$$\bar{t}_6 = \bar{t}_5 + d((v,0),(i.F,2)),\ d((v,0),(i.F,2)) \geq 0$$

Note that these constraints are unsatisfiable when combined with (i) the assumption $d > 0$, (ii) the verifier check $\bar{t}_6 - \bar{t}_2 \leq 2 * d$, (iii) the assumption that the honest prover is outside the verifier's neighborhood, $d((p,0),(v,0)) > d$, (iv) the triangular inequality from the metric space, $d((p,0),(v,0)) \leq d((p,0),(i.F,0)) + d($ $(i.F,0),(v,0))$, and (v) the assumption that there is only one intruder $d((i.F,0),$ $(i.F,1)) = 0$ and $d((i.F,0),(i.F,2)) = 0$.

As explained previously in the remark, there are some implicit conditions based on the *mte* function to calculate the time increment to the closest destination of a message. However, the *mte* function disappears in the untimed process algebra and those implicit conditions are incorporated into the symbolic run. In the following, we define a transformation of the timed process algebra

by (i) removing the global clock; (ii) adding the time data into untimed messages of a process algebra without time (as done in [97]); and (iii) adding linear arithmetic conditions over the reals for the time constraints (as is done in [98]). The soundness and completeness proof of the transformation is included in the full version of the paper, available at `https://arxiv.org/abs/2010.13707`.

Since all the relevant time information is actually stored in messages of the form M @ $(A : t \to (B_1 : t_1 \cdots B_n : t_n))$ and controlled by the transition rules (TPA++),(TPA+), and (TPA-), the mapping *tpa2pa* of Definition 11 below transforms each message $M@t$ of a timed process into a message M @ $(A : t_? \to AS_?)$ of an untimed process. That is, we use a timed choice variable $t_?$ for the sending time and a variable $AS_?$ for the reception information $(B_1 : t'_1 \cdots B_n : t'_n)$ associated to the sent message. Since choice variables are replaced by specific values, both $t_?$ and $AS_?$ will be replaced by the appropriate values that make the execution and all its time constraints possible. Note that these two choice variables will be replaced by logical variables during the symbolic execution.

Definition 10 (Adding Time Variables and Time Constraints to Untimed Processes)**.** *The mapping tpa2pa from timed processes into untimed processes and its auxiliary mapping tpa2pa* are defined as follows:*

$$
\begin{array}{l}
tpa2pa(\emptyset) = \emptyset \\
tpa2pa((ro,i,j)\,P \;\&\; PS) = (ro,i,j)\,tpa2pa^*(P,ro,i) \;\&\; tpa2pa(PS) \\
\\
tpa2pa^*(nilP, ro, i) = nilP \\
tpa2pa^*(\ +(M@t)\ .\ P, ro, i) = +(M@((ro,i) : t_? \to AS_?))\ .\ tpa2pa^*(P\gamma, ro, i) \\
\quad where\ \gamma = \{t \mapsto t_?\} \\
tpa2pa^*(\ -(M@t)\ .\ P, ro, i) = \\
\ -(M@((ro',i') : t' \to ((ro,i) : t) \uplus AS))\ . \\
\ if\ t = t' + d((ro,i),(ro',i')) \wedge d((ro,i),(ro',i')) \geq 0\ then\ tpa2pa^*(P, ro, i)\ else\ nilP \\
tpa2pa^*(\ (if\ C\ then\ P\ else\ Q)\ .\ R, ro, i, x, y) \\
\ = \ (if\ C\ then\ tpa2pa^*(P, ro, i, x, y)\ else\ tpa2pa^*(Q, ro, i, x, y))\ .\ tpa2pa^*(R, ro, i, x, y) \\
tpa2pa^*(\ (P\ ?\ Q)\ .\ R, ro, i, x, y) \\
\ = \ (tpa2pa^*(P, ro, i, x, y)\ ?\ tpa2pa^*(Q, ro, i, x, y))\ .\ tpa2pa^*(R, ro, i, x, y)
\end{array}
$$

where $t_?$ and $AS_?$ are choice variables different for each one of the sending actions, ro', i', t', d, AS are pattern variables different for each one of the receiving actions, P, Q, and R are processes, M is a message, and C is a constraint.

Example 15. *The timed processes of Example 21 are transformed into the following untimed processes. We remove the "else nilP" branches for clarity.*

$$
\begin{aligned}
(\mathit{Verifier}) : & -(\mathit{Commit} \ @ \ A_1 : t_1' \rightarrow V_? : t_1 \uplus AS_1) \cdot \\
& \mathit{if}\ t_1 = t_1' + d(A_1, V_?) \wedge d(A_1, V_?) \geq 0\ \mathit{then} \\
& +(n(V_?, f_1) \ @ \ V_? : t_{2?} \rightarrow AS_{2?}) \cdot \\
& -((n(V_?, f_1) \oplus N_P) \ @ \ A_3 : t_3' \rightarrow V_? : t_3 \uplus AS_3) \cdot \\
& \mathit{if}\ t_3 = t_3' + d(A_3, V_?) \wedge d(A_3, V_?) \geq 0\ \mathit{then} \\
& \mathit{if}\ t_3 \dot{-} t_{2?} \leq 2 * d\ \mathit{then} \\
& -(S_P \ @ \ A_4 : t_4' \rightarrow V_? : t_4 \uplus AS_4) \cdot \\
& \mathit{if}\ t_4 = t_4' + d(A_4, V_?) \wedge d(A_4, V_?) \geq 0\ \mathit{then} \\
& \mathit{if}\ \mathit{open}(N_P, S_P, \mathit{Commit})\ \mathit{then} \\
& -(\mathit{sign}(P, n(V_?, f_1); N_P \oplus n(V_?, f_1)) \ @ \ A_5 : t_5' \rightarrow V_? : t_5 \uplus AS_5) \\
& \mathit{if}\ t_5 = t_5' + d(A_5, V_?) \wedge d(A_5, V_?) \geq 0 \\
(\mathit{Prover}) : & +(\mathit{commit}(n(P_?, f_1), s(P_?, f_2)) @ P_? : t_{1?} \rightarrow AS_{1?}) \cdot \\
& -(V; N_V @ \ A_2 : t_2' \rightarrow V_? : t_2 \uplus AS_2) \cdot \\
& \mathit{if}\ t_2 = t_2' + d(A_2, P_?) \wedge d(A_2, P_?) \geq 0\ \mathit{then} \\
& +((N_V \oplus n(P_?, f_1)) @ P_? : t_{3?} \rightarrow AS_{3?}) \cdot \\
& +(s(P_?, f_2) @ P_? : t_{4?} \rightarrow AS_{4?}) \cdot \\
& +(\mathit{sign}(P_?, N_V; n(P_?, f_2) \oplus N_V) @ P_? : t_{5?} \rightarrow AS_{5?}))
\end{aligned}
$$

Example 16. *The timed processes of Example 23 for the intruder are transformed into the following untimed processes. Note that we use the intruder identifier I associated to each role instead of a choice variable $I_?$.*

$$
\begin{aligned}
(\mathit{I.Conc}) : & -(X @ \ A_1 : t_1 \rightarrow I : t_1' \uplus AS_1) \cdot \\
& \mathit{if}\ t_1' = t_1 + d(A_1, I) \wedge d(A_1, I) \geq 0\ \mathit{then} \\
& -(Y @ \ A_2 : t_2 \rightarrow I : t_2' \uplus AS_2) \cdot \\
& \mathit{if}\ t_2' = t_2 + d(A_2, I) \wedge d(A_2, I) \geq 0\ \mathit{then} \\
& +(X; Y @ I : t_{3?} \rightarrow AS_?)
\end{aligned}
$$

$$
\begin{aligned}
(\mathit{I.Deconc}) : & -(X; Y @ \ A_1 : t_1 \rightarrow I : t_1' \uplus AS_1) \cdot \\
& \mathit{if}\ t_1' = t_1 + d(A_1, I) \wedge d(A_1, I) \geq 0\ \mathit{then} \\
& +(X @ I : t_{2?} \rightarrow AS_?) \\
(\mathit{I.Forward}) : & -(X @ \ A_1 : t_1 \rightarrow I : t_1' \uplus AS_1) \cdot \\
& \mathit{if}\ t_1' = t_1 + d(A_1, I) \wedge d(A_1, I) \geq 0\ \mathit{then} \\
& +(X @ I : t_{2?} \rightarrow AS_?)
\end{aligned}
$$

Once a timed process is transformed into an untimed process with time variables and time constraints using the notation of Maude-NPA, we rely on

both a soundness and completeness proof from the Maude-NPA process notation into Maude-NPA forward rewriting semantics and on a soundness and completeness proof from Maude-NPA forward rewriting semantics into Maude-NPA backwards symbolic semantics, see [114, 115]. Since the Maude-NPA backwards symbolic semantics already considers constraints in a very general sense [57], we only need to perform the additional satisfiability check for linear arithmetic over the reals.

4.5 Timed Process Algebra into Strands in Maude-NPA

This section is provided to help in understanding the experimental output. Although Maude-NPA accepts protocol specifications in either the process algebra language or the strand space language, it still gives outputs only in the strand space notation. Thus, in order to make our experimental output easier to understand, we describe the translation from timed process into strands with time variables and time constraints. This translation is also sound and complete, as it imitates the transformation of Section 4.4 and the transformation of [114, 115].

Strands [62] are used in Maude-NPA to represent both the actions of honest principals (with a strand specified for each protocol role) and those of an intruder (with a strand for each action an intruder is able to perform on messages). In Maude-NPA, strands evolve over time. The symbol | is used to divide past and future. That is, given a strand [$msg_1^{\pm}$, ..., $msg_i^{\pm}$ | $msg_{i+1}^{\pm}$, ..., $msg_k^{\pm}$], messages $msg_1^{\pm},\ldots,msg_i^{\pm}$ are the *past messages*, and messages $msg_{i+1}^{\pm},\ldots,msg_k^{\pm}$ are the *future messages* ($msg_{i+1}^{\pm}$ is the immediate future message). Constraints can be also inserted into strands. A strand [$msg_1^{\pm},\ldots,msg_k^{\pm}$] is shorthand for [*nil* | $msg_1^{\pm},\ldots,msg_k^{\pm}$,*nil*]. An *initial state* is a state where the bar is at the beginning for all strands in the state, and the network has no possible intruder fact of the form $m \in \mathscr{I}$. A *final state* is a state where the bar is at the end for all strands in the state and there is no negative intruder fact of the form $m \notin \mathscr{I}$.

In the following example, we illustrate how the timed process algebra can be transformed into strands specifications of Maude-NPA.

Example 17. *The timed processes of Example 21 are transformed into the following strand specification.*

$$
\begin{array}{rl}
(\mathit{Verifier}): & [-(\mathit{Commit} \ @ \ A_1 : t_1' \to V : t_1 \uplus AS_1), \\
& (t_1 = t_1' + d(A_1,V) \wedge d(A_1,V) \geq 0), \\
& +(n(V,f_1) \ @ \ V : t_2 \to AS_2), \\
& -((n(V,f_1) \oplus N_P) \ @ \ A_3 : t_3' \to V : t_3 \uplus AS_3), \\
& (t_3 = t_3' + d(A_3,V) \wedge d(A_3,V) \geq 0), \\
& (t_3 \dot{-} t_2 \leq 2 * d), \\
& -(S_P \ @ \ A_4 : t_4' \to V : t_4 \uplus AS_4), \\
& (t_4 = t_4' + d(A_4,V) \wedge d(A_4,V) \geq 0), \\
& \mathit{open}(N_P, S_P, \mathit{Commit}), \\
& -(\mathit{sign}(P, n(V,f_1); N_P \oplus n(V,f_1)) @ \ A_5 : t_5' \to V : t_5 \uplus AS_5), \\
& (t_5 = t_5' + d(A_5,V) \wedge d(A_5,V) \geq 0)] \\
(\mathit{Prover}): & [+(\mathit{commit}(n(P,f_1), s(P,f_2)) @ P : t_1 \to AS_1), \\
& -(N_V @ \ A_2 : t_2' \to V : t_2 \uplus AS_2), \\
& (t_2 = t_2' + d(A_2,P) \wedge d(A_2,P) \geq 0), \\
& +((N_V \oplus n(P,f_1)) @ P : t_3 \to AS_3), \\
& +(s(P,f_2) @ P : t_4 \to AS_4), \\
& +(\mathit{sign}(P, N_V; n(P,f_2) \oplus N_V) @ P : t_5 \to AS_5)]
\end{array}
$$

We specify the desired security properties in terms of *attack patterns* including logical variables, which describe the insecure states that Maude-NPA is trying to prove unreachable. Specifically, the tool attempts to find a *backwards narrowing sequence* path from the attack pattern to an initial state until it can no longer form any backwards narrowing steps, at which point it terminates. If it has not found an initial state, the attack pattern is judged *unreachable*.

The following example shows how a classic mafia fraud attack for the Brands-Chaum protocol can be encoded in Maude-NPA's strand notation.

Example 18. *Following the strand specification of the Brands-Chaum protocol given in Example 31, the mafia attack of Example 19 is given as the following attack pattern. Note that Maude-NPA uses symbol* `===` *for equality on the reals,* `+=+` *for addition on the reals,* `*=*` *for multiplication on the reals, and* `-=-` *for subtraction on the reals. Also, we consider one prover* `p`*, one verifier* `v`*, and one intruder* `i` *at fixed locations. Extra time constraints are included in an* `smt` *section, where a triangular inequality has been added. The mafia fraud attack is secure for Brands-Chaum and no initial state is found in the backwards search.*

```
eq ATTACK-STATE(1) ---  Mafia fraud
   = :: r :: --- Verifier
     [ nil, -(commit(n(p,r1),s(p,r2))  @ i : t1 -> v : t2),
            ((t2 === t1 +=+ d(i,v)) and d(i,v) >= 0/1),
```

```
        +(n(v,r)                  @ v : t2 -> i : t2''),
        -(n(v,r) * n(p,r1)        @ i : t3 -> v : t4),
        (t3 >= t2 and (t4 === t3 +=+ d(i,v)) and d(i,v) >= 0/1),
        ((t4 -=- t2) <= (2/1 *=* d)) | nil ] &
:: r1,r2 :: --- Prover
[ nil,  +(commit(n(p,r1),s(p,r2)) @ p : t1' -> i : t1''),
        -(n(v,r) @ i : t2'' -> p : t3'),
        ((t3' === t2'' +=+ d(i,p)) and d(i,p) >= 0/1),
        +(n(v,r) * n(p,r1)        @ p : t3' -> i : t3'') | nil ]
|| smt(d(v,p) > 0/1 and d(i,p) > 0/1 and d(i,v) > 0/1 and d(v,i) <= d and
        (d(v,i) +=+ d(p,i)) >= d(v,p) and d(v,p) > d)
|| nil || nil || nil [nonexec] .
```

4.6 Experiments

As a feasibility study, we have encoded several distance bounding protocols in Maude-NPA. It was necessary to slightly alter the Maude-NPA tool by (i) including minor modifications to the state space reduction techniques to allow for timed messages; (ii) the introduction of the sort Real and its associated operations; and (iii) the connection of Maude-NPA to a *Satisfiability Modulo Theories (SMT)* solver[1] (see [96] for details on SMT). The specifications, outputs, and the modified version of Maude-NPA are available at `http://personales.upv.es/sanesro/indocrypt2020/`.

Although the timed model allows an unbounded number of principals, the attack patterns used to specify insecure goal states allow us to limit the number of principals in a natural way. In this case we specified one verifier, one prover, and one attacker, but allowed an unbounded number of sessions.

In Table 4.1 below we present the results for the different distance-bounding protocols that we have analyzed. Two attacks have been analyzed for each protocol: a *mafia fraud* attack (i.e., an attacker tries to convince the verifier that an honest prover is closer to him than he really is), and a *distance hijacking* attack (i.e., a dishonest prover located far away succeeds in convincing a verifier that they are actually close, and he may only exploit the presence of honest participants in the neighborhood to achieve his goal). Symbol ✓ means the property is satisfied and × means an attack was found. The columns labelled *tm*(*sec*) give the times in seconds that it took for a search to complete. Finally, the column labeled PreProc gives the time it takes Maude-NPA to

[1] Several SMT solvers are publicly available, but the programming language Maude [35] currently supports CVC4 [1] and Yices [5].

Protocol	PreProc (sec)	Mafia	tm (sec)	Hijacking	tm (sec)
Brands and Chaum [27]	3.0	✓	4.3	×	11.4
Meadows et al ($n_V \oplus n_P$,P) [83]	3.7	✓	1.3	✓	22.5
Meadows et al (n_V,$n_P \oplus P$) [83]	3.5	✓	1.1	×	1.5
Hancke and Kuhn [67]	1.2	✓	12.5	✓	0.7
MAD [30]	5.1	✓	110.5	×	318.8
Swiss-Knife [72]	3.1	✓	4.8	✓	24.5
Munilla et al. [94]	1.7	✓	107.1	✓	4.5
CRCS [102]	3.0	✓	450.1	×	68.6
TREAD [14]	2.4	✓	4.7	×	4.2

Table 4.1: Experiments performed for different distance-bounding protocols

perform some preprocessing on the specification that eliminates searches for some provably unreachable state. This only needs to be done once, after which the results can be used for any query, so it is displayed separately.

We note that, since our semantics is defined over arbitrary metric spaces, not just Euclidean space, it is also necessary to verify that an attack returned by the tool is realizable over Euclidean space. We note that the Mafia and hijacking attacks returned by Maude-NPA in these experiments are all realizable on a straight line, and hence are realizable over n-dimensional Euclidean space for any n. In general, this realizability check can be done via a final step in which the constraints with the Euclidean metric substituted for distance is checked via an SMT solver that supports checking quadratic constraints over the reals, such as Yices [5], Z3 [8], or Mathematica [4]. Although this feature is not yet implemented in Maude-NPA, we have begun experimenting with these solvers.

4.7 Conclusions

We have developed a timed model for protocol analysis based on timing constraints, and provided a prototype extension of Maude-NPA handling protocols with time by taking advantage of Maude's support of SMT solvers, as was done by Nigam et al. in [98], and Maude-NPA's support of constraint handling. We also performed some initial analyses to test the feasibility of the approach. This approach should be applicable to other tools that support constraint handling.

There are several ways this work can be extended. One is to extend the ability of the tool to reason about a larger numbers or principals, in particular

an unbounded number of principals. This includes an unbounded number of attackers; since each attacker must have its own location, we cannot assume a single attacker as in Dolev-Yao. Our specification and query language, and its semantics, supports reasoning about an unbounded number of principals, so this is a question of developing means of telling when a principal or state is redundant and developing state space reduction techniques based on this.

Another important extension is to protocols that require the full Euclidean space model, in particular those in which location needs to be explicitly included in the constraints. This includes for example protocols used for localization. For this, we have begun experimenting with SMT solvers that support solving quadratic constraints over the reals.

Looking further afield, we consider adding different types of timing models. In the timing model used in this paper, time is synonymous with distance. But we may also be interested including other ways in which time is advanced, e.g. the amount of time a principal takes to perform internal processing tasks. In our model, the method in which timing is advanced is specified by the *mte* function, which is in turn used to generate constraints on which messages can be ordered. Thus changing the way in which timing is advanced can be accomplished by modifying the *mte* function. Thus, potential future research includes design of *generic mte* functions together with rules on their instantiation that guarantee soundness and completeness

Finally, there is also no reason for us to limit ourselves to time and location. This approach should be applicable to other quantitative properties as well. For example, the inclusion of cost and utility would allow us to tackle new classes of problems not usually addressed by cryptographic protocol analysis tools, such as performance analyses (e.g., resistance against denial of service attacks), or even analysis of game-theoretic properties of protocols, thus opening up a whole new set of problems to explore.

CHAPTER 5

Protocol Analysis with Time and Space

Damián Aparicio-Sánchez[1], Santiago Escobar[1], Catherine Meadows[2]
José Meseguer[3], Julia Sapiña[1]

[1] VRAIN, Universitat Politècnica de València, Spain
{daapsnc,sescobar,jsapina}@upv.es

[2] Naval Research Laboratory, Washington DC, USA
meadows@itd.nrl.navy.mil

[3] University of Illinois at Urbana-Champaign, USA
meseguer@illinois.edu

Abstract We present a formal framework for the analysis of cryptographic protocols that make use of time and space in their execution. In a previous work we provided a timed process algebra syntax and a timed transition semantics. The timed process algebra only made message sending-and-reception times available to processes whereas the timed transition semantics modelled the actual time interactions between processes. In this paper we extend the previous process algebra syntax to make spatial location information also available to processes and provide a transition semantics that takes account of fundamental properties of both time and

This work has been partially supported by the EU (FEDER) and the Spanish MCIU under grant RTI2018-094403-B-C32, by Generalitat Valenciana under grant PROMETEO/2019/098, by EIG-CONCERT-JAPAN under grant PCI2020-120708-2, and by NRL under contract number N00173-17-1-G002. Julia Sapiña has been supported by the Generalitat Valenciana APOSTD/2019/127 grant.

space. This time and space protocol framework can be implemented either as a simulation tool or as a symbolic analysis tool in which time and space information are not represented by specific values but by logical variables, and in which the properties of time and space are reasoned about in terms of constraints on those time and space logical variables. All these time and space constraints are carried along the symbolic execution of the protocol and their satisfiability can be evaluated as the analysis proceeds, so attacks that violate the laws of physics can be discarded as impossible. We demonstrate the feasibility of our approach by using the Maude-NPA protocol analyzer together with an SMT solver that is used to evaluate the satisfiability of timing and location constraints. We provide a sound and complete protocol transformation from our time and space process algebra to the Maude-NPA syntax and semantics, and we prove its soundness and completeness. We analyze two protocols using time and space constraints.

5.1 Introduction

The laws of physics are an important aspect of many cryptographic protocols, and there has been increasing interest in the formal analysis of protocols that require them to function properly. Model checking of protocols that use time and space can be done using either an explicit model with time and space information or by using an untimed model and showing it is sound and complete with respect to a time and space model. The former is more intuitive for the user, but the latter is often chosen because not all cryptographic protocol analysis tools support reasoning about either time or space.

In previous Chapter 4 as well as [13], we provided a framework for analyzing protocols involving time. We combined the advantages of both approaches: an explicit timed specification language was developed with a timed syntax and semantics, and was automatically and faithfully translated into an existing untimed language. We applied this approach to the Maude-NPA tool by taking advantage of its built-in support for constraints and analyzed Mafia fraud and distance hijacking attacks on a suite of distance-bounding protocols.

We celebrate Joshua Guttman with a paper on a tool and approach based on one of his most important contributions to security: the strand space model introduced by Thayer, Herzog, and Guttman in [62]. In this graph-based model both protocol roles and adversarial actions are represented by strands, which are lists of terms sent and received by a principal in the order that they occur. A protocol execution (or *bundle*) is constructed by matching sent terms

with received terms in different strands. We have used strand spaces as the basis of Maude-NPA syntax and semantics [53], and have found that they allow us to represent them in a very natural way. We have also found [115], that this syntax and semantics can be naturally extended to a process algebra syntax and semantics. Moreover, strand spaces are very amenable to extension via adding constraints to the strand space implementation. In particular, we have found this approach useful for adding features such as state space reduction [56, 57], deterministic and nondeterministic choice [115], timed protocols [13], and now, protocols that use both space and time.

In previous Chapter 4 as well as [13], we assumed a metric space with a distance function such that (i) $d(A,A) = 0$, (ii) $d(A,B) = d(B,A)$, and (iii) $d(A,B) \leq d(A,C) + d(C,B)$. In this paper, we actually compute the real distances according to a three-dimensional space: $d(A,B)^2 = (A_x - B_x)^2 + (A_y - B_y)^2 + (A_z - B_z)^2$. We extend the previous process algebra syntax to make spatial location information also available to processes and provide a transition semantics that takes account of fundamental properties of both time and location. The new time and space protocol framework clearly subsumes and extends the previous time framework.

As it already happened in [13], this time and space protocol framework can be implemented either as a simulation tool or as a symbolic analysis tool in which time and space information is not represented by specific values but by logical variables, and in which the properties of time and space are represented as constraints on those time and space logical variables. All these time and space constraints are carried along the symbolic execution of the protocol and their satisfiability can be evaluated as the analysis proceeds, so attacks that violate the laws of physics can be discarded as impossible. We realize the time and space semantics by translating it into the semantics of the Maude-NPA protocol analysis tool, in which time and space are expressed as constraints. The constraints generated during the Maude-NPA search are then checked using an embedded SMT solver.

We believe that this approach can be applied to other tools that support constraint handling as well. Many tools support constraint handling, e.g., Maude-NPA [57] and Tamarin [85]. The laws of physics can be naturally added to a process algebra. Many tools support processes, e.g., Maude-NPA [115] and AKISS [41].

The rest of this paper is organized as follows. In Section 5.2, we present our two running examples: the Brands-Chaum protocol and a secure localization protocol. In Section 5.3, we present the time and space process algebra with

its intended semantics. In Section 5.4, we present a sound and complete protocol transformation from our time and space process algebra to an untimed process algebra with constraints. In Section 5.5, we present a second transformation from the untimed process algebra into Maude-NPA strand notation. We conclude in Section 5.6.

5.1.1 Related work

There are a number of security protocols that make use of time. In general, there are two types: those that make use of assumptions about time, most often assuming some sort of loose synchronization, and those that guarantee these assumptions. The first kind includes protocols such as Kerberos [95], which uses timestamps to defend against replay attacks, the TESLA protocol [101], which relies on loose synchronization to amortize digital signatures, and blockchain protocols, which use timestamps to order blocks in the chain. The other kind provides guarantees based on physical properties of time: for example, distance bounding, which guarantees that a prover is within a certain distance of a verifier, and secure time synchronization, which guarantees that the clocks of two different nodes are synchronized within a certain margin of error. We refer the reader to [13] for a discussion on timed protocols.

For location-based protocols, the concepts of *physical proximity*, *secure localization*, *secure neighbor discovery* and *secure distance measurement* are used quite often. In [19,20,107], Basin et al. define formal models for reasoning about physical properties of security protocols, including timing and location, using Isabelle/HOL and a technique similar to Paulson's inductive approach [100]. The notion of *secure distance measurement* has been studied in [29, 31, 32,79]. In [79], Message Time Of Arrival Codes (MTACs) are developed, a new class of cryptographic primitives that allow receivers to verify if an adversary has manipulated the message arrival time in a similar way to how Message Authentication Codes protect message integrity.

5.2 Two Time and Space Protocols

Example 19. *The Brands-Chaum protocol [27] specifies communication between a verifier V and a prover P. P needs to authenticate itself to V, and also needs to prove that it is within a distance "d" of it. A typical interaction between the prover and the verifier is as follows, where N_A denotes a nonce generated by A, S_A denotes a secret generated by A, $X;Y$ denotes concatenation of two messages X and Y, commit(N,S)*

denotes commitment of secret S with a nonce N, open(N,S,C) denotes opening a commitment C using the nonce N and checking whether it carries the secret S, $\oplus$ is the exclusive-or operator, and sign(A,M) denotes A signing message M.

$P \to V : commit(N_P, S_P)$
//The prover sends his name and a commitment

$V \to P : N_V$
//The verifier sends a nonce and records the time when this message was sent

$P \to V : N_P \oplus N_V$
//The verifier checks the answer message arrives within two times a fixed distance

$P \to V : S_P$
//The prover sends the committed secret and the verifier opens the commitment

$P \to V : sign_P(N_V; N_P \oplus N_V)$
//The prover signs the two rapid exchange messages

In previous Chapter 4 as well as [13], , we already considered this Brands-Chaum protocol. We assumed the participants were located at an arbitrary given topology (participants do not move from their assigned locations) with distance constraints, where time and distance are equivalent for simplification and are represented by a real number. We assumed a metric space with a distance function $d : \mathscr{A} \times \mathscr{A} \to Real$ from a set $\mathscr{A}$ of participants such that $d(A,B) \geq 0$, $d(A,A) = 0$, $d(A,B) = d(B,A)$, and $d(A,B) \leq d(A,C) + d(C,B)$.

In this paper, we assume coordinates P_x, P_y, P_z for each participant P and the distance function $d : \mathscr{A} \times \mathscr{A} \to Real$ calculated from the positions of the participants. From now on, we will use the following notation in order to improve readability: $\lfloor d(A,B) \rfloor$ that provides the set of constraints associated to a symbolic distance between participants A and B and $d((x,y,z),(x',y',z'))$ that calculates the actual distance between participants A and B from their given concrete coordinates:

$$\lfloor d(A,B) \rfloor := \ (d(A,B) \geq 0 \wedge d(A,B)^2 = (A_x - B_x)^2 + (A_y - B_y)^2 + (A_z - B_z)^2)$$

$$d((x,y,z),(x',y',z')) := \ \sqrt{(x-x')^2 + (y-y')^2 + (z-z')^2}$$

The previous informal Alice&Bob notation was naturally extended to include time in Chapter 4 and we further extend it here to include both time and location. First, we add the time when a message was sent or received as a subindex $P_{t_1} \to V_{t_2}$. Second, the sending and receiving times of a message differ by the distance between them just by adding the location constraints $\lfloor d(A,B) \rfloor$. Third, the distance bounding constraint of the verifier is represented as an arbitrary distance d. Time and space constraints are written

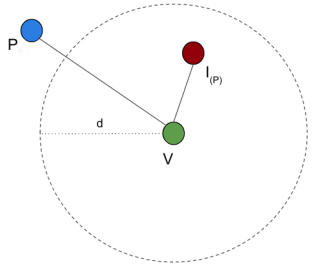


Figure 5.1: Mafia Attack

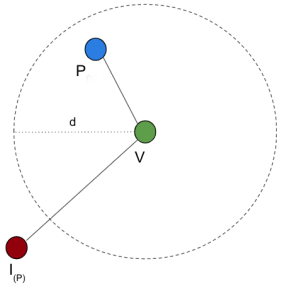


Figure 5.2: Hijacking Attack

using quantifier-free formulas in real arithmetic. For convenience, we allow both $2*x = x+x$ and the monus function $x \dot{-} y = \mathit{if}\ y < x\ \mathit{then}\ x-y\ \mathit{else}\ 0$ as definitional extensions.

In the following time and space sequence of actions, a vertical bar differentiates between the process and corresponding constraints associated to the metric space. We remove the constraint $\mathit{open}(N_P, S_P, \mathit{commit}(N_P, S_P))$ for simplification. The following action sequence differs from Chapter 4 only on the terms $\lfloor d(P,V) \rfloor$.

$$
\begin{array}{ll}
P_{t_1} \to V_{t'_1} : \mathit{commit}(N_P, S_P) & \mid t'_1 = t_1 + d(P,V) \wedge \lfloor d(P,V) \rfloor \\
V_{t_2} \to P_{t'_2} : N_V & \mid t'_2 = t_2 + d(P,V) \wedge t_2 \geq t'_1 \wedge \lfloor d(P,V) \rfloor \\
P_{t_3} \to V_{t'_3} : N_P \oplus N_V & \mid t'_3 = t_3 + d(P,V) \wedge t_3 \geq t'_2 \wedge \lfloor d(P,V) \rfloor \\
\quad V : t'_3 \dot{-} t_2 \leq 2 * d & \\
P_{t_4} \to V_{t'_4} : S_P & \mid t'_4 = t_4 + d(P,V) \wedge t_4 \geq t'_3 \wedge \lfloor d(P,V) \rfloor \\
P_{t_5} \to V_{t'_5} : \mathit{sign}_P(N_V; N_P \oplus N_V) & \mid t'_5 = t_5 + d(P,V) \wedge t_5 \geq t'_4 \wedge \lfloor d(P,V) \rfloor
\end{array}
$$

The Brands-Chaum protocol is designed to defend against mafia frauds, where an honest prover is outside the neighborhood of the verifier (i.e., $d(P,V) > d$) but an intruder is inside (i.e., $d(I,V) \leq d$), pretending to be the honest prover as depicted in Figure 5.1. The following is an example of an *attempted* mafia fraud, in which the intruder simply forwards messages back and forth between the prover and the verifier. We write $I(P)$ to denote an intruder pretending to be an honest prover P.

$$
\begin{array}{rll}
P_{t_1} \rightarrow I_{t_2} & : commit(N_P, S_P) & \mid t_2 = t_1 + d(P,I) \wedge \lfloor d(P,I) \rfloor \\
I(P)_{t_2} \rightarrow V_{t_3} & : commit(N_P, S_P) & \mid t_3 = t_2 + d(V,I) \wedge \lfloor d(V,I) \rfloor \\
V_{t_3} \rightarrow I(P)_{t_4} & : N_V & \mid t_4 = t_3 + d(V,I) \wedge \lfloor d(V,I) \rfloor \\
I_{t_4} \rightarrow P_{t_5} & : N_V & \mid t_5 = t_4 + d(P,I) \wedge \lfloor d(P,I) \rfloor \\
P_{t_5} \rightarrow I_{t_6} & : N_P \oplus N_V & \mid t_6 = t_5 + d(P,I) \wedge \lfloor d(P,I) \rfloor \\
I(P)_{t_6} \rightarrow V_{t_7} & : N_P \oplus N_V & \mid t_7 = t_6 + d(V,I) \wedge \lfloor d(V,I) \rfloor \\
V & : t_7 \dot{-} t_3 \leq 2 * d & \\
P_{t_8} \rightarrow I_{t_9} & : S_P & \mid t_9 = t_8 + d(P,I) \wedge t_8 \geq t_5 \wedge \lfloor d(P,I) \rfloor \\
I(P)_{t_{10}} \rightarrow V_{t_{11}} & : S_P & \mid t_{11} = t_{10} + d(V,I) \wedge t_{11} \geq t_7 \wedge \lfloor d(V,I) \rfloor \\
I(P)_{t_{12}} \rightarrow V_{t_{13}} & : sign_P(N_V; N_P \oplus N_V) & \mid t_{13} = t_{12} + d(V,I) \wedge t_{13} \geq t_{11} \wedge \lfloor d(V,I) \rfloor
\end{array}
$$

This attack is physically unfeasible, since it would require that $2 * d(V,I) + 2 * d(P,I) \leq 2 * d$, which is unsatisfiable by $d(V,P) > d > 0$ and the triangular inequality $d(V,P) \leq d(V,I) + d(P,I)$, satisfied in three-dimensional space. This attack was already unfeasible in Chapter 4 using only the metric space assumptions.

However, a distance hijacking attack is possible (i.e., the time and distance constraints are satisfiable), as depicted in Figure 5.2, where an intruder located outside the neighborhood of the verifier (i.e., $d(V,I) > d$) succeeds in convincing the verifier that he is inside the neighborhood by exploiting the presence of an honest prover in the neighborhood (i.e., $d(V,P) \leq d$) to achieve his goal. The following is an example of a *successful* distance hijacking, in which the intruder listens to the exchanges messages between the prover and the verifier but builds the last message.

$$
\begin{array}{rll}
P_{t_1} \rightarrow V_{t_2} & : commit(N_P, S_P) & \mid t_2 = t_1 + d(P,V) \wedge \lfloor d(P,V) \rfloor \\
V_{t_2} \rightarrow P_{t_3}, I_{t'_3} & : N_V & \mid t_3 = t_2 + d(P,V) \wedge \lfloor d(P,V) \rfloor \\
 & & \mid t'_3 = t_2 + d(I,V) \wedge \lfloor d(V,I) \rfloor \\
P_{t_3} \rightarrow V_{t_4}, I_{t'_4} & : N_P \oplus N_V & \mid t_4 = t_3 + d(P,V) \wedge \lfloor d(P,V) \rfloor \\
 & & \mid t'_4 = t_3 + d(I,P) \wedge \lfloor d(I,P) \rfloor \\
V & : t_4 \dot{-} t_2 \leq 2 * d & \\
P_{t_5} \rightarrow V_{t_6} & : S_P & \mid t_6 = t_5 + d(P,V) \wedge \lfloor d(P,V) \rfloor \\
 & & \mid t_5 \geq t_3 \wedge t_6 \geq t_4 \\
I(P)_{t_7} \rightarrow V_{t_8} & : sign_I(N_V; N_P \oplus N_V) & \mid t_8 = t_7 + d(I,V) \wedge \lfloor d(I,V) \rfloor \\
 & & \mid t_7 \geq t'_4 \wedge t_8 \geq t_6
\end{array}
$$

This attack was feasible in Chapter 4 using the metric space assumptions, and it is also possible in three-dimensional space. Note that an attack may be possible in *some* metric space but it may not be possible in *all* metric spaces, let alone in Euclidean metric spaces like three-dimensional space. This inspired us to add location to our previous framework, as motivated by the following protocol.

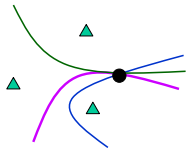


Figure 5.3: Trilateration

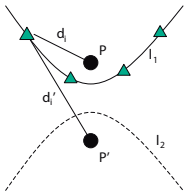


Figure 5.4: Insecure

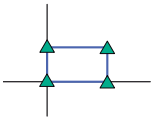


Figure 5.5: Secure

Example 20. *A secure localization protocol determines the physical location of a mobile device such as a sensor, a mobile phone, or a small computer with applications to location-based access control and security. In [109], a malicious device may lie about its location in an environment with different beacons to appear either farther away than its true location or closer than it really is.*

We consider a very simple protocol in two-dimensional space. A device sends a timestamp to different beacons. All beacons are honest and receive the timestamp. Figure 5.3 shows how three beacons infer the position of the device by trilateration, i.e., the intersection of the hyperbolas associated to the distance travelled from the device's location. There is a base station that receives the positions inferred by the beacons and checks whether they coincide or not.

$D \rightarrow Be^i : timestamp$
//The device broadcasts a timestamp, maybe different to its
//actual time to appear farther or closer than its true location

$Be^i \rightarrow Ba : timediff\ ;\ Be^i_x\ ;\ Be^i_y$
//Each beacon sends to a base station the difference between
//the received timestamp and the actual reception time plus
//her position.

An informal Alice-Bob presentation with time and location is as follows, where (D^i_x, D^i_y) is the inferred location of the device D according to beacon Be^i. The base calculates whether the positions of the device inferred by the beacons coincide, in symbols $D^1_x = \cdots = D^n_x$ and $D^1_y = \cdots = D^n_y$.

$$
\begin{array}{rll}
D_{t_1} \to Be^i_{t'_1} : t & & \mid t'_1 = t_1 + d(D, Be^i) \wedge \lfloor d(D, Be^i) \rfloor \\
Be^i : \bar{t} = t \dot{-} t'_1 & & \mid \bar{t} \geq 0 \\
Be^i_{t_2} \to Ba_{t'_2} : \bar{t} \ ; \ Be^i_x \ ; \ Be^i_y & & \mid t'_2 = t_2 + d(Be^i, Ba) \wedge \lfloor d(Be^i, Ba) \rfloor \\
Ba : \bar{t}^2 = (D^1_x - Be^1_x)^2 + (D^1_y - Be^1_y)^2 & & \\
\vdots & & \\
Ba : \bar{t}^2 = (D^n_x - Be^n_x)^2 + (D^n_y - Be^n_y)^2 & & \\
Ba : D^1_x = \cdots = D^n_x \wedge D^1_y = \cdots = D^n_y & &
\end{array}
$$

If the device is honest, the constraints on the real numbers computed by the base station are always satisfied. If the device is malicious, [109] shows two interesting configurations.

(i) (Insecure configuration) If the beacons are in the same lobe of a hyperbola, as shown in Figure 5.4, it is possible for a malicious device at position P to choose a timestamp to pretend to be at position P'.

(ii) (Secure configuration) If there are four beacons and they form a rectangle, shown in Figure 5.5, then [109] proves that the device is always caught by the base station.

Note that these two statements, (i) an attack and (ii) the absence of any attack, are verified by our time and space process algebra below without requiring exact positions. That is, (i) is verified just by showing an execution where the computed time and space constraints are satisfied and (ii) is verified by obtaining a finite search space where all the computed time and space constraints are unsatisfiable.

5.3 A Time and Space Process Algebra

In Chapter 4 we provided a timed process algebra syntax and a timed transition semantics. The timed process algebra only made message sending-and-reception times available to processes whereas the timed transition semantics modelled the actual time interactions between processes under metric space constraints. In this section, we extend the previous process algebra syntax to make spatial location information also available to processes and provide a transition semantics that models the actual time and space interactions between processes and Euclidean space constraints.

5.3.1 New Syntax for Location

In our time and space protocol process algebra, the behaviors of both honest principals and the intruders are represented by *labeled processes*. Therefore, a protocol is specified as a set of labeled processes. Each process performs a sequence of actions, namely sending $(+m)$ or receiving $(-m)$ a message m, but without knowing who actually sent it or received it. Each process may also perform deterministic or non-deterministic choices. We define a protocol $\mathscr{P}$ in the time and space protocol process algebra, written $\mathscr{P}_{TPA}$, as a pair of the form $\mathscr{P}_{TPA} = ((\Sigma_{TPA_{\mathscr{P}}}, E_{TPA_{\mathscr{P}}}), P_{TPA})$, where $(\Sigma_{TPA_{\mathscr{P}}}, E_{TPA_{\mathscr{P}}})$ is the equational theory specifying the equational properties of the cryptographic functions and the state structure, and P_{TPA} is a $\Sigma_{TPA_{\mathscr{P}}}$-term denoting a *well-formed* time and space process. The time and space protocol process algebra's syntax Σ_{TPA} is parameterized by a sort Msg of messages. Moreover, time and coordinates are represented by a new sort Real, since we allow conditional expressions on time and location to be constraints in real polynomial arithmetic.

Similar to [13, 115], processes support four different kinds of choice: (i) a process expression P ? Q supports *explicit non-deterministic choice* between P and Q; (ii) a choice variable $X_?$ appearing in a send message expression +m supports *implicit non-deterministic choice* of its value, which can furthermore be an *unbounded* non-deterministic choice if $X_?$ ranges over an infinite set; (iii) a conditional *if C then P else Q* supports *explicit deterministic choice* between P and Q determined by the result of its condition C; and (iv) a receive message expression $-m(X_1,...,X_n)$ supports *implicit deterministic* choice about accepting or rejecting a received message, depending on whether or not it matches the pattern $m(X_1,...,X_n)$. This deterministic choice is implicit, but it could be made explicit by replacing $-m(X_1,...,X_n) \cdot P$ by the semantically equivalent conditional expression $-X.$ *if* $X = m(X_1,...,X_n)$ *then* P *else nilP*, where X is a variable of sort Msg, which therefore accepts any message.

The time and space process algebra has the following syntax, also similar to that of [13, 115] plus the addition of the suffix *@Real* to the sending and receiving actions:

$$
\begin{aligned}
ProcConf &::= LProc \mid ProcConf \ \&\ ProcConf \mid \emptyset \\
ProcId &::= (Role, Nat) \\
LProc &::= (ProcId, Nat, Real, Real, Real)\ Proc \\
Proc &::= nilP \mid +(Msg@Real) \mid -(Msg@Real) \mid Proc \cdot Proc \mid \\
&\quad\ Proc\ ?\ Proc \mid \mathit{if}\ Cond\ \mathit{then}\ Proc\ \mathit{else}\ Proc
\end{aligned}
$$

- *ProcConf* stands for a *process configuration*, i.e., a set of labeled processes, where the symbol & is used to denote set union for sets of labeled processes.

- *ProcId* stands for a *process identifier*, where *Role* refers to the role of the process in the protocol (e.g., prover or verifier) and *Nat* is a natural number denoting the identity of the process, which distinguishes different instances (sessions) of a process specification.

- *LProc* stands for a *labeled process*, i.e., a process *Proc* with a label $(ProcId, J)$. For convenience, we sometimes write $(Role, I, J, x, y, z)$, where J indicates that the action at stage J of the process $(Role, I)$ will be the next one to be executed, i.e., the first $J-1$ actions of the process for role *Role* have already been executed. The three *Real* elements x, y, z represent the coordinates on three-dimensional space. Note that the I and J of a process $(Role, I, J, x, y, z)$ are omitted in a protocol specification.

- *Proc* defines the actions that can be executed within a process, where $+Msg@T$, and $-Msg@T$ respectively denote sending out a message or receiving a message *Msg*. Note that T must be a variable where the underlying Euclidean space determines the exact sending or receiving time, which can be used later in the process. Moreover, "*Proc* · *Proc*" denotes *sequential composition* of processes, where symbol _ · _ is associative and has the empty process *nilP* as identity. Finally, "*Proc* ? *Proc*" denotes an explicit *nondeterministic choice*, whereas "*if Cond then Proc else Proc*" denotes an explicit *deterministic choice*, whose continuation depends on the satisfaction of the constraint *Cond*. Note that choice is explicitly represented by either a non-deterministic choice between P_1 ? P_2 or by the deterministic evaluation of a conditional expression *if Cond then* P_1 *else* P_2, but it is also implicitly represented by the instantiation of a variable in different runs.

In all process specifications we assume five disjoint kinds of variables, similar to the variables of [115] plus time variables as in Chapter 4 and location coordinate variables:

- ***fresh variables***: each one of these variables receives a *distinct constant value* from a data type $\mathsf{V}_{\mathsf{fresh}}$, denoting unguessable values such as nonces. Throughout this paper we will denote this kind of variables as $f, f_1, f_2, \ldots$.

- ***choice variables***: variables first appearing in a *sent message* $+M$, which can be substituted by any value arbitrarily chosen from a possibly infinite domain. A choice variable indicates an *implicit non-deterministic choice*. Given a protocol with choice variables, each possible substitution of these variables denotes a possible run of the protocol. We always denote choice variables by letters postfixed with the symbol "?" as a subscript, e.g., $A_?, B_?, \ldots$.

- ***pattern variables***: variables first appearing in a *received message* $-M$. These variables will be instantiated when matching sent and received messages. *Implicit deterministic choices* are indicated by terms containing pattern variables, since failing to match a pattern term leads to the rejection of a message. A pattern term plays the implicit role of a guard, so that, depending on the different ways of matching it, the protocol can have different continuations. Pattern variables are written with uppercase letters, e.g., $A, B, N_A, \ldots$.

- ***time variables***: a process cannot access the global clock, which implies that a time variable T of a reception or sending action $+(M@T)$ can never appear in M but can appear in the remaining part of the process. Also, given a receiving action $-(M_1@t_1)$ and a sending action $+(M_2@t_2)$ in a process of the form $P_1 \cdot -(M_1@t_1) \cdot P_2 \cdot +(M_2@t_2) \cdot P_3$, the assumption that timed actions are performed from left to right forces the constraint $t_1 \leq t_2$. Time variables are always written with a (subscripted) t, e.g., $t_1, t'_1, t_2, t'_2, \ldots$.

- ***coordinate variables***: a process can only access its own coordinates x, y, and z. Its coordinates can be sent and coordinate variables can be received, sent again and used in comparisons. The location of a process never changes, so coordinate variables can never be updated. Coordinate variables are always written with a (subscripted) x, y or z, e.g., $x_1, x'_1, y_2, z'_2, \ldots$.

These requirements about variables are formalized by the function $wf : Proc \rightarrow Bool$ for *well-formed* processes given in Chapter 4. The definition of wf uses an auxiliary function $shVar : Proc \rightarrow VarSet$ also given in Chapter 4.

Example 21. *Let us specify the Brands and Chaum protocol of Example 19, where variables are distinct between processes. A nonce is represented as $n(A_?, f)$, whereas a secret value is represented as $s(A_?, f)$. The identifier of each process is represented*

by a choice variable $A_?$. Recall that there is an arbitrary distance $d > 0$. Since participants in this protocol do not make use of their own coordinates, the following specification is identical to that of Chapter 4.

$$
\begin{aligned}
(\mathit{Verifier}, x, y, z) : & -(\mathit{Commit}@t_1) \cdot \\
& +(n(V_?, f_1)@t_2) \cdot \\
& -((n(V_?, f_1) \oplus N_P)@t_3) \cdot \\
& \mathit{if}\ t_3 \dot{-} t_2 \leq 2 * d \\
& \mathit{then}\ -(S_P@t_4) \cdot \\
& \qquad \mathit{if}\ \mathit{open}(N_P, S_P, \mathit{Commit}) \\
& \qquad \mathit{then}\ -(\mathit{sign}(P, n(V_?, f_1); N_P \oplus n(V_?, f_1))@t_5) \\
(\mathit{Prover}, x, y, z) : & +(\mathit{commit}(n(P_?, f_1), s(P_?, f_2))@t_1) \cdot \\
& -(N_V@t_2) \cdot \\
& +((N_V \oplus n(P_?, f_1))@t_3) \cdot \\
& +(s(P_?, f_2)@t_4) \cdot \\
& +(\mathit{sign}(P_?, N_V; n(P_?, f_1) \oplus N_V)@t_5)
\end{aligned}
$$

Example 22. *Let us specify the secure localization protocol of Example 20 for four beacons. The timestamp is represented by variable t.*

$$
\begin{aligned}
(Be^i, x, y, z) : & -(t@t_1) \cdot \\
& +(((t \dot{-} t_1)\ ;\ x\ ;\ y)@t_2) \cdot \\
& -((ok@t_3)\ \cdot \mathit{nilP} \\
(Ba, x, y, z) : & -((t_1'\ ;\ x_1\ ;\ y_1)@t_1) \cdot \\
& -((t_2'\ ;\ x_2\ ;\ y_2)@t_2) \cdot \\
& -((t_3'\ ;\ x_3\ ;\ y_3)@t_3) \cdot \\
& -((t_4'\ ;\ x_4\ ;\ y_4)@t_4) \cdot \\
& \mathit{if}\ \exists dx, dy : (t_1')^2 = (dx - x_1)^2 + (dy - y_1)^2 \wedge \\
& \qquad\qquad (t_2')^2 = (dx - x_2)^2 + (dy - y_2)^2 \wedge \\
& \qquad\qquad (t_3')^2 = (dx - x_3)^2 + (dy - y_3)^2 \wedge \\
& \qquad\qquad (t_4')^2 = (dx - x_4)^2 + (dy - y_4)^2 \quad \mathit{then}\ +(ok@t_5)\ \mathit{else}\ \mathit{nilP}
\end{aligned}
$$

5.3.2 Time and Space Intruder Model

The active Dolev-Yao intruder model is followed, which implies that an intruder can intercept, forward, or create messages from received messages. We assume that intruders are *located* and cannot change their location, as in Chapter 4. In particular, they cannot change the physics of the metric space, e.g., cannot send messages from a different location or intercept a message that it is not within range.

In our time and space intruder model, we consider several located intruders, each with its own coordinates, with a family of capabilities (concatenation, deconcatenation, encryption, decryption, etc.), and each capability may have arbitrarily many instances. The combined actions of two intruders requires time, i.e., their distance; but a single intruder can perform many actions in zero time[1]. Note that, unlike in the standard Dolev-Yao model, we cannot assume just one intruder, since the time required for a principal to communicate with a given intruder is an observable characteristic of that intruder. Thus, although the Mafia fraud and distance hijacking attacks of the Brands and Chaum protocol and the insecure and secure configurations of the secure localization protocol only require one intruder, the framework itself allows general participant configurations with multiple intruders; although one intruder co-located with each honest participant is enough [97].

Example 23. *In our timed process algebra, the family of capabilities associated to an intruder k are also described as processes. For instance, concatenating two received messages is represented by the process*[2]

$$(k.Conc, x, y, z) : \ -(X@t_1) \cdot -(Y@t_2) \cdot +(X;Y@t_3)$$

and extracting one of them from a concatenation is described by the processes

$$(k.DeconcLeft, x, y, z) : \ -(X;Y@t_1) \cdot +(X@t_2)$$

$$(k.DeconcRight, x, y, z) : \ -(X;Y@t_1) \cdot +(X@t_2)$$

Roles of intruder capabilities include the identifier of the intruder, and it is possible to combine several intruder capabilities from the same or from different intruders. For example, we may say that the $+(X;Y@t)$ of a process $I1.Conc$ associated to an intruder $I1$ may be synchronized with the $-(X;Y@t')$ of a process $I2.DeconcLeft$ associated to an intruder $I2$. The physical space determines that $t' = t + d(I1, I2)$, where $d(I1, I2) > 0$ if $I1 \neq I2$ and $d(I1, I2) = 0$ if $I1 = I2$.

As presented in Chapter 4, a special forwarding *intruder capability, not considered in the standard Dolev-Yao model, has to be included in order to take into account the time travelled by a message from an honest participant to the intruder and later to another participant, possibly another intruder.*

$$(k.Forward, x, y, z) : \ -(X@t_1) \cdot +(X@t_2)$$

[1] Adding time cost to single-intruder actions could be done with additional time constraints, but is outside the scope of this paper.

[2] Time variables t_1, t_2, t_3 as well as its coordinates are not actually used by the intruder but could be in the future.

5.3.3 Time and Space Process Semantics

A *state* of a protocol $\mathscr{P}$ consists of a set of (possibly partially executed) *labeled processes*, a set of terms in the network $\{Net\}$, and the global clock. That is, a state is a term of the form $\{LP_1 \,\&\, \cdots \,\&\, LP_n \mid \{Net\} \mid \bar{t}\}$.

In Chapter 4, the only time information available to a process is the variable T associated to input and output messages $M@T$; the global clock is inaccessible. However, once these messages have been sent or received, we included them in the network *Net* with extra information. When a message $M@T$ is sent, we stored $M \;@\; (A : t \to \emptyset)$ denoting that message M was sent by process A at the global time clock t, and propagated $T \mapsto t$ within the process A. When this message is received by an action $M'@T'$ of process B (honest participant or intruder) at the global clock time t', M is matched against M' modulo the cryptographic functions, $T' \mapsto t'$ is propagated within the process B, and $B : t'$ is added to the stored message, following the general pattern $M \;@\; (A : t \to (B_1 : t_1 \cdots B_n : t_n))$.

In our new time and space process algebra, we simply annotate stored messages with the coordinates from where the message was sent and, when the message is received by another process, we calculate actual distances between the stored coordinates and the coordinates of the current process. When a message $M@T$ is sent by process (A,x,y,z), we store $M \;@\; (A : x,y,z,t \to \emptyset)$ denoting that message M was sent at the global time clock t from location (x,y,z). When this message is received by an action $M'@T'$ of process B (honest participant or intruder) at the global clock time t', M is matched against M' modulo the cryptographic functions, $T' \mapsto t'$ is propagated within the process B, and $B : t'$ is added to the stored message, following the general pattern $M \;@\; (A : x,y,z,t \to (B_1 : t_1 \cdots B_n : t_n))$. No reception coordinates are stored, but we check that process B is reachable from process A at distance $t' - t$, i.e., $(t'-t)^2 = (B_x - A_x)^2 + (B_y - A_y)^2 + (B_z - A_z)^2$.

The rewrite theory $(\Sigma_{TPA_{\mathscr{P}}+State}, E_{TPA_{\mathscr{P}}}, R_{TPA_{\mathscr{P}}})$ characterizes the behavior of a protocol $\mathscr{P}$, where $\Sigma_{TPA_{\mathscr{P}}+State}$ extends $\Sigma_{TPA_{\mathscr{P}}}$, by adding state constructor symbols. We assume that a protocol run begins with an empty state, i.e., a state with an empty set of labeled processes, an empty network, and at time zero. Therefore, the initial empty state is always of the form $\{\emptyset \mid \{\emptyset\} \mid 0.0\}$. Note that, in a specific run, all the distances are provided a priori according to the Euclidean space and a chosen topology, whereas in a symbolic analysis, they will be represented by variables, probably occurring within space and time constraints.

State changes are defined by a set $R_{TPA_{\mathscr{P}}}$ of *rewrite rules* given below. Each transition rule in $R_{TPA_{\mathscr{P}}}$ is labeled with a tuple (ro, i, j, a, n, t), where:

- ro is the role of the labeled process being executed in the transition.
- i denotes the instance of the same role being executed in the transition.
- j denotes the process' step number since its beginning.
- a is a ground term identifying the action that is being performed in the transition. It has different possible values: "$+m$" or "$-m$" if the message m was sent (and added to the network) or received, respectively; "m" if the message m was sent but did not increase the network, "?" if the transition performs an explicit non-deterministic choice, "T" if the transition performs an explicit deterministic choice, "*Time*" when the global clock is incremented, or "*New*" when a new process is added.
- n is a number that, if the action that is being executed is an explicit choice, indicates which branch has been chosen as the process continuation. In this case n takes the value of either 1 or 2. If the transition does not perform any explicit choice, then $n = 0$.
- t is the global clock at each transition step.

Note that in the transition rules $R_{TPA_{\mathscr{P}}}$ shown below, *Net* denotes the network, represented by a set of messages of the form M @ $(A : x, y, z, t \rightarrow (B_1 : t_1 \cdots B_n : t_n))$, P denotes the rest of the process being executed and PS denotes the rest of labeled processes of the state (which can be the empty set $\emptyset$).

- *Sending a message* is represented by the two transition rules below, depending on whether the message M is stored, (TPA++), or is just discarded, (TPA+). In (TPA++), we store the sent message with its sending information, $(ro, i) : \bar{t}$, and add an empty set for those who will be receiving the message in the future $(M\sigma'@(ro, i) : x, y, z, \bar{t} \rightarrow \emptyset)$.

$$\{(ro,i,j,x,y,z)\ (+M@t\cdot P)\ \&\ PS \mid \{Net\} \mid \bar{t}\}$$
$$\longrightarrow_{(ro,i,j,+(M\sigma'),0,\bar{t})}$$
$$\{(ro,i,j+1,x,y,z)\ P\sigma'\ \&\ PS \mid \{(M\sigma'@(ro,i):x,y,z,\bar{t}\to\emptyset),Net\} \mid \bar{t}\}$$
$$\textit{if } (M\sigma':(ro,i):x,y,z,\bar{t}\to\emptyset)\notin Net$$
$$\textit{where } \sigma \textit{ is a ground substitution binding choice variables in } M$$
$$\textit{and } \sigma' = \sigma \uplus \{t\mapsto\bar{t}\} \qquad \text{(TPA++)}$$

$$\{(ro,i,j,x,y,z)\ (+M@t\cdot P)\ \&\ PS \mid \{Net\} \mid \bar{t}\}$$
$$\longrightarrow_{(ro,i,j,M\sigma',0,\bar{t})}$$
$$\{(ro,i,j+1,x,y,z)\ P\sigma'\ \&\ PS \mid \{Net\} \mid \bar{t}\}$$
$$\textit{where } \sigma \textit{ is a ground substitution binding choice variables in } M$$
$$\textit{and } \sigma' = \sigma \uplus \{t\mapsto\bar{t}\} \qquad \text{(TPA+)}$$

- *Receiving a message* is represented by the transition rule below. We add the reception information to the stored message, i.e., we replace $(M'@((ro',k):x',y',z',t'\to AS))$ by $(M'@((ro',k):x',y',z',t'\to(AS\uplus(ro,i):\bar{t}))$.

$$\{(ro,i,j,x,y,z)\ (-(M@t)\cdot P)\ \&\ PS \mid \{(M'@((ro',k):x',y',z',t'\to AS)),Net\} \mid \bar{t}\}$$
$$\longrightarrow_{(ro,i,j,-(M\sigma'),0,\bar{t})}$$
$$\{(ro,i,j+1,x,y,z)\ P\sigma'\ \&\ PS \mid \{(M'@((ro',k):x',y',z',t'\to(AS\uplus(ro,i):\bar{t})),Net\} \mid \bar{t}\}$$
$$\mathsf{IF}\ \exists\sigma : M' =_{E_{\mathscr{D}}} M\sigma,\bar{t}=t'+d((x,y,z),(x',y',z')),\sigma'=\sigma\uplus\{t\mapsto\bar{t}\} \qquad \text{(TPA-)}$$

- An *explicit deterministic choice* is defined as follows. More specifically, the rule (TPAif1) describes the *then* case, i.e., if the constraint T is satisfied, then the process continues as P, whereas rule (TPAif2) describes the *else* case, that is, if the constraint C is *not* satisfied, the process continues as Q.

$$\{(ro,i,j,x,y,z)\ ((\textit{if } C \textit{ then } P \textit{ else } Q)\cdot R)\ \&\ PS \mid \{Net\} \mid \bar{t}\}$$
$$\longrightarrow_{(ro,i,j,C,1,\bar{t})} \{(ro,i,j+1,x,y,z)\ (P\cdot R)\,\&\,PS \mid \{Net\} \mid \bar{t}\}\ \mathsf{IF}\ C \qquad \text{(TPAif1)}$$
$$\{(ro,i,j,x,y,z)\ ((\textit{if } C \textit{ then } P \textit{ else } Q)\cdot R)\ \&\ PS \mid \{Net\} \mid \bar{t}\}$$
$$\longrightarrow_{(ro,i,j,C,2,\bar{t})} \{(ro,i,j+1,x,y,z)\ (Q\cdot R)\,\&\,PS \mid \{Net\} \mid \bar{t}\}\ \mathsf{IF}\ \neg C \qquad \text{(TPAif2)}$$

- An *explicit non-deterministic choice* is defined as follows. The process can continue either as P, denoted by rule (TPA?1), or as Q, denoted by rule (TPA?2).

$$\{(ro,i,j,x,y,z)\ ((P\ ?\ Q)\cdot R)\ \&\ PS \mid \{Net\} \mid \bar{t}\}$$
$$\longrightarrow_{(ro,i,j,?,1,\bar{t})} \{(ro,i,j+1,x,y,z)\ (P\cdot R)\ \&\ PS \mid \{Net\} \mid \bar{t}\} \qquad \text{(TPA?1)}$$

$$\{(ro,i,j,x,y,z)\ ((P\ ?\ Q)\cdot R)\ \&\ PS \mid \{Net\} \mid \bar{t}\}$$
$$\longrightarrow_{(ro,i,j,?,2,\bar{t})} \{(ro,i,j+1,x,y,z)(Q\cdot R)\ \&\ PS \mid \{Net\} \mid \bar{t}\} \qquad \text{(TPA?2)}$$

- *Global Time advancement* is represented by the transition rule below that increments the global clock enough to make one sent message arrive to its closest destination.

$$\{PS \mid \{Net\} \mid \bar{t}\} \longrightarrow_{(\bot,\bot,\bot,Time,0,\bar{t}+t')} \{PS \mid \{Net\} \mid \bar{t}+t'\}$$
$$\mathsf{IF}\ t' = mte(PS,Net,\bar{t}) \wedge t' \neq 0 \qquad \text{(PhyTime)}$$

where the function *mte* is defined as follows:

$$mte(\emptyset,Net,\bar{t}) = \infty$$
$$mte(P\&PS,Net,\bar{t}) = min(mte(P,Net,\bar{t}),mte(PS,Net,\bar{t}))$$
$$mte((ro,i,j,x,y,z)\ nilP,Net,\bar{t}) = \infty$$
$$mte((ro,i,j,x,y,z)\ +(M@t)\cdot P,Net,\bar{t}) = 0$$
$$mte((ro,i,j,x,y,z)\ -(M@t)\cdot P,Net,\bar{t}) =$$
$$min\left(\left\{\begin{array}{c} d((x,y,z),(x',y',z')) \mid (M'@(ro',i') : x',y',z',t' \to AS) \in Net \\ \wedge \exists \sigma : M\sigma =_B M' \end{array}\right\}\right)$$
$$mte((ro,i,j,x,y,z)\ (if\ T\ then\ P\ else\ Q)\cdot R,Net,\bar{t}) = 0$$
$$mte((ro,i,j,x,y,z)\ P_1?P_2,Net,\bar{t}) = 0$$

Note that the function *mte* evaluates to 0 if some instantaneous action by the previous rules can be performed. Otherwise, *mte* computes the smallest non-zero time increment required for some already sent message (existing in the network) to be received by some process (by matching with such an existing message in the network).

Further time and space constraints In [13], the timed process semantics assumed only a metric space with a distance function $d : ProcId \times ProcId \to Real$ such that (i) $d(A,A) = 0$, (ii) $d(A,B) = d(B,A)$, and (iii) $d(A,B) \leq d(A,C) + d(C,B)$. For every message M @ $(A : t \to (B_1 : t_1 \cdots B_n : t_n))$ stored in the network *Net*, the semantics ensured that (iv) $t_i =$

$t + d(A,B_i)$, $\forall 1 \leq i \leq n$. Furthermore, according to our wireless communication model, our semantics assumed (v) a *time sequence monotonicity* property, i.e., there is no other process C such that $d(A,C) \leq d(A,B_i)$ for some i, $1 \leq i \leq n$, and C is not included in the set of recipients of the message M. Also, for each class of attacks such as the Mafia fraud or the hijacking attack analyzed in Chapter 4, (vi) some extra topology constraints were necessary (see Figures 5.1 and 5.2).

In our time and space semantics, all those assumptions except (v) are unnecessary by considering actual coordinates. This simplifies the transformation of time and space processes into untimed processes of Section 5.4 compared to the transformation presented in Chapter 4.

- New processes can be added as follows.

$$\left\{ \begin{array}{l} \forall\ (ro)\ P_k \in P_{PA} \\ \{PS \mid \{Net\} \mid \bar{t}\} \\ \xrightarrow{(ro,i+1,1,New,0,\bar{t})} \\ \{(ro,i+1,1,x_?\sigma, y_?\sigma, z_?\sigma)\ P_k\sigma\rho_{ro,i+1}\ \&\ PS \mid \{Net\} \mid \bar{t}\} \\ \textit{where } \rho_{ro,i+1} \textit{ is a fresh substitution,} \\ \quad \sigma \textit{ is a ground substitution binding } x_?, y_?, z_?, \\ \quad \text{and } i = id(PS,ro) \end{array} \right\} \qquad \text{(TPA\&)}$$

The auxiliary function id counts the instances of a role

$$id(\emptyset, ro) = 0$$
$$id((ro',i,j,x,y,z)P\&PS, ro) = \begin{cases} max(id(PS,ro),i) & \textit{if } ro = ro' \\ id(PS,ro) & \textit{if } ro \neq ro' \end{cases}$$

where PS denotes a process configuration, P a process, and ro, ro' role names.

Therefore, the behavior of a timed protocol in the process algebra is defined by the set of transition rules $R_{TPA_{\mathscr{P}}} = \{\text{(TPA++)}, \text{(TPA+)}, \text{(PhyTime)}, \text{(TPA-)}, \text{(TPAif1)}, \text{(TPAif2)}, \text{(TPA?1)}, \text{(TPA?2)}\} \cup \text{(TPA\&)}$.

Example 24. *Continuing Example 21, it is possible to create a configuration of the Brands and Chaum for the mafia attack (which are impossible due to unsatisfiability of the timing and distance constraints) with a prover p, an intruder i, and a verifier v. The neighborhood distance is set to $d = 1.0$, the verifier is at coordinates $(0,0,0)$,*

the prover is at $(2,0,0)$*, and the intruder is at* $(1,0,0)$*. That is, the distance between the prover and the verifier is* $d(p,v) = 2.0$*, but the distance between the prover and the intruder as well as the distance between the verifier and the intruder are* $d(v,i) = d(p,i) = 1.0$*, i.e., the honest prover p is outside v's neighborhood,* $d(v,p) > d$*, where* $d(v,p) = d(v,i) + d(p,i)$*.*

Example 25. *Continuing Example 22, it is possible to create a configuration of the beacons protocol where a malicious device is caught cheating. We consider four beacons at the following positions in two-dimensional space (we omit* $z = 0$*):* Be^1 : $(0,0)$*,* Be^2 : $(4,0)$*,* Be^3 : $(0,3)$*, and* Be^4 : $(4,3)$*. We assume a device at position* $(4,6)$*. If the device is honest and sends the right timestamp, the distances* d_1, d_2, d_3, d_4 *computed by each beacon are*

$$
\begin{aligned}
d_1 &= \sqrt{(4-0)^2 + (6-0)^2} = \sqrt{16+36} = \sqrt{52}, \\
d_2 &= \sqrt{(4-4)^2 + (6-0)^2} = \sqrt{36} = 6, \\
d_3 &= \sqrt{(4-0)^2 + (6-3)^2} = \sqrt{16+9} = \sqrt{25} = 5, \\
d_4 &= \sqrt{(4-4)^2 + (6-4)^2} = \sqrt{9} = 3
\end{aligned}
$$

the base station receives the distances and the beacons positions and computes the following set of equations

$$
\begin{aligned}
52 &= (x-0)^2 + (y-0)^2 \\
36 &= (x-4)^2 + (y-0)^2 \\
25 &= (x-0)^2 + (y-3)^2 \\
9 &= (x-3)^2 + (y-3)^2
\end{aligned}
$$

and it is not difficult to calculate x and y by Gaussian elimination:

$$
\begin{aligned}
x &= ((36+16-36)+16)/8 = (16+16)/8 = 4 \\
y &= ((52-25)+9)/6 = (27+9)/6 = 36/6 = 6
\end{aligned}
$$

If the device is malicious and sends the original timestamp plus 1 *unit, the distances* d_1, d_2, d_3, d_4 *computed by each beacon are* $d_1 = \sqrt{52} - 1$*,* $d_2 = 6 - 1 = 5$*,* $d_3 = 5 - 1 = 4$*, and* $d_4 = 3 - 1 = 2$*. But when plugged in the previous equations, it is easy to check that the device is lying.*

As it already happened in Chapter 4 with our timed protocol semantics, our new time and space protocol semantics can be implemented straightforwardly as a simulation tool. Note, however, that, since the number of different topologies is infinite, model checking a protocol for a *concrete configuration* with a simulation tool is very limited, since it cannot prove the *absence* of an attack for *all* topologies. For this reason, we follow a symbolic approach that can explore *all* relevant configurations.

In the following section we provide a sound and complete protocol transformation from our time and space process algebra to the untimed process algebra with constraints of the Maude-NPA tool, in a similar manner to the protocol transformation provided in Chapter 4. In order to do this, we represent time and location information as well as those constraints checked by the participants as real arithmetic constraints. As a path is built, an SMT solver can be used to check that the constraints are satisfiable as we did in [13] only for time.

5.4 Time and Space Process Algebra into Untimed Process Algebra

In this section, we extend the general constraint satisfiability approach of Chapter 4 where all possible (not only some) runs are symbolically analyzed. This time and space semantics provides both a *trace-based insecure statement*, i.e., a run leading to an insecure secrecy or authentication property where all constraints are satisfiable is discovered given enough resources, and an *unsatisfiability-based secure statement*, i.e., there is no run leading to an insecure secrecy or authentication property due to time and space constraint unsatisfiability.

Example 26. *Consider again the initial configuration for the Brands-Chaum protocol of Example 24. We can abstract away from the specific locations and just use logical variables for the coordinates of the prover* (p_x, p_y)*, the verifier* (v_x, v_y)*, and the intruder* (i_x, i_y)*. Then, it is possible to obtain a symbolic trace using logical variables* $\bar{t}_0, \ldots, \bar{t}_6$ *where the following time constraints are accumulated:*

$$t_1 = t_0 + d((p_x, p_y), (i_x, i_y))$$

$$t_2 = t_1 + d((v_x, v_y), (i_x, i_y))$$

$$t_3 = t_2 + d((v_x, v_y), (i_x, i_y))$$

$$t_4 = t_3 + d((p_x, p_y), (i_x, i_y))$$

$$t_5 = t_4 + d((p_x, p_y), (i_x, i_y))$$

$$t_6 = t_5 + d((v_x, v_y), (i_x, i_y))$$

Note that these constraints are unsatisfiable when combined with (i) the assumption $d > 0$, (ii) the verifier check $\bar{t}_6 - \bar{t}_2 \leq 2 * d$, (iii) the assumption that the honest prover is outside the verifier's neighborhood, $d((p_x, p_y), (v_x, v_y)) > d$, (iv) the triangular inequality $d((p_x, p_y), (v_x, v_y)) \leq d((p_x, p_y), (i_x, i_y)) + d((v_x, v_y), (i_x, i_y))$, and (v) the assumption that there is only one intruder.

Example 27. *Consider again the initial configuration for the beacons protocol of Example 25. Again, we can abstract away from the specific locations and just put logical variables for the coordinates of the four beacons $x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4$, the base station (but they are irrelevant) and the malicious device d_x, d_y. Then, it is possible to obtain a symbolic trace using logical variables $t_0, \ldots, t_4$ and the sent timestamp t where the following time and space constraints are accumulated:*

$$t_1 = t_0 + d((d_x, d_y), (x_1, y_1))$$

$$t_2 = t_0 + d((d_x, d_y), (x_2, y_2))$$

$$t_3 = t_0 + d((d_x, d_y), (x_3, y_3))$$

$$t_4 = t_0 + d((d_x, d_y), (x_4, y_4))$$

$$(t - t_1)^2 = (d_x - x_1)^2 + (d_y - y_1)^2$$

$$(t - t_2)^2 = (d_x - x_2)^2 + (d_y - y_2)^2$$

$$(t - t_3)^2 = (d_x - x_3)^2 + (d_y - y_3)^2$$

$$(t - t_4)^2 = (d_x - x_4)^2 + (d_y - y_4)^2$$

It is easy to check that a malicious device will be caught if $t \neq t_0$.

As explained previously, there are some implicit conditions based on the *mte* function to calculate the time increment to the closest destination of a message. However, the *mte* function is unnecessary in the untimed process algebra, where those implicit conditions are incorporated into the symbolic run. In the following, we define a transformation of the time and space process algebra by (i) removing the global clock; (ii) adding the time data into

untimed messages of a process algebra without time; (iii) adding the coordinates of each process to the untimed messages of a process algebra without space information; and (iv) adding real arithmetic conditions over the reals for the time and location constraints (those generated by the time and space semantics and those checked by the processes).

Since all the relevant time information is actually stored in messages of the form M @ $(A : x,y,z,t \rightarrow (B_1 : t_1 \cdots B_n : t_n))$ and controlled by the transition rules (TPA++),(TPA+), and (TPA-), the mapping *phy2pa* of Definition 11 below transforms each message $M@t$ of a timed process into a message M @ $(A : x_?, y_?, z_?, t_? \rightarrow AS_?)$ of an untimed process. That is, we use a timed choice variable $t_?$ for the sending time, choice variables $x_?, y_?, z_?$ for the coordinates and a variable $AS_?$ for the reception information $(B_1 : t'_1 \cdots B_n : t'_n)$ associated to the sent message. The transformation below ensures that the choice variables for the coordinates are all the same within the messages of the untimed process. Since choice variables are replaced by specific values, $t_?$,$x_?, y_?, z_?$ and $AS_?$ will be replaced by the appropriate values that make the execution and all its time and space constraints possible. Note that these choice variables will be replaced by logical variables during the symbolic execution.

Definition 11 (Adding Real Variables and Time and Space Constraints)**.** *The mapping phy2pa from time and space processes into untimed processes and its auxiliary mapping phy2pa* is defined as follows:*

$$
\begin{array}{l}
phy2pa(\emptyset) = \emptyset \\
phy2pa((ro,i,j,x,y,z)\,P \;\&\; PS) = (ro,i,j)\,phy2pa^*(P,ro,i,x,y,z) \;\&\; phy2pa(PS) \\
\\
phy2pa^*(nilP,ro,i,x,y,z) = nilP \\
phy2pa^*(\,+(M@t)\;.\;P,ro,i,x,y,z) = \\
\quad +(M@((ro,i) : x_?,y_?,z_?,t_? \rightarrow AS_?))\;.\;phy2pa^*(P\gamma,ro,i,x,y,z) \\
\quad\; where\ \gamma = \{t \mapsto t_?\} \\
phy2pa^*(\,-(M@t)\;.\;P,ro,i,x,y,z) = \\
\quad -(M@((ro',i') : x',y',z',t' \rightarrow ((ro,i) : t) \uplus AS))\;. \\
\quad if\ t = t' + d((x_?,y_?,z_?),(x',y',z'))\ then\ phy2pa^*(P,ro,i)\ else\ nilP \\
phy2pa^*(\,(if\ C\ then\ P\ else\ Q)\;.\;R,ro,i,x,y,z) \\
\quad = (if\ C\ then\ phy2pa^*(P,ro,i,x,y,z)\ else\ phy2pa^*(Q,ro,i,x,y))\;.\;phy2pa^*(R,ro,i,x,y,z) \\
phy2pa^*(\,(P\ ?\ Q)\;.\;R,ro,i,x,y,z) \\
\quad = (phy2pa^*(P,ro,i,x,y,z)\ ?\ phy2pa^*(Q,ro,i,x,y,z))\;.\;phy2pa^*(R,ro,i,x,y,z)
\end{array}
$$

where $t_?$ and $AS_?$ are choice variables different for each one of the sending actions, $x_?, y_?, z_?$ are always the same variables for all sending or receiving actions,

$ro', i', t', d, x', y', z', AS$ are pattern variables different for each one of the receiving actions, P, Q, and R are processes, M is a message, and C is a constraint.

The soundness and completeness proof of this transformation is almost identical to the soundness and completeness proof of Chapter 4, available at `https://arxiv.org/abs/2010.13707`, since it just replaces time constraints by the time and space constraints associated to the expression $d((x,y,z),(x',y',z'))$.

Example 28. *The time and space processes of Example 21 are transformed into the following untimed processes. We remove the "else nilP" branches for clarity.*

$$
\begin{aligned}
(\mathit{Verifier}) : & -(\mathit{Commit} \ @ \ A_1 : x_1, y_1, z_1, t_1' \rightarrow V_? : t_1 \uplus AS_1) \cdot \\
& \mathit{if}\ t_1 = t_1' + d((x_1,y_1,z_1),(x_?,y_?,z_?)) \wedge d((x_1,y_1,z_1),(x_?,y_?,z_?)) \geq 0\ \mathit{then} \\
& +(n(V_?, f_1) \ @ \ V_? : x_?, y_?, z_?, t_{2?} \rightarrow AS_{2?}) \cdot \\
& -((n(V_?, f_1) \oplus N_P) \ @ \ A_3 : x_3, y_3, z_3, t_3' \rightarrow V_? : t_3 \uplus AS_3) \cdot \\
& \mathit{if}\ t_3 = t_3' + d((x_3,y_3,z_3),(x_?,y_?,z_?)) \wedge d((x_3,y_3,z_3),(x_?,y_?,z_?)) \geq 0\ \mathit{then} \\
& \mathit{if}\ t_3 \dot{-} t_{2?} \leq 2 * d\ \mathit{then} \\
& -(S_P \ @ \ A_4 : x_4, y_4, z_4, t_4' \rightarrow V_? : t_4 \uplus AS_4) \cdot \\
& \mathit{if}\ t_4 = t_4' + d((x_4,y_4,z_4),(x_?,y_?,z_?)) \wedge d((x_4,y_4,z_4),(x_?,y_?,z_?)) \geq 0\ \mathit{then} \\
& \mathit{if}\ \mathit{open}(N_P, S_P, \mathit{Commit})\ \mathit{then} \\
& -(\mathit{sign}(P, n(V_?, f_1); N_P \oplus n(V_?, f_1)) \ @ \ A_5 : x_5, y_5, z_5, t_5' \rightarrow V_? : t_5 \uplus AS_5) \cdot \\
& \mathit{if}\ t_5 = t_5' + d((x_5,y_5,z_5),(x_?,y_?,z_?)) \wedge d((x_5,y_5,z_5),(x_?,y_?,z_?)) \geq 0\ \mathit{then} \\
& \mathit{nilP} \\
(\mathit{Prover}) : & +(\mathit{commit}(n(P_?, f_1), s(P_?, f_2)) @ P_? : x_?, y_?, z_?, t_{1?} \rightarrow AS_{1?}) \cdot \\
& -(V; N_V \ @ \ A_2 : x_2, y_2, z_2, t_2' \rightarrow V_? : t_2 \uplus AS_2) \cdot \\
& \mathit{if}\ t_2 = t_2' + d((x_2,y_2,z_2),(x_?,y_?,z_?)) \wedge d((x_2,y_2,z_2),(x_?,y_?,z_?)) \geq 0\ \mathit{then} \\
& +((N_V \oplus n(P_?, f_1)) @ P_? : x_?, y_?, z_?, t_{3?} \rightarrow AS_{3?}) \cdot \\
& +(s(P_?, f_2) @ P_? : x_?, y_?, z_?, t_{4?} \rightarrow AS_{4?}) \cdot \\
& +(\mathit{sign}(P_?, N_V; n(P_?, f_1) \oplus N_V) @ P_? : x_?, y_?, z_?, t_{5?} \rightarrow AS_{5?}))
\end{aligned}
$$

Example 29. *The time and space processes of Example 23 for the intruder are transformed into the following untimed processes. Note that we use the intruder identifier*

I associated to each role instead of a choice variable $I_?$.

$$
\begin{aligned}
(I.Conc) : & -(X @ A_1 : x_1, y_1, z_1, t_1 \rightarrow I : t_1' \uplus AS_1) \cdot \\
& \textit{if } t_1' = t_1 + d((x_1, y_1, z_1), (x_?, y_?, z_?)) \wedge d((x_1, y_1, z_1), (x_?, y_?, z_?)) \geq 0 \textit{ then} \\
& -(Y @ A_2 : x_2, y_2, z_2, t_2 \rightarrow I : t_2' \uplus AS_2) \cdot \\
& \textit{if } t_2' = t_2 + d((x_2, y_2, z_2), (x_?, y_?, z_?)) \wedge d((x_2, y_2, z_2), (x_?, y_?, z_?)) \geq 0 \textit{ then} \\
& +(X;Y @ I : x_?, y_?, z_?, t_{3?} \rightarrow AS_?) \\
(I.DeconcLeft) : & -(X;Y @ A_1 : x_1, y_1, z_1, t_1 \rightarrow I : t_1' \uplus AS_1) \cdot \\
& \textit{if } t_1' = t_1 + d((x_1, y_1, z_1), (x_?, y_?, z_?)) \wedge d((x_1, y_1, z_1), (x_?, y_?, z_?)) \geq 0 \textit{ then} \\
& +(X @ I : x_?, y_?, z_?, t_{2?} \rightarrow AS_?) \\
(I.Forward) : & -(X @ A_1 : x_1, y_1, z_1, t_1 \rightarrow I : t_1' \uplus AS_1) \cdot \\
& \textit{if } t_1' = t_1 + d((x_1, y_1, z_1), (x_?, y_?, z_?)) \wedge d((x_1, y_1, z_1), (x_?, y_?, z_?)) \geq 0 \textit{ then} \\
& +(X @ I : x_?, y_?, z_?, t_{2?} \rightarrow AS_?)
\end{aligned}
$$

Example 30. *The time and space processes of Example 22 are transformed into the following untimed processes. We remove the "else nilP" branches for clarity.*

$$
\begin{aligned}
(Be^i, x, y, z) : & -(t @ x_1, y_1, z_1, t_1' \rightarrow Be_? : t_1 \uplus AS_1) \cdot \\
& \textit{if } t_1 = t_1' + d((x_1, y_1, z_1), (x_?, y_?, z_?)) \wedge d((x_1, y_1, z_1), (x_?, y_?, z_?)) \geq 0 \textit{ then} \\
& +(((t \dot{-} t_1) \ ; \ x_? \ ; \ y_?) @ x_?, y_?, z_?, t_2 \rightarrow AS_{2?}) \cdot \\
& -((ok @ x_3, y_3, z_3, t_3' \rightarrow Be_? : t_3 \uplus AS_3) \cdot \\
& \textit{if } t_3 = t_3' + d((x_3, y_3, z_3), (x_?, y_?, z_?)) \wedge d((x_3, y_3, z_3), (x_?, y_?, z_?)) \geq 0 \textit{ then} \\
& \textit{nilP} \\
(Ba, x, y, z) : & -((t_1' \ ; \ x_1' \ ; \ y_1') @ x_1, y_1, z_1, t_1' \rightarrow Ba_? : t_1 \uplus AS_1) \cdot \\
& \textit{if } t_1 = t_1' + d((x_1, y_1, z_1), (x_?, y_?, z_?)) \wedge d((x_1, y_1, z_1), (x_?, y_?, z_?)) \geq 0 \textit{ then} \\
& -((t_2' \ ; \ x_2' \ ; \ y_2') @ x_2, y_2, z_2, t_2' \rightarrow Ba_? : t_2 \uplus AS_2) \cdot \\
& \textit{if } t_2 = t_2' + d((x_2, y_2, z_2), (x_?, y_?, z_?)) \wedge d((x_2, y_2, z_2), (x_?, y_?, z_?)) \geq 0 \textit{ then} \\
& -((t_3' \ ; \ x_3' \ ; \ y_3') @ x_3, y_3, z_3, t_3' \rightarrow Ba_? : t_3 \uplus AS_3) \cdot \\
& \textit{if } t_3 = t_3' + d((x_3, y_3, z_3), (x_?, y_?, z_?)) \wedge d((x_3, y_3, z_3), (x_?, y_?, z_?)) \geq 0 \textit{ then} \\
& -((t_4' \ ; \ x_4' \ ; \ y_4') @ x_4, y_4, z_4, t_4' \rightarrow Ba_? : t_4 \uplus AS_4) \cdot \\
& \textit{if } t_4 = t_4' + d((x_4, y_4, z_4), (x_?, y_?, z_?)) \wedge d((x_4, y_4, z_4), (x_?, y_?, z_?)) \geq 0 \textit{ then} \\
& \textit{if } \exists dx, dy : (t_1')^2 = (dx - x_1')^2 + (dy - y_1')^2 \wedge \\
& \qquad\qquad (t_2')^2 = (dx - x_2')^2 + (dy - y_2')^2 \wedge \\
& \qquad\qquad (t_3')^2 = (dx - x_3')^2 + (dy - y_3')^2 \wedge \\
& \qquad\qquad (t_4')^2 = (dx - x_4')^2 + (dy - y_4')^2 \\
& \textit{then } +(ok @ x_?, y_?, z_?, t_5 \rightarrow AS_{5?})
\end{aligned}
$$

Once a time and space process is transformed into an untimed process with time and location variables and time and locations constraints using the

notation of Maude-NPA, we can easily adapt the soundness and completeness proof of Chapter 4, which relies on both a soundness and completeness proof from the Maude-NPA process notation into Maude-NPA forward rewriting semantics and on a soundness and completeness proof from Maude-NPA forward rewriting semantics into Maude-NPA backwards symbolic semantics, see [114, 115]. Since the Maude-NPA backwards symbolic semantics already considers constraints in a very general setting [57], we only need to perform the additional satisfiability check for real polynomial arithmetic.

5.5 Timed Process Algebra into Strands in Maude-NPA

This section is provided to help in understanding the experimental work. Although Maude-NPA accepts protocol specifications in either the process algebra notation or the strand space notation, its output is given in the stand space notation. Thus, in order to make our experiments easier to understand, we describe the translation from untimed processes with time and space constraints into untimed strands with time and location variables and time and space constraints. This translation is also sound and complete, as it replicates the transformation of [114, 115].

Strands [62] are used in Maude-NPA to represent both the actions of honest principals (with a strand specified for each protocol role) and those of an intruder (with a strand for each action an intruder is able to perform on messages). In Maude-NPA strands evolve over time. The symbol | is used to divide past and future. That is, given a strand [$msg_1^{\pm}$, ..., $msg_i^{\pm}$ | $msg_{i+1}^{\pm}$, ..., $msg_k^{\pm}$], messages $msg_1^{\pm}, \ldots, msg_i^{\pm}$ are the *past messages*, and messages $msg_{i+1}^{\pm}, \ldots, msg_k^{\pm}$ are the *future messages* ($msg_{i+1}^{\pm}$ is the immediate future message). Constraints can be also inserted into strands. A strand $[msg_1^{\pm}, \ldots, msg_k^{\pm}]$ is shorthand for $[nil \mid msg_1^{\pm}, \ldots, msg_k^{\pm}, nil]$. An *initial state* is a state where the bar is at the beginning for all strands in the state, and the network has no possible intruder fact of the form $m \in \mathscr{I}$. A *final state* is a state where the bar is at the end for all strands in the state and there is no negative intruder fact of the form $m \notin \mathscr{I}$.

In the following, we illustrate how the untimed process algebra can be transformed into strands specifications of Maude-NPA for our two running examples. We simply replaced · by comma, and each if-then-else by its boolean constraint.

Example 31. *The untimed processes of Example 28 are transformed into the following strands.*

$$
\begin{aligned}
(\mathit{Verifier}) : [&-(\mathit{Commit}\ @\ A_1 : x_1, y_1, z_1, t_1' \rightarrow V_? : t_1 \uplus AS_1),\\
&(t_1 = t_1' + d((x_1, y_1, z_1), (x_?, y_?, z_?)) \wedge d((x_1, y_1, z_1), (x_?, y_?, z_?)) \geq 0),\\
&+(n(V_?, f_1)\ @\ V_? : x_?, y_?, z_?, t_{2?} \rightarrow AS_{2?}),\\
&-((n(V_?, f_1) \oplus N_P)\ @\ A_3 : x_3, y_3, z_3, t_3' \rightarrow V_? : t_3 \uplus AS_3),\\
&(t_3 = t_3' + d((x_3, y_3, z_3), (x_?, y_?, z_?)) \wedge d((x_3, y_3, z_3), (x_?, y_?, z_?)) \geq 0),\\
&(t_3 \dot{-} t_{2?} \leq 2 * d),\\
&-(S_P\ @\ A_4 : x_4, y_4, z_4, t_4' \rightarrow V_? : t_4 \uplus AS_4),\\
&(t_4 = t_4' + d((x_4, y_4, z_4), (x_?, y_?, z_?)) \wedge d((x_4, y_4, z_4), (x_?, y_?, z_?)) \geq 0),\\
&\mathit{open}(N_P, S_P, \mathit{Commit}),\\
&-(\mathit{sign}(P, n(V_?, f_1); N_P \oplus n(V_?, f_1))\ @\ A_5 : x_5, y_5, z_5, t_5' \rightarrow V_? : t_5 \uplus AS_5),\\
&(t_5 = t_5' + d((x_5, y_5, z_5), (x_?, y_?, z_?)) \wedge d((x_5, y_5, z_5), (x_?, y_?, z_?)) \geq 0)]
\end{aligned}
$$

$$
\begin{aligned}
(\mathit{Prover}) : [&+(\mathit{commit}(n(P_?, f_1), s(P_?, f_2))@P_? : x_?, y_?, z_?, t_{1?} \rightarrow AS_{1?}),\\
&-(V; N_V @\ A_2 : x_2, y_2, z_2, t_2' \rightarrow V_? : t_2 \uplus AS_2),\\
&(t_2 = t_2' + d((x_2, y_2, z_2), (x_?, y_?, z_?)) \wedge d((x_2, y_2, z_2), (x_?, y_?, z_?)) \geq 0),\\
&+((N_V \oplus n(P_?, f_1))@P_? : x_?, y_?, z_?, t_{3?} \rightarrow AS_{3?}),\\
&+(s(P_?, f_2)@P_? : x_?, y_?, z_?, t_{4?} \rightarrow AS_{4?}),\\
&+(\mathit{sign}(P_?, N_V; n(P_?, f_1) \oplus N_V)@P_? : x_?, y_?, z_?, t_{5?} \rightarrow AS_{5?}))]
\end{aligned}
$$

Example 32. *The untimed processes of Example 30 are transformed into the following strands.*

$$
\begin{aligned}
(\mathit{Be}^i, x, y, z) : [&-(t@x_1, y_1, z_1, t_1' \rightarrow \mathit{Be}_? : t_1 \uplus AS_1),\\
&(t_1 = t_1' + d((x_1, y_1, z_1), (x_?, y_?, z_?)) \wedge d((x_1, y_1, z_1), (x_?, y_?, z_?)) \geq 0),\\
&+(((t \dot{-} t_1)\ ;\ x_?\ ;\ y_?)@x_?, y_?, z_?, t_2 \rightarrow AS_{2?}),\\
&-((\mathit{ok}@x_3, y_3, z_3, t_3' \rightarrow \mathit{Be}_? : t_3 \uplus AS_3),\\
&(t_3 = t_3' + d((x_3, y_3, z_3), (x_?, y_?, z_?)) \wedge d((x_3, y_3, z_3), (x_?, y_?, z_?)) \geq 0)]
\end{aligned}
$$

$$
\begin{aligned}
(Ba,x,y,z) : [&-((t_1' \ ; \ x_1' \ ; \ y_1')@x_1,y_1,z_1,t_1' \rightarrow Ba_? : t_1 \uplus AS_1),\\
&(t_1 = t_1' + d((x_1,y_1,z_1),(x_?,y_?,z_?)) \wedge d((x_1,y_1,z_1),(x_?,y_?,z_?)) \geq 0),\\
&-((t_2' \ ; \ x_2' \ ; \ y_2')@x_2,y_2,z_2,t_2' \rightarrow Ba_? : t_2 \uplus AS_2),\\
&(t_2 = t_2' + d((x_2,y_2,z_2),(x_?,y_?,z_?)) \wedge d((x_2,y_2,z_2),(x_?,y_?,z_?)) \geq 0),\\
&-((t_3' \ ; \ x_3' \ ; \ y_3')@x_3,y_3,z_3,t_3' \rightarrow Ba_? : t_3 \uplus AS_3),\\
&(t_3 = t_3' + d((x_3,y_3,z_3),(x_?,y_?,z_?)) \wedge d((x_3,y_3,z_3),(x_?,y_?,z_?)) \geq 0),\\
&-((t_4' \ ; \ x_4' \ ; \ y_4')@x_4,y_4,z_4,t_4' \rightarrow Ba_? : t_4 \uplus AS_4),\\
&(t_4 = t_4' + d((x_4,y_4,z_4),(x_?,y_?,z_?)) \wedge d((x_4,y_4,z_4),(x_?,y_?,z_?)) \geq 0),\\
&\begin{pmatrix}(t_1')^2 = (dx - x_1')^2 + (dy - y_1')^2 \wedge\\ (t_2')^2 = (dx - x_2')^2 + (dy - y_2')^2 \wedge\\ (t_3')^2 = (dx - x_3')^2 + (dy - y_3')^2 \wedge\\ (t_4')^2 = (dx - x_4')^2 + (dy - y_4')^2\end{pmatrix}\\
&+(ok@x_?,y_?,z_?,t_5 \rightarrow AS_{5?})]
\end{aligned}
$$

We specify the desired security properties in terms of *attack patterns* including logical variables, which describe the insecure states that Maude-NPA is trying to prove unreachable. The specifications, outputs, and a modified version of Maude-NPA are available at `http://personales.upv.es/sanesro/guttman2021`. Specifically, the tool attempts to find a *backwards narrowing sequence* path from the attack pattern to an initial state until it can no longer form any backwards narrowing steps, at which point it terminates. If it has not found an initial state, the attack pattern is judged *unreachable*.

The following examples show how a classic mafia fraud attack for the Brands-Chaum protocol can be specified in Maude-NPA's strand notation. Note that Maude-NPA uses symbol `===` for equality on the reals, `+=+` for addition on the reals, `*=*` for multiplication on the reals, and `-=-` for subtraction on the reals. Extra time and space constraints are included in an `smt` section. In general, Maude-NPA requires an SMT solver that supports checking quadratic constraints over the reals, such as Yices [5], Z3 [8], or Mathematica [4].

Example 33. *Following the strand specification of the Brands-Chaum protocol given in Example 31, the mafia attack of Example 19 is given as the following attack pattern. We consider one prover* `p`*, one verifier* `v`*, and one intruder* `i` *at fixed locations* (px,py,pz)*,* (vx,vy,vz) *and* (ix,iy,iz)*, respectively. Brands-Chaum is secure against the mafia fraud attack and no initial state is found in the backwards search.*

```
eq ATTACK-STATE(1) --- Mafia fraud
```

```
= :: r :: ---Alice --- Verifier
[ nil,
 -(commit(n(b,r1),s(b,r2)) @ i : ix,iy,iz,t1 -> a : t2),
 ((t2 === t1 +=+ dai) and (dai > 0/1) and
 ((dai *=* dai) === (((ix -=- ax) *=* (ix -=- ax)) +=+ ((iy -=- ay) *=* (iy -=- ay)))
                                                  +=+ ((iz -=- az) *=* (iz -=- az)))),
 +(n(a,r) @ a : ax,ay,az,t2 -> i : t2'''),
 -(n(a,r) * n(b,r1) @ i : ix,iy,iz,t3 -> a : t4),
 ((t4 === (t3 +=+ dai)) and (dai > 0/1) and
 (((t4 -=- t2) <= (2/1 *=* d)) and (d > 0/1))  | nil ]
&
:: r1,r2 :: ---Bob  --- Prover
[ nil,
 +(commit(n(b,r1),s(b,r2)) @ b : bx,by,bz,t1' -> i : t1''),
 -(n(a,r) @ i : ix,iy,iz,t2'' -> b : t3'),
 ((t3' === (t2'' +=+ dbi)) and (dbi > 0/1) and
 ((dbi *=* dbi) === ((((ix -=- bx) *=* (ix -=- bx)) +=+ ((iy -=- by) *=* (iy -=- by)))
                                                   +=+ ((iz -=- bz) *=* (iz -=- bz))))),
 +(n(a,r) * n(b,r1) @ b : bx,by,bz,t3' -> i : t3'') | nil ]
 || smt(((dai +=+ dbi) > d) and (dbi > 0/1) and (dab > 0/1) and (dai > 0/1) and
         ((dab *=* dab) === ((((ax -=- bx) *=* (ax -=- bx)) +=+
         ((ay -=- by) *=* (ay -=- by))) +=+ ((az -=- bz) *=* (az -=- bz)))))
```

Example 34. *Continuing Example 33, the hijacking attack of Example 19 is given as the following attack pattern. And the backwards search of Maude-NPA from this attack pattern does find an initial state.*

```
eq ATTACK-STATE(2) --- Hijacking
= :: r :: --- Alice --- Verifier
[ nil,
 -(commit(n(b,r1),s(b,r2)) @ b : bx,by,bz,t1 -> a : t2),
 ((t2 === t1 +=+ dab) and (dab > 0/1) and
 ((dab *=* dab) === (((ax -=- bx) *=* (ax -=- bx)) +=+ ((ay -=- by) *=* (ay -=- by)))
                                                  +=+ ((az -=- bz) *=* (az -=- bz)))),
 +(n(a,r) @ a : ax,ay,az,t2 -> b : t3 # i : t2''),
 -(n(a,r) * n(b,r1) @ b : bx,by,bz,t3 -> a : t4 # i : t4''),
 ((t4 === t3 +=+ dab)),
 ((t4 -=- t2) <= (2/1 *=* d)),
 -(s(b,r2) @ b : bx,by,bz,t5 -> a : t6),
 ((t6 === t5 +=+ dab)),
 -(sign(i,(n(a,r) * n(b,r1)) ; n(a,r)) @ i : ix,iy,iz,t7 -> a : t8),
 ((t8 === (t7 +=+ dai)) and (dai > 0/1) and
 ((dai *=* dai) === (((ax -=- ix) *=* (ax -=- ix)) +=+ ((ay -=- iy) *=* (ay -=- iy)))
                                                  +=+ ((az -=- iz) *=* (az -=- iz))))
  | nil ]
&
:: r1,r2 :: ---Bob  --- Prover
[ nil,
 +(commit(n(b,r1),s(b,r2)) @ b : bx,by,bz,t1 -> a : t2),
 -(n(a,r) @ a : ax,ay,az,t2 -> b : t3 # i : t3''),
 ((t3 === (t2 +=+ dab)) and (dab > 0/1) and
 ((dab *=* dab) === (((ax -=- bx) *=* (ax -=- bx)) +=+ ((ay -=- by) *=* (ay -=- by)))
```

```
                                                    +=+ ((az -=- bz) *=* (az -=- bz)))),
 +(n(a,r) * n(b,r1) @ b : bx,by,bz,t3 -> a : t4 # i : t4''),
 +(s(b,r2) @ b : bx,by,bz,t5 -> a : t6) | nil]
|| smt( (dai > d) and (dab <= d))
```

Example 35. *Following the strand specification of the beacons protocol given in Example 32, we can give a very general attack pattern.*

```
eq ATTACK-STATE(0) =
:: r :: --- Intruder
[ nil, +(t @ i : x1,y1,z1,t0 -> Be1 : t1 # Be2 : t2 # Be3 : t3 # Be4 : t4) | nil ]
&
:: r1 :: --- Beacon 1
[ nil ,
 -(t @ i : x1,y1,z1,t0 -> Be1 : t1 # Be2 : t2 # Be3 : t3 # Be4 : t4),
 ((t1 === t0 +=+ dbe1) and (dbe1 > 0/1) and
 ((dbe1 *=* dbe1) === (((be1x -=- x1) *=* (be1x -=- x1))
        +=+ ((be1y -=- y1) *=* (be1y -=- y1))) +=+ ((be1z -=- z1) *=* (be1z -=- z1)))),
 +((t1 -=- t) ; be1x ; be1y @ Be1 : be1x,be1y,be1z,t1 -> Ba : t1'),
 -(ok @ Ba : bax,bay,baz,t5 -> Be1 : t1'' # Be2 : t2'' # Be3 : t3'' # Be4 : t4''),
 ((t1'' === t5 +=+ dbabe1) and (dbabe1 > 0/1) and
 ((dbabe1 *=* dbabe1) === (((be1x -=- bax) *=* (be1x -=- bax))
    +=+ ((be1y -=- bay) *=* (be1y -=- bay))) +=+ ((be1z -=- baz) *=* (be1z -=- baz))))
    | nil]
  &
:: r2 :: --- Beacon 2
[ nil ,
 -(t @ i : x1,y1,z1,t0 -> Be1 : t1 # Be2 : t2 # Be3 : t3 # Be4 : t4),
 ((t2 === t0 +=+ dbe2) and (dbe2 > 0/1) and
 ((dbe2 *=* dbe2) === (((be2x -=- x1) *=* (be2x -=- x1))
     +=+ ((be2y -=- y1) *=* (be2y -=- y1))) +=+ ((be2z -=- z1) *=* (be2z -=- z1)))),
 +((t2 -=- t) ; be2x ; be2y @ Be2 : be2x,be2y,be2z,t2 -> Ba : t2'),
 -(ok @ Ba : bax,bay,baz,t5 -> Be1 : t1'' # Be2 : t2'' # Be3 : t3'' # Be4 : t4''),
 ((t2'' === t5 +=+ dbabe2) and (dbabe2 > 0/1) and
 ((dbabe2 *=* dbabe2) === (((be2x -=- bax) *=* (be2x -=- bax))
    +=+ ((be2y -=- bay) *=* (be2y -=- bay))) +=+ ((be2z -=- baz) *=* (be2z -=- baz))))
    | nil]
&
:: r3 :: --- Beacon 3
[ nil ,
 -(t @ i : x1,y1,z1,t0 -> Be1 : t1 # Be2 : t2 # Be3 : t3 # Be4 : t4),
 ((t3 === t0 +=+ dbe3) and (dbe3 > 0/1) and
 ((dbe3 *=* dbe3) === (((be3x -=- x1) *=* (be3x -=- x1))
     +=+ ((be3y -=- y1) *=* (be3y -=- y1))) +=+ ((be3z -=- z1) *=* (be3z -=- z1)))),
 +((t3 -=- t) ; be3x ; be3y @ Be3 : be3x,be3y,be3z,t3 -> Ba : t3'),
 -(ok @ Ba : bax,bay,baz,t5 -> Be1 : t1'' # Be2 : t2'' # Be3 : t3'' # Be4 : t4''),
 ((t3'' === t5 +=+ dbabe3) and (dbabe3 > 0/1) and
 ((dbabe3 *=* dbabe3) === (((be3x -=- bax) *=* (be3x -=- bax))
  +=+ ((be3y -=- bay) *=* (be3y -=- bay))) +=+ ((be3z -=- baz) *=* (be3z -=- baz))))
  | nil]
&
:: r4 :: --- Beacon 4
```

```
[ nil ,
 -(t @ i : x1,y1,z1,t0 -> Be1 : t1 # Be2 : t2 # Be3 : t3 # Be4 : t4),
 ((t4 === t0 +=+ dbe4) and (dbe4 > 0/1) and
 ((dbe4 *=* dbe4) === (((be4x -=- x1) *=* (be4x -=- x1))
     +=+ ((be4y -=- y1) *=* (be4y -=- y1))) +=+ ((be4z -=- z1) *=* (be4z -=- z1)))),
 +((t4 -=- t) ; be4x ; be4y @ Be4 : be4x,be4y,be4z,t4 -> Ba : t4'),
 -(ok(r') @ Ba : bax,bay,baz,t5 -> Be1 : t1'' # Be2 : t2'' # Be3 : t3'' # Be4 : t4''),
 ((t4'' === t5 +=+ dbabe4) and (dbabe4 > 0/1) and
 ((dbabe4 *=* dbabe4) === (((be4x -=- bax) *=* (be4x -=- bax))
     +=+ ((be4y -=- bay) *=* (be4y -=- bay))) +=+ ((be4z -=- baz) *=* (be4z -=- baz))))
     | nil]
&
:: r' :: --- Base Station
[ nil ,
 -((t1 -=- t) ; be1x ; be1y @ Be1 : be1x,be1y,be1z,t1 -> Ba : t1'),
 ((t1' === t1 +=+ dbabe1) and (dbabe1 > 0/1) and
 ((dbabe1 *=* dbabe1) === (((be1x -=- bax) *=* (be1x -=- bax))
     +=+ ((be1y -=- bay) *=* (be1y -=- bay))) +=+ ((be1z -=- baz) *=* (be1z -=- baz)))),
 -((t2 -=- t) ; be2x ; be2y @ Be2 : be2x,be2y,be2z,t2 -> Ba : t2'),
 ((t2' === t2 +=+ dbabe2) and (dbabe2 > 0/1) and
 ((dbabe2 *=* dbabe2) === (((be2x -=- bax) *=* (be2x -=- bax))
     +=+ ((be2y -=- bay) *=* (be2y -=- bay))) +=+ ((be2z -=- baz) *=* (be2z -=- baz)))),
 -((t3 -=- t) ; be3x ; be3y @ Be3 : be3x,be3y,be3z,t3 -> Ba : t3'),
 ((t3' === t3 +=+ dbabe3) and (dbabe3 > 0/1) and
 ((dbabe3 *=* dbabe3) === (((be3x -=- bax) *=* (be3x -=- bax))
     +=+ ((be3y -=- bay) *=* (be3y -=- bay))) +=+ ((be3z -=- baz) *=* (be3z -=- baz)))),
 -((t4 -=- t) ; be4x ; be4y @ Be4 : be4x,be4y,be4z,t4 -> Ba : t4'),
 ((t4' === t4 +=+ dbabe4) and (dbabe4 > 0/1) and
 ((dbabe4 *=* dbabe4) === (((be4x -=- bax) *=* (be4x -=- bax))
     +=+ ((be4y -=- bay) *=* (be4y -=- bay))) +=+ ((be4z -=- baz) *=* (be4z -=- baz)))),
 (((t1 -=- t) *=* (t1 -=- t)) === (((dx -=- be1x) *=* (dx -=- be1x))
    +=+ ((dy -=- be1y) *=* (dy -=- be1y))) and
  ((t2 -=- t) *=* (t2 -=- t)) === (((dx -=- be2x) *=* (dx -=- be2x))
    +=+ ((dy -=- be2y) *=* (dy -=- be2y))) and
  ((t3 -=- t) *=* (t3 -=- t)) === (((dx -=- be3x) *=* (dx -=- be3x))
    +=+ ((dy -=- be3y) *=* (dy -=- be3y))) and
  ((t4 -=- t) *=* (t4 -=- t)) === (((dx -=- be4x) *=* (dx -=- be4x))
    +=+ ((dy -=- be4y) *=* (dy -=- be4y)))),
 +(ok @ Ba : bax,bay,baz,t5 -> Be1 : t1'' # Be2 : t2'' # Be3 : t3'' # Be4 : t4''),
 (t5 >= t1' and t5 >= t2' and t5 >= t3' and t5 >= t4') | nil]
   || smt((t =/== t0))
```

The insecure configuration of Figure 5.4 is now obtained by just adding extra constraints to the attack pattern: (i) fixing concrete locations for the beacons in a hyperbola, (ii) adding the distances from the malicious device to the beacons, (iii) adding the distances inferred by the beacons from the malicious device, and (iv) adjusting the sent timestamp to differ from the actual sending time in the appropriate amount to fake the base station. And the backwards search of Maude-NPA from this attack pattern does find an initial state.

```
smt( --- hyperbola with a^2 = 4, b^2 = 5, c^2 = 9
```

```
(t > t0) and (t0 === 0/1) and (z1 === 0/1) and
(be1z === 0/1) and (be2z === 0/1) and (be3z === 0/1) and
(be4z === 0/1) and (baz === 0/1) and
((be1x === 3/1) and (be1y === 5/2)) and
((be2x === 3/1) and (be2y === -(5/2))) and
((be3x === 4/1) and ((be3y *=* be3y) === 60/4) and (be3y > 0/1)) and
((be4x === 4/1) and ((be4y *=* be4y) === 60/4) and (be4y < 0/1)) and
(x1 === -(3/1)) and (dx === 3/1) and (dy === y1) and (y1 === 0/1))
```

The secure configuration of Figure 5.5 is now obtained by just adding extra constraints to the attack pattern: (i) fixing concrete locations for the beacons in a rectangle for a parametric height and width, and (ii) asking whether the timestamp is different from the sending time.

```
smt( (t =/== t0) and (t >= 0/1) and z1 === 0/1 and baz === 0/1 and
be1z === 0/1 and be2z === 0/1 and be3z === 0/1 and be4z === 0/1 and
(h > 0/1) and (v > 0/1) and (be1x === 0/1) and (be1y === 0/1) and
(be2x === be1x) and (be2y === be1y +=+ v) and
(be3x === be1x +=+ h) and (be3y === be1y) and
(be4x === be1x +=+ h) and (be4y === be1y +=+ v))
```

Our analysis of this protocol uncovered some interesting challenges that would need to be addressed in future research. When we gave the constraints to the SMT solvers, including Yices [5] and Z3 [8], which support non-linear real arithmetic, none of them were able to prove that they were unsatisfiable. It was not until we simplified them by hand by using Gaussian elimination on the matrix defined by the coefficients of the constraints, producing the set of constraints given below, that we were able to get one solver, Mathematica [4], to prove unsatisfiability. This suggest that more research is needed on heuristics for preprocessing the types of constraints that arise from reasoning about time and space protocols so that they can be handled by available SMT solvers.

```
In[8]:= Solve[w > 0 && h > 0 && d != 0 && d1 > 0 && d2 > 0 && d3 > 0 && d4 > 0 && (d1-d) > 0 && (d2 - d) > 0 &&
          (d3 - d) > 0 && (d4 - d) > 0 && (((d1 == d2) && (d2 == d4)) || ((d1 == d3) && d2 == d4)) &&
          x == (1 / 2) * w && y == ((((d1 ^ 2)-(d3 ^ 2)) + (w ^ 2)) / (2 * w)) && x ^ 2 + y ^ 2 == d1 ^ 2 &&
          (((2 * d * (d3 - d1)) / h) * y) ^ 2 + (d * ((d3 - d1) / h)) ^ 2 + (2 * d1 * d) - d ^ 2 == 0, {d}, Reals]

Out[8]= { }
```

5.6 Conclusions

We have extended our previous paper with a time model for protocols using time constraints to a time and space model for protocol analysis based on

timing and space constraints. We have also extended our previous prototype of Maude-NPA handling protocols with time to handle time and space by taking advantage of Maude's support of SMT solvers, and Maude-NPA's support of constraint handling. We have used the Brands and Chaum protocol to illustrate how this extension is natural and smoothly subsumes our previous time-only framework, and a secure localization protocol with complex location and time constraints. This approach should be applicable to other tools that support constraint handling. There are several ways this work can be extended, as suggested within the paper. And there are many interesting protocols that can be tested with this time and space model, for example protocols using the Message Time Of Arrival Codes (MTACs) of [79].

CHAPTER 6

Variant-based Equational Unification under Constructor Symbols

Damián Aparicio-Sánchez[1], Santiago Escobar[1], Julia Sapiña[1]

[1] VRAIN, Universitat Politècnica de València,
Spain
{daapsnc,sescobar,jsapina}@upv.es

Abstract Equational unification of two terms consists of finding a substitution that, when applied to both terms, makes them equal modulo some equational properties. A narrowing-based equational unification algorithm relying on the concept of the *variants* of a term is available in the most recent version of Maude, version 3.0, which provides quite sophisticated unification features. A variant of a term t is a pair consisting of a substitution σ and the canonical form of $t\sigma$. Variant-based unification is decidable when the equational theory satisfies the *finite variant property*. However, this unification procedure does not take into account constructor symbols and, thus, may compute many more unifiers than the necessary or may not be able to stop immediately. In this paper, we integrate the notion of constructor symbol into the variant-based unification algorithm. Our experiments on positive and negative unification problems show an impressive speedup.

This work has been partially supported by the EU (FEDER) and the Spanish MCIU under grant RTI2018-094403-B-C32, by the Spanish Generalitat Valenciana under grants PROMETEO/2019/098 and APOSTD/2019/127, and by the US Air Force Office of Scientific Research under award number FA9550-17-1-0286.

6.1 Introduction

Equational unification of two terms is of special relevance to many areas in computer science, including logic programming, and consists of finding a substitution that, when applied to both terms, makes them equal modulo some equational properties. Several algorithms have been developed in the literature for specific equational theories, such as associative-commutative symbols, exclusive-or, Diffie-Hellman, or Abelian Groups (see [15]). Narrowing was proved to be complete for unification [68] and several cases have been studied where narrowing provides a decidable unification algorithm [11, 12]. A narrowing-based equational unification algorithm relying on the concept of the *variants* of a term [38] has been developed in [60] and it is available in the most recent version of Maude, version 3.0, which provides quite sophisticated unification features [35, 46].

Several tools and techniques rely on Maude's advanced unification capabilities, such as termination [47] and local confluence and coherence [48,49] proofs, narrowing-based theorem proving [104] or testing [103], and *logical model checking* [16, 58]. The area of cryptographic protocol analysis has also benefited from advanced unification algorithms: Maude-NPA [55], Tamarin [45] and AKISS [17] rely on the different unification features of Maude. Furthermore, numerous decision procedures for formula satisfiability modulo equational theories also rely on unification, either based on narrowing [111] or by using variant generation in finite variant theories [91].

Constructor symbols are extensively used in computer science: for representing data instead of functions, for manipulating programs as data, or for reasoning in complex semantic structures. In an equational theory, constructors can be characterized in the "no junk, no confusion" style of Goguen and Burstall [28], providing the mathematical semantics of the equational theory as the *initial algebra* of a Maude functional module, which corresponds to the least Herbrand model in logic programming (see [35]). However, this more general notion of constructor differs from the "logic" notion of a functor and the "functional" notion of a symbol not appearing in the root position of the left-hand side of any equation. The notion of a constructor symbol has not yet been integrated into the variant-based equational unification procedure of Maude and, thus, it may compute many more unifiers than the necessary or it may not be able to stop immediately. In this paper, we integrate the notion of constructor symbol into the variant-based unification algorithm with an impressive speedup.

After some preliminaries in Section 6.2, we recall variant-based unification in Section 6.3. In Section 6.4, we define our new unification algorithm that reduces the total execution time. Our experiments in Section 6.5 show that this improved unification algorithm works well in practice. We conclude in Section 6.6.

6.2 Preliminaries

We follow the classical notation and terminology from [18] for term rewriting, from [15] for unification, and from [86] for rewriting logic and order-sorted notions.

We assume an order-sorted signature $\Sigma = (S, \leq, \Sigma)$ with a poset of sorts $(S, \leq)$. The poset $(\mathsf{S}, \leq)$ of sorts for Σ is partitioned into equivalence classes, called *connected components*, by the equivalence relation $(\leq \cup \geq)^+$. We assume that each connected component $[\mathsf{s}]$ has a *top element* under $\leq$, denoted $\top_{[\mathsf{s}]}$ and called the *top sort* of $[\mathsf{s}]$. This involves no real loss of generality, since if $[\mathsf{s}]$ lacks a top sort, it can be easily added. We also assume an S-sorted family $\mathcal{X} = \{\mathcal{X}_\mathsf{s}\}_{\mathsf{s}\in\mathsf{S}}$ of disjoint variable sets with each $\mathcal{X}_\mathsf{s}$ countably infinite. $\mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ is the set of terms of sort s, and $\mathcal{T}_{\Sigma,\mathsf{s}}$ is the set of ground terms of sort s. We write $\mathcal{T}_\Sigma(\mathcal{X})$ and $\mathcal{T}_\Sigma$ for the corresponding order-sorted term algebras. Given a term t, $\mathcal{V}ar(t)$ denotes the set of variables in t.

Positions are represented by sequences of natural numbers denoting an access path in the term when viewed as a tree. The top or root position is denoted by the empty sequence Λ. We define the relation $p \leq q$ between positions as $p \leq p$ for any p; and $p \leq p.q$ for any p and q. Given $U \subseteq \Sigma \cup \mathcal{X}$, $Pos_U(t)$ denotes the set of positions of a term t that are rooted by symbols or variables in U. The set of positions of a term t is written $Pos(t)$, and the set of non-variable positions $Pos_\Sigma(t)$. The subterm of t at position p is $t|_p$ and $t[u]_p$ is the term t where $t|_p$ is replaced by u.

A *substitution* $\sigma \in \mathcal{S}ubst(\Sigma, \mathcal{X})$ is a sorted mapping from a finite subset of $\mathcal{X}$ to $\mathcal{T}_\Sigma(\mathcal{X})$. Substitutions are written as $\sigma = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$ where the domain of σ is $Dom(\sigma) = \{X_1, \ldots, X_n\}$ and the set of variables introduced by terms $t_1, \ldots, t_n$ is written $Ran(\sigma)$. The identity substitution is *id*. Substitutions are homomorphically extended to $\mathcal{T}_\Sigma(\mathcal{X})$. The application of a substitution σ to a term t is denoted by $t\sigma$ or $\sigma(t)$. For simplicity, we assume that every substitution is idempotent, i.e., σ satisfies $Dom(\sigma) \cap Ran(\sigma) = \emptyset$. The restriction of σ to a set of variables V is $\sigma|_V$, i.e., $\forall x \in V$, $\sigma|_V(x) = \sigma(x)$ and $\forall x \notin V$,

$\sigma|_V(x) = x$. Composition of two substitutions σ and σ' is denoted by $\sigma \circ \sigma'$. Combination of two substitutions σ and σ' such that $Dom(\sigma) \cap Dom(\sigma') = \emptyset$ is denoted by $\sigma \cup \sigma'$. We call a substitution σ a variable *renaming* if there is another substitution σ^{-1} such that $(\sigma\sigma^{-1})|_{Dom(\sigma)} = id$.

A Σ-*equation* is an unoriented pair $t = t'$, where $t, t' \in \mathscr{T}_\Sigma(\mathscr{X})_\mathsf{s}$ for some sort $\mathsf{s} \in \mathsf{S}$. An *equational theory* (Σ, E) is a pair with Σ an order-sorted signature and E a set of Σ-equations. Given Σ and a set E of Σ-equations, order-sorted equational logic induces a congruence relation $=_E$ on terms $t, t' \in \mathscr{T}_\Sigma(\mathscr{X})$ (see [87]). We say $\sigma_1 =_E \sigma_2$ iff $\sigma_1(x) =_E \sigma_2(x)$ for any variable x. Throughout this paper we assume that $\mathscr{T}_{\Sigma,\mathsf{s}} \neq \emptyset$ for every sort s, because this affords a simpler deduction system. An equational theory (Σ, E) is *regular* if for each $t = t'$ in E, we have $\mathscr{V}ar(t) = \mathscr{V}ar(t')$. An equational theory (Σ, E) is *linear* if for each $t = t'$ in E, each variable occurs only once in t and in t'. An equational theory (Σ, E) is *sort-preserving* if for each $t = t'$ in E, each sort s, and each substitution σ, we have $t\sigma \in \mathscr{T}_\Sigma(\mathscr{X})_\mathsf{s}$ iff $t'\sigma \in \mathscr{T}_\Sigma(\mathscr{X})_\mathsf{s}$. An equational theory (Σ, E) is *defined using top sorts* if for each equation $t = t'$ in E, all variables in $\mathscr{V}ar(t)$ and $\mathscr{V}ar(t')$ have a top sort. Given two terms t and t', we say t is more general than t', denoted as $t \sqsupseteq_E t'$, if there is a substitution η such that $t\eta =_E t'$. Similarly, given two substitutions σ and ρ, we say σ is more general than ρ for a set W of variables, denoted as $\sigma|_W \sqsupseteq_E \rho|_W$, if there is a substitution η such that $(\sigma \circ \eta)|_W =_E \rho|_W$. The $\sqsupseteq_E$ relation induces an equivalence relation $\simeq_E$, i.e., $t \simeq_E t'$ iff $t \sqsupseteq_E t'$ and $t \sqsubseteq_E t'$.

An E-*unifier* for a Σ-equation $t = t'$ is a substitution σ such that $t\sigma =_E t'\sigma$. For $\mathscr{V}ar(t) \cup \mathscr{V}ar(t') \subseteq W$, a set of substitutions $CSU_E^W(t = t')$ is said to be a *complete* set of unifiers for the equality $t = t'$ modulo E away from W iff: (i) each $\sigma \in CSU_E^W(t = t')$ is an E-unifier of $t = t'$; (ii) for any E-unifier ρ of $t = t'$ there is a $\sigma \in CSU_E^W(t = t')$ such that $\sigma|_W \sqsupseteq_E \rho|_W$; and (iii) for all $\sigma \in CSU_E^W(t = t')$, $Dom(\sigma) \subseteq (\mathscr{V}ar(t) \cup \mathscr{V}ar(t'))$ and $Ran(\sigma) \cap W = \emptyset$. Given a conjunction Γ of equations, a set U of E-unifiers of Γ is said to be *minimal* if it is complete and for all distinct elements σ and σ' in U, $\sigma \sqsupseteq_E \sigma'$ implies $\sigma =_E \sigma'$. A unification algorithm is said to be *finitary* and complete if it always terminates after generating a finite and complete set of unifiers. A unification algorithm is said to be *minimal* and complete if it always returns a minimal and complete set of unifiers.

A *rewrite rule* is an oriented pair $l \to r$, where $l \notin \mathscr{X}$ and $l, r \in \mathscr{T}_\Sigma(\mathscr{X})_\mathsf{s}$ for some sort $\mathsf{s} \in \mathsf{S}$. An *(unconditional) order-sorted rewrite theory* is a triple (Σ, E, R) with Σ an order-sorted signature, E a set of Σ-equations, and R a set of rewrite rules. The set R of rules is *sort-decreasing* if for each $t \to t'$ in

R, each $\mathsf{s} \in \mathsf{S}$, and each substitution σ, $t'\sigma \in \mathscr{T}_\Sigma(\mathscr{X})_\mathsf{s}$ implies $t\sigma \in \mathscr{T}_\Sigma(\mathscr{X})_\mathsf{s}$. The rewriting relation on $\mathscr{T}_\Sigma(\mathscr{X})$, written $t \to_R t'$ holds between t and t' iff there exist $p \in Pos_\Sigma(t)$, $l \to r \in R$ and a substitution σ, such that $t|_p = l\sigma$, and $t' = t[r\sigma]_p$. The relation $\to_{R/E}$ on $\mathscr{T}_\Sigma(\mathscr{X})$ is $=_E;\to_R;=_E$. The transitive (resp. transitive and reflexive) closure of $\to_{R/E}$ is denoted $\to^+_{R/E}$ (resp. $\to^*_{R/E}$).

Reducibility of $\to_{R/E}$ is undecidable in general since E-congruence classes can be arbitrarily large. Therefore, R/E-rewriting is usually implemented by R,E-rewriting under some conditions on R and E such as confluence, termination, and coherence (see [69, 89, 92]). A relation $\to_{R,E}$ on $\mathscr{T}_\Sigma(\mathscr{X})$ is defined as: $t \to_{R,E} t'$ iff there is a non-variable position $p \in Pos_\Sigma(t)$, a rule $l \to r$ in R, and a substitution σ such that $t|_p =_E l\sigma$ and $t' = t[r\sigma]_p$. The narrowing relation $\rightsquigarrow_{R,E}$ on $\mathscr{T}_\Sigma(\mathscr{X})$ is defined as: $t \overset{\sigma}{\rightsquigarrow}_{R,E} t'$ iff there is a non-variable position $p \in Pos_\Sigma(t)$, a rule $l \to r$ in R, and a substitution σ such that $t|_p\sigma =_E l\sigma$ and $t' = (t[r]_p)\sigma$. We call (Σ, B, E) a *decomposition* of an order-sorted equational theory $(\Sigma, E \uplus B)$ if B is regular, linear, sort-preserving, defined using top sorts, and has a finitary and complete unification algorithm, and equations E are oriented into rules $\overrightarrow{E}$ such that they are sort-decreasing and *convergent*, i.e., confluent, terminating, and strictly coherent modulo B [48, 80, 89]. The irreducible version of a term t is denoted by $t{\downarrow_{E,B}}$.

Given a decomposition (Σ, B, E) of an equational theory and a term t, a pair (t', θ) of a term t' and a substitution θ is an E,B-*variant* (or just a variant) of t if $t\theta{\downarrow_{E,B}} =_B t'$ and $\theta{\downarrow_{E,B}} =_B \theta$ [38, 60]. A *complete set of* E,B-*variants* [60] (up to renaming) of a term t is a subset, denoted by $[\![t]\!]_{\vec{E},B}$, of the set of all E,B-variants of t such that, for each E,B-variant (t', σ) of t, there is an E,B-variant $(t'', \theta) \in [\![t]\!]_{\vec{E},B}$ such that $(t'', \theta) \sqsupseteq_{E,B} (t', \sigma)$, i.e., there is a substitution ρ such that $t' =_E t''\rho$ and $\sigma|_{\mathscr{V}ar(t)} =_E (\theta\rho)|_{\mathscr{V}ar(t)}$. A decomposition (Σ, B, E) has the *finite variant property* (FVP) [60] (also called a *finite variant decomposition*) iff for each Σ-term t, there exists a complete and finite set $[\![t]\!]_{\vec{E},B}$ of variants of t. Note that whether a decomposition has the finite variant property is undecidable [26], but a technique based on the dependency pair framework has been developed in [60] and a semi-decision procedure that works well in practice is available in [33].

6.3 Variant-based Equational Unification in Maude 3.0

Rewriting logic [86] is a flexible semantic framework within which different concurrent systems can be naturally specified (see [88]). Rewriting Logic is

efficiently implemented in the high-performance system Maude [35], which has itself a formal environment of verification tools thanks to its reflective capabilities (see [36, 88]).

Maude 3.0 offers quite sophisticated symbolic capabilities (see [90] and references therein). Among these symbolic features, equational unification [35] is a twofold achievement. On the one hand, Maude provides an order-sorted equational unification command for any combination of symbols having any combination of associativity, commutativity, and identity [46]. This is remarkable, since there is no other system with such an advanced unification algorithm. On the other hand, a narrowing-based equational unification algorithm relying on the concept of the *variants* [38] of a term is also available. A variant of a term t is a pair consisting of a substitution σ and the canonical form of $t\sigma$. Narrowing was proved to be complete for unification in [68], but variant-based unification is decidable when the equational theory satisfies the *finite variant property* [38, 60]. The finite variant property has become an essential property in some research areas, such as cryptographic protocol analysis, where Maude-NPA [55], Tamarin [45] and AKISS [17] rely on the different unification features of Maude.

Let us make explicit the relation between variants and equational unification. First, we define the intersection of two sets of variants. Without loss of generality, we assume in this paper that each variant pair (t', σ) of a term t uses new freshly generated variables.

Definition 12 (Variant Intersection)**.** [60] *Given a decomposition* (Σ, B, E) *of an equational theory, two* Σ*-terms* t_1 *and* t_2 *such that* $W_\cap = \mathscr{V}ar(t_1) \cap \mathscr{V}ar(t_2)$ *and* $W_\cup = \mathscr{V}ar(t_1) \cup \mathscr{V}ar(t_2)$*, and two sets* V_1 *and* V_2 *of variants of* t_1 *and* t_2*, respectively, we define* $V_1 \cap V_2 = \{(u_1\sigma, \theta_1\sigma \cup \theta_2\sigma \cup \sigma) \mid (u_1, \theta_1) \in V_1 \wedge (u_2, \theta_2) \in V_2 \wedge \exists \sigma : \sigma \in CSU_B^{W_\cup}(u_1 = u_2) \wedge (\theta_1\sigma)|_{W_\cap} =_B (\theta_2\sigma)|_{W_\cap}\}$.

Then, we define variant-based unification as the computation of the variants of the two terms in a unification problem and their intersection.

Corollary 2 (Finitary $\mathscr{E}$-unification)**.** [60] *Let* (Σ, B, E) *be a finite variant decomposition of an equational theory. Given two terms* t, t'*, the set* $CSU_{E \cup B}^{\cap}(t = t') = \{\theta \mid (w, \theta) \in [\![t]\!]_{\vec{E},B} \cap [\![t']\!]_{\vec{E},B}\}$ *is a* finite and complete *set of unifiers for* $t = t'$.

The most recent version 3.0 of Maude [35] incorporates variant-based unification based on the folding variant narrowing strategy [60]. First, there exists a variant generation command of the form:

```
get variants [ n ] in ModId : Term .
```

where *n* is an optional argument providing a bound on the number of variants requested, so that if the cardinality of the set of variants is greater than the specified bound, the variants beyond that bound are omitted; and `ModId` is the identifier of the module where the command takes place. Second, there exists a variant-based unification command of the form:

```
variant unify [ n ] in ModId : T1 =? T1' /\ ... /\  Tk =? Tk' .
```

where $k \geq 1$ and *n* is an optional argument providing a bound on the number of unifiers requested, so that if there are more unifiers, those beyond that bound are omitted; and `ModId` is the identifier of the module where the command takes place.

Example 36. *Consider the following equational theory for exclusive-or that assumes three extra constants* `a`*,* `b`*, and* `c`*. The second equation is necessary for coherence modulo AC.*

```
fmod EXCLUSIVE-OR is
  sorts Elem EXor .
  subsort Elem < EXor .
  ops a b c : -> Elem .
  op mt : -> EXor .
  op _*_ : EXor EXor -> EXor [assoc comm] .
  vars X Y Z U V : [EXor] .
  eq [idem] :     X * X = mt    [variant] .
  eq [idem-Coh] : X * X * Z = Z [variant] .
  eq [id] :       X * mt = X    [variant] .
endfm
```

The attribute `variant` *specifies that these equations will be used for variant-based unification. Since this theory has the finite variant property (see [38, 60]), given the term* `X * Y` *it is easy to verify that there are seven most general variants.*

```
Maude> get variants in EXCLUSIVE-OR : X * Y .

Variant #1                                ...          Variant #7
[EXor]: #1:[EXor] * #2:[EXor]             ...          [EXor]: %1:[EXor]
X --> #1:[EXor]                           ...          X --> %1:[EXor]
Y --> #2:[EXor]                           ...          Y --> mt
```

Note that Maude produces fresh variables of the form `#n:Sort` *or* `%n:Sort` *using two different counters (see [35] for details). When we consider a variant unification problem between terms* $X * Y$ *and* $U * V$*, there are* 57 *unifiers:*

```
Maude> variant unify in EXCLUSIVE-OR : X * Y =? U * V  .
Unifier #1                          ...          Unifier #2
X --> %1:[EXor] * %3:[EXor]         ...          X --> %1:[EXor] * %3:[EXor]
Y --> %2:[EXor] * %4:[EXor]         ...          Y --> %2:[EXor]
V --> %1:[EXor] * %2:[EXor]         ...          V --> %1:[EXor] * %2:[EXor]
U --> %3:[EXor] * %4:[EXor]         ...          U --> %3:[EXor]
```

However, this variant-based unification algorithm may compute many more unifiers than the necessary or may not be able to stop immediately. For instance, it is well-known that unification in the exclusive-or theory is unitary, i.e., there exists only one most general unifier modulo exclusive-or [71]. For the unification problem $X * Y \stackrel{?}{=} U * V$ of Example 36, the most general unifier w.r.t. $\sqsupseteq_{E \cup B}$ is $\{X \mapsto Y * U * V\}$, which should be appropriately written as $\sigma = \{X \mapsto Y' * U' * V', Y \mapsto Y', U \mapsto U', V \mapsto V'\}$. Note that $\{Y \mapsto X * U * V\}$, $\{U \mapsto Y * X * V\}$, and $\{V \mapsto Y * U * X\}$ are equivalent to the former unifier w.r.t. $\sqsupseteq_{E \cup B}$ by composing σ with, respectively, $\rho_1 = \{Y' \mapsto X'' * U'' * V'', X' \mapsto X'', U' \mapsto U'', V' \mapsto V''\}$, $\rho_2 = \{U' \mapsto Y'' * X'' * V'', X' \mapsto X'', Y' \mapsto Y'', V' \mapsto V''\}$, and $\rho_3 = \{V' \mapsto Y'' * U'' * X'', X' \mapsto X'', U' \mapsto U'', Y' \mapsto Y''\}$. Similarly, $\{X \mapsto U, Y \mapsto V\}$ and $\{X \mapsto V, Y \mapsto U\}$ are equivalent to all the previous ones.

Furthermore, since the variants of both terms are generated by Corollary 2, there may be very simple unification problems such as $X \stackrel{?}{=} t$ where the generation of the variants of t is unnecessary. For example, when unifying terms X and $U * V$, the variants of $U * V$ are generated

```
Maude> variant unify in EXCLUSIVE-OR : X =? U * V  .

Unifier #1                    Unifier #2          Unifier #3
X --> %1:[EXor] * %2:[EXor]   X --> mt            X --> #2:[EXor] * #3:[EXor]
V --> %1:[EXor]               V --> #1:[EXor]     V --> #1:[EXor] * #2:[EXor]
U --> %2:[EXor]               U --> #1:[EXor]     U --> #1:[EXor] * #3:[EXor]

Unifier #4                  Unifier #5                    Unifier #6
X --> #1:[EXor]               X --> #1:[EXor]               X --> #1:[EXor]
V --> #1:[EXor] * #2:[EXor]   V --> #2:[EXor]               V --> mt
U --> #2:[EXor]               U --> #1:[EXor] * #2:[EXor]   U --> #1:[EXor]

Unifier #7
X --> #1:[EXor]
```

```
  V --> #1:[EXor]
  U --> mt
```

but it is clear that the simplest, most general unifier is $\{X \mapsto U * V\}$. In [59], a new procedure to reduce the number of variant unifiers in situations like this was developed. We showed that this new procedure pays off in practice using both the exclusive-or and the abelian group equational theories.

6.4 Constructor-Root Variant-based Unification

Both the "logic" notion of a functor and the "functional" notion of a constructor refer to a symbol not appearing in the root position of the left-hand side of any predicate or equation. This notion of constructor allows to split a signature Σ as a disjoint union $\Sigma = \mathscr{D} \uplus \mathscr{C}$ where $\mathscr{D}$ are called *defined* symbols and $\mathscr{C}$ are called *constructor* symbols. In a decomposition (Σ, B, E), the *canonical term algebra* $Can_{\Sigma/(E,B)} = \{t{\downarrow_{E,B}} \mid t \in \mathscr{T}_\Sigma\}$ is typically made of constructor terms, but this more general notion of constructor differs from the "logic" and "functional" notions. A decomposition (Σ, B, E) *protects* a *constructor decomposition* $(\mathscr{C}, B_\mathscr{C}, E_\mathscr{C})$ iff $\mathscr{C} \subseteq \Sigma$, $B_\mathscr{C} \subseteq B$, and $E_\mathscr{C} \subseteq E$, and for all $t, t' \in \mathscr{T}_\mathscr{C}(\mathscr{X})$ we have: (i) $t =_{B_\mathscr{C}} t' \iff t =_B t'$, (ii) $t = t{\downarrow_{E_\mathscr{C}, B_\mathscr{C}}} \iff t = t{\downarrow_{E,B}}$, and (iii) $Can_{\mathscr{C}/(E_\mathscr{C}, B_\mathscr{C})} = Can_{\Sigma/(E,B)}|_\mathscr{C}$. A *constructor decomposition* $(\mathscr{C}, B_\mathscr{C}, \emptyset)$ is called *free*. For instance, the modular exponentiation property typical of Diffie-Hellman protocols is defined using two versions of the exponentiation operator and an auxiliary associative-commutative symbol $*$ for exponents so that $(z^x)^y = (z^y)^x = z^{x*y}$. Note that, in the lefthand side of the equation, the outermost exponentiation operator is defined, whereas the innermost exponentiation operator is constructor.

```
fmod DH-CFVP is
  sorts Exp Elem ElemSet Gen .
  subsort Elem < ElemSet .
  ops a b c : -> Elem [ctor] .
  op exp : Gen ElemSet -> Exp [ctor] .
  op exp : Exp ElemSet -> Exp .
  op _*_ : ElemSet ElemSet -> ElemSet [assoc comm ctor] .
  var X : Gen .
  vars Y Z : ElemSet .
  eq exp(exp(X,Y),Z) = exp(X,Y * Z) [variant] .
endfm
```

Note that it may not always be possible to provide a (free) constructor decomposition, such as Example 36 where the exclusive-or symbol works both as defined and constructor (see [35] for a detailed discussion). However, it is common to combine an equational theory with many different additional constructor symbols, as shown in Section 6.5.

The notion of a constructor symbol has not yet been integrated into the variant-based equational unification procedure of Maude. An integration of the notion of constructor involves two challenges. On the one hand, when we consider the variant unification problem above between terms X and $U * V$, the fast unification algorithm of [59] is able to return only one unifier but still generates all the variants of term $U * V$, unnecessarily consuming resources. On the other hand, a unification problem between terms $f(X * Y)$ and $g(U * V)$ where f and g are different constructor symbols forces the generation of all the variants of the terms $X * Y$ and $U * V$ wasting resources. Let us consider a unification problem $C_1[X] \stackrel{?}{=} C_2[t]$ where both C_1 and C_2 are made of constructor symbols and either there exists σ s.t. $C_1[\Box]\sigma =_B C_2[\Box]\sigma$ or there is no such σ.

Definition 13 (Constructor-root Position)**.** *Given a decomposition (Σ, B, E) protecting a free constructor decomposition $(\mathscr{C}, B_{\mathscr{C}}, \emptyset)$ and given a Σ-term t and a position $p \in \mathscr{P}os(t)$, we say p is a* constructor-root *position in t if for all $q < p$, $root(t|_q) \in \mathscr{C}$.*

Definition 14 (Constructor-root Variable)**.** *Given a Σ-term t and a variable x, we say x is a* constructor-root *variable in t if for all $p \in Pos_x(t)$, p is constructor-root in t.*

First, we define the case when there exists σ s.t. $C_1[\Box]\sigma =_B C_2[\Box]\sigma$. Intuitively, a variant unifier σ of t_1 and t_2 is constructor-root if each variable in $Ran(\sigma)$ is under a constructor-root variable of t_1 and t_2.

Definition 15 (Constructor-root Variant Unifier)**.** *Given a decomposition (Σ, B, E) protecting a free constructor decomposition $(\mathscr{C}, B_{\mathscr{C}}, \emptyset)$, two Σ-terms t_1 and t_2 s.t. $W_\cap = \mathscr{V}ar(t_1) \cap \mathscr{V}ar(t_2)$, $W_\cup = \mathscr{V}ar(t_1) \cup \mathscr{V}ar(t_2)$, $(u_1, \theta_1) \in [\![t_1]\!]_{\vec{E},B}$, $(u_2, \theta_2) \in [\![t_2]\!]_{\vec{E},B}$, and $\sigma \in CSU_B^{W_\cup}(u_1 = u_2)$ s.t. $(\theta_1\sigma)|_{W_\cap} =_B (\theta_2\sigma)|_{W_\cap}$, the unifier $(\theta_1 \cup \theta_2)\sigma$ is called* constructor-root *if for each $x \mapsto t \in \sigma$, either (i) $x \mapsto t$ is a variable renaming, (ii) x is a constructor-root variable in u_1 and u_2, or (iii) for each $x' \mapsto t' \in \sigma \setminus \{x \mapsto t\}$ (and there exists at least one such binding) s.t. $t' =_B C[t]$, then x' is a constructor-root variable in u_1 and u_2.*

Let us motivate the usefulness of a constructor-root unifier. Given the unification problem $\mathtt{X} \stackrel{?}{=} \mathtt{V} * \mathtt{U}$ above, the unifier $\{\mathtt{X} \mapsto \%1 * \%2, \mathtt{V} \mapsto \%1, \mathtt{U} \mapsto \%2\}$ is constructor-root, since $\mathtt{X}$ is a constructor-root variable in the left unificand

and V and U are not constructor-root variables but the variables %1 and %2 used in the bindings of V and U appear in the binding of X. Hence, we can safely avoid the generation of the variants of $V * U$. Note that the unifier $\{\mathtt{X} \mapsto \mathtt{mt},\ \mathtt{V} \mapsto \%1,\ \mathtt{U} \mapsto \%1\}$ is not constructor-root because V and U are not constructor-root variables and for the bindings $\mathtt{V} \mapsto \%1$ and $\mathtt{U} \mapsto \%1$ there is no other binding $x' \mapsto t'$ such that %1 is a subterm of t' and x' is a constructor-root variable.

Lemma 16 (Constructor-root Variant Unifier)**.** *Given a decomposition* (Σ, B, E) *protecting a free constructor decomposition* $(\mathscr{C}, B_{\mathscr{C}}, \emptyset)$*, two* Σ*-terms* t_1 *and* t_2 *s.t.* $W_\cap = \mathscr{V}ar(t_1) \cap \mathscr{V}ar(t_2)$*,* $W_\cup = \mathscr{V}ar(t_1) \cup \mathscr{V}ar(t_2)$*,* $(u_1, \theta_1) \in [\![t_1]\!]_{\vec{E},B}$*,* $(u_2, \theta_2) \in [\![t_2]\!]_{\vec{E},B}$*, and a* constructor-root variant unifier $\sigma \in CSU_B^{W_\cup}(u_1 = u_2)$ *s.t.* $(\theta_1\sigma)|_{W_\cap} =_B (\theta_2\sigma)|_{W_\cap}$*, then* $\forall (u_1', \theta_1') \in [\![t_1]\!]_{\vec{E},B}$ *s.t.* $(u_1', \rho) \in [\![u_1]\!]_{\vec{E},B}$ *and* $\theta_1'|_{W_\cup} =_B \theta_1\rho|_{W_\cup}$*, if there exists* $\sigma' \in CSU_B^{W_\cup}(u_1' = u_2)$ *s.t.* $(\theta_1'\sigma')|_{W_\cap} =_B (\theta_2\sigma')|_{W_\cap}$*, then* $((\theta_1 \cup \theta_2)\sigma)|_{W_\cup}$ *and* $((\theta_1' \cup \theta_2)\sigma')|_{W_\cup}$ *are both equational unifiers of* t_1 *and* t_2 *but* $((\theta_1 \cup \theta_2)\sigma)|_{W_\cup} \sqsupseteq_{E \cup B} ((\theta_1' \cup \theta_2)\sigma')|_{W_\cup}$*. Similarly for any* $(u_2', \theta_2') \in [\![t_2]\!]_{\vec{E},B}$*.*

Proof 2. *By contradiction. Let us assume* $\exists \sigma' \in CSU_B^{W_\cup}(u_1' = u_2)$ *s.t.* $((\theta_1 \cup \theta_2)\sigma)|_{W_\cup} \not\sqsupseteq_{E \cup B} ((\theta_1' \cup \theta_2)\sigma')|_{W_\cup}$*. First,* $\theta_1'|_{t_1} = \theta_1|_{t_1}\rho|_{u_1}$ *and thus the difference is in* $\sigma|_{u_2}$ *and* $\sigma'|_{u_2}$*. By the constructor-root property,* $\forall x \mapsto t \in \sigma|_{u_2}$ *either* x *is a constructor-root variable in* u_2 *or* $\forall x' \mapsto t' \in (\sigma \setminus \{x \mapsto t\})|_{u_2}$ *s.t.* $t' =_B C[t]$*,* x' *is a constructor-root variable in* u_2*. But then* $\forall y \in \mathscr{V}ar(u_2)$*, there exist* $y \mapsto w_1 \in \sigma|_{u_2}$ *and* $y \mapsto w_2 \in \sigma'|_{u_2}$ *and* $(w_2, \rho) \in [\![w_1]\!]_{\vec{E},B}$*, i.e.,* $\sigma|_{u_2} \sqsupseteq_{E \cup B} \sigma'|_{u_2}$*, which contradicts the assumption.*

We define the case when there is no σ s.t. $C_1[\Box]\sigma =_B C_2[\Box]\sigma$. Intuitively, two terms that form a constructor-root failure pair will never unify despite any further variant computation.

Definition 17 (Constructor-root Failure Pair)**.** *Given a decomposition* (Σ, B, E) *protecting a free constructor decomposition* $(\mathscr{C}, B_{\mathscr{C}}, \emptyset)$*, two* Σ*-terms* t_1 *and* t_2 *s.t.* $W_\cap = \mathscr{V}ar(t_1) \cap \mathscr{V}ar(t_2)$*,* $W_\cup = \mathscr{V}ar(t_1) \cup \mathscr{V}ar(t_2)$*,* $(u_1, \theta_1) \in [\![t_1]\!]_{\vec{E},B}$*, and* $(u_2, \theta_2) \in [\![t_2]\!]_{\vec{E},B}$*, the pair* (u_1, u_2) *is a* constructor-root failure pair *if* $CSU_B^{W_\cup}(u_1 = u_2) = \emptyset$ *and there exists two constructor contexts* $C_1[\Box, \ldots, \Box]$ *and* $C_2[\Box, \ldots, \Box]$*, terms* $v_1, \ldots, v_n, w_1, \ldots, w_m$*, and fresh distinct variables* $x_1, \ldots, x_n, y_1, \ldots, y_m$ *s.t.* $u_1 =_B C_1[v_1, \ldots, v_n]$*,* $u_2 =_B C_2[w_1, \ldots, w_m]$*, and* $CSU_B^{W_\cup}(C_1[x_1, \ldots, x_n] = C_2[y_1, \ldots, y_m]) = \emptyset$*.*

Let us motivate the usefulness of a constructor-root failure pair. Given the unification problem $\mathtt{f(X * Y)} \stackrel{?}{=} \mathtt{g(V * U)}$ above where f and g are different constructor symbols without axioms, the two terms do not unify modulo the

axioms of $*$ but neither $f(W)$ and $g(W')$ do. Hence, we can safely avoid the generation of the variants of $V*U$ and $X*Y$.

Lemma 18 (Constructor-root Failure Pair)**.** *Given a decomposition (Σ, B, E) protecting a free constructor decomposition $(\mathscr{C}, B_{\mathscr{C}}, \emptyset)$, two Σ-terms t_1 and t_2 s.t. $W_\cup = \mathscr{V}ar(t_1) \cup \mathscr{V}ar(t_2)$, $(u_1, \theta_1) \in [\![t_1]\!]_{\vec{E},B}$, $(u_2, \theta_2) \in [\![t_2]\!]_{\vec{E},B}$, and (u_1, u_2) is a constructor-root failure term, then $CSU^{W_\cup}_{E \cup B}(u_1 = u_2) = \emptyset$.*

Proof 3. *Immediate by Definition 17.*

These positive and negative stopping criteria, however, become useful only if we do not generate all the variants a priori, as it is done in Corollary 2 as well as the fast unification technique of [59]. Some incremental generation of variants is required.

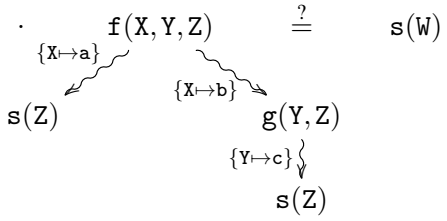
Example 37. *Consider the following theory where constructors have the* `ctor` *attribute.*

```
fmod FASTvsCR is sort S .
  ops a b c : -> S [ctor] .
  op s : S -> S [ctor] .
  op g : S S -> S .
  op f : S S S -> S .
  vars X Y Z W : S .
  eq f(a,X,Y) = s(Y) [variant] .
  eq f(b,X,Y) = g(X,Y) [variant] .
  eq g(c,Y) = s(Y) [variant] .
endfm
```

Consider the unification problem (a) $f(X,Y,Z) = s(W)$ with only two unifiers and its variant generation.

```
variant unify in FG : f(X, Y, Z) =? s(W) .

 Unifier #1        Unifier #2
 X --> a           X --> b
 Y --> #2:S        Y --> c
 Z --> #1:S        Z --> %1:S
 W --> #1:S        W --> %1:S
```



Let us assume we have an expression ♣ with a considerably large narrowing tree and two new unification problems (b) $f(X,Y,\clubsuit) = s(W)$ and (c) $f(X,\clubsuit,Z) = s(W)$. Note that the unifiers of (a) are still valid for (b), whereas only the first unifier of (a) is valid for (c), assuming ♣ never narrows into c. Both the variant-based unification

command of Maude and the fast command of [59] cannot avoid the computation of ♣ in both unification problems (b) and (c). However, the technique described below is able to avoid the full computation of ♣ in (b), since the two unifiers are constructor-root, although it cannot avoid the full computation of ♣ in (c).

We extend the notions of constructor-root unifier and constructor-root failure pair to the pairwise combination of all the variants of a unification problem.

Definition 19 (Constructor-Root Intersection)**.** *Given a decomposition* (Σ, B, E) *protecting a free constructor decomposition* $(\mathscr{C}, B_{\mathscr{C}}, \emptyset)$*, two* Σ*-terms* t_1 *and* t_2 *such that* $W_\cap = \mathscr{V}ar(t_1) \cap \mathscr{V}ar(t_2)$ *and* $W_\cup = \mathscr{V}ar(t_1) \cup \mathscr{V}ar(t_2)$*, and two sets* V_1 *and* V_2 *of variants of* t_1 *and* t_2*, respectively, we say that an intersection* $V_1 \cap V_2$ *is* constructor-root *if for each leaf* $(u_1, \theta_1) \in V_1$ *(resp.* $(u_2, \theta_2) \in V_2$*), and for each leaf* $(u_2, \theta_2) \in V_2$ *(resp.* $(u_1, \theta_1) \in V_1$*) such that* $\sigma \in CSU_B^{W_\cup}(u_1 = u_2)$ *and* $(\theta_1\sigma)|_{W_\cap} =_B (\theta_2\sigma)|_{W_\cap}$*, we have* $(\theta_1 \cup \theta_2)\sigma$ *is* constructor-root*.*

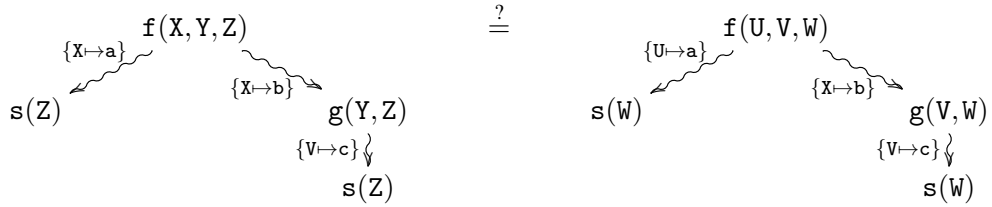
Definition 20 (Failure Intersection)**.** *Given a decomposition* (Σ, B, E) *protecting a free constructor decomposition* $(\mathscr{C}, B_{\mathscr{C}}, \emptyset)$ *two* Σ*-terms* t_1 *and* t_2 *such that* $W_\cap = \mathscr{V}ar(t_1) \cap \mathscr{V}ar(t_2)$ *and* $W_\cup = \mathscr{V}ar(t_1) \cup \mathscr{V}ar(t_2)$*, and two sets* V_1 *and* V_2 *of variants of* t_1 *and* t_2*, respectively, we say that an intersection* $V_1 \cap V_2$ *is a* failure intersection *if for each leaf* $(u_1, \theta_1) \in V_1$ *(resp.* $(u_2, \theta_2) \in V_2$*), and for each leaf* $(u_2, \theta_2) \in V_2$ *(resp.* $(u_1, \theta_1) \in V_1$*) such that* $\sigma \in CSU_B^{W_\cup}(u_1 = u_2)$ *and* $(\theta_1\sigma)|_{W_\cap} =_B (\theta_2\sigma)|_{W_\cap}$*, we have the pair* (u_1, u_2) *is a constructor-root failure pair.*

The following example shows that the folding variant narrowing trees of both terms t_1, t_2 of a unification problem $t_1 = t_2$ must be unfolded down to a frontier where all leaves of t_1 are tested for unification with all the leaves of t_2.

Example 38. *Let us consider Example 37 and the unification problem* $f(X,Y,Z) = f(U,V,W)$*.*

```
variant unify in FG : f(X, Y, Z) =? f(U, V, W)  .

  Unifier #1          Unifier #2          Unifier #3          Unifier #4
  X --> %1:S          X --> a             X --> a             X --> b
  Y --> %2:S          Y --> %2:S          Y --> #2:S          Y --> c
  Z --> %3:S          Z --> %1:S          Z --> #1:S          Z --> #1:S
  U --> %1:S          U --> a             U --> b             U --> a
  V --> %2:S          V --> %3:S          V --> c             V --> #2:S
  W --> %3:S          W --> %1:S          W --> #1:S          W --> #1:S
```



The terms at the top position of both narrowing trees clearly unify, but the unifier is not constructor-root, so we must continue expanding both narrowing trees. The condition that all leaves of t_1 are unifiable with all the leaves of t_2 is reached only at depth 2*. Indeed, if we expand the left unificand completely but the right unificand only down to the leftmost branch, then the two leaves of the narrowing tree of the left unificand unify with the leftmost leaf of the narrowing tree of the right unificand, but we may miss the last two unifiers reported above if we stop here.*

We define variant-based unification as the computation of the variants of the two terms in a unification problem. We abuse the notation and write $\mathbb{P}([\![t]\!]_{\vec{E},B})$ for the powerset of all the subsets of $[\![t]\!]_{\vec{E},B}$ such that each $V \in \mathbb{P}([\![t]\!]_{\vec{E},B})$ corresponds to the variants of a term t associated to a particular narrowing tree produced by the folding variant narrowing strategy from term t. We also write $CSU_{E\cup B}^{\cap,V_1,V_2}(t = t')$ for a version of the unification algorithm of Corollary 2 that uses sets V_1 and V_2 of variants of t and t', respectively, instead of generating all the variants.

Definition 21 (Constructor-Root Variant-based Unification)**.** *Let (Σ, B, E) be a finite variant decomposition of an equational theory protecting a free constructor decomposition $(\mathscr{C}, B_{\mathscr{C}}, \emptyset)$. Given two terms t, t' and two sets of variants $V_1 \in \mathbb{P}([\![t]\!]_{\vec{E},B})$, $V_2 \in \mathbb{P}([\![t']\!]_{\vec{E},B})$, the constructor-root variant unifiers are*

$$CSU_{E\cup B}^{\overline{\cap}}(t = t') = \begin{cases} \emptyset & \text{if } \exists V_1 \in \mathbb{P}([\![t]\!]_{\vec{E},B}), V_2 \in \mathbb{P}([\![t']\!]_{\vec{E},B}), \\ & \text{and they are the smallest sets s.t} \\ & V_1 \cap V_2 \text{ is a failure intersection} \\ CSU_{E\cup B}^{\cap,V_1,V_2}(t = t') & \text{if } \exists V_1 \in \mathbb{P}([\![t]\!]_{\vec{E},B}), V_2 \in \mathbb{P}([\![t']\!]_{\vec{E},B}), \\ & \text{and they are the smallest sets s.t} \\ & V_1 \cap V_2 \text{ is constructor-root} \\ CSU_{E\cup B}^{\cap,[\![t]\!]_{\vec{E},B},[\![t']\!]_{\vec{E},B}}(t = t') & \text{otherwise} \end{cases}$$

Proposition 22 (Constructor-Root Variant-based Unification)**.** *Let* (Σ, B, E) *be a finite variant decomposition of an equational theory protecting a free constructor decomposition* $(\mathscr{C}, B_{\mathscr{C}}, \emptyset)$*. Given two terms* t, t'*, the set* $CSU^{\overline{\cap}}_{E\cup B}(t = t')$ *is a* finite and complete *set of unifiers for* $t = t'$*.*

Proof 4. *By contradiction. Let us assume that* $CSU^{\overline{\cap}}_{E\cup B}(t = t')$ *is not a complete set of unifiers of* t *and* t'*. That is, there exists a unifier* $\rho' \in CSU^{\cap}_{E\cup B}(t = t')$ *and there is no unifier* $\rho \in CSU^{\overline{\cap}}_{E\cup B}(t = t')$ *s.t.* $\rho \sqsupseteq_{E\cup B} \rho'$*. By definition, there exist smallest sets* $V_1 \in \mathbb{P}([\![t]\!]_{\vec{E},B})$*,* $V_2 \in \mathbb{P}([\![t']\!]_{\vec{E},B})$ *s.t.* $V_1 \cap V_2$ *is constructor-root or a failure pair. The case of a failure pair is immediate by Lemma 18. Since* $\rho' \in CSU^{\cap}_{E\cup B}(t = t')$*, we have that there exists* $u_1, u_2, \theta_1, \theta_2, \sigma$ *s.t.* $\rho' = \theta_1\sigma \cup \theta_2\sigma \cup \sigma$*,* $(u_1, \theta_1) \in [\![t]\!]_{\vec{E},B}$*,* $(u_2, \theta_2) \in [\![t']\!]_{\vec{E},B}$*,* $\sigma \in CSU^{W_\cup}_B(u_1 = u_2)$*, and* $(\theta_1\sigma)|_{W_\cap} =_B (\theta_2\sigma)|_{W_\cap}$*. Since* $V_1 \cap V_2$ *is constructor-root, there must be two leaves* $(v_1, \tau_1) \in V_1$*,* $(v_2, \tau_2) \in V_2$ *and a substitution* τ_3 *s.t.* $\tau_3 \in CSU^{W_\cup}_B(v_1 = v_2)$*,* $(\tau_1\tau_3)|_{W_\cap} =_B (\tau_2\tau_3)|_{W_\cap}$*, and* $(\tau_1 \cup \tau_2)\tau_3$ *is* constructor-root*. Furthermore, the variant* (u_1, θ_1) *(resp.* (u_2, θ_2)*) is obtained by further narrowing of* v_1 *(resp.* v_2*), i.e.,* $(u_1, \tau_1') \in [\![v_1]\!]_{\vec{E},B}$ *and* $(u_2, \tau_2') \in [\![v_2]\!]_{\vec{E},B}$*. But then the conclusion follows, since the statement is* $((\tau_1 \cup \tau_2)\tau_3)|_{W_\cup} \sqsupseteq_{E\cup B} ((\theta_1 \cup \theta_2)\sigma)|_{W_\cup}$*.*

6.5 Experimental Evaluation

We have performed some experiments with the constructor-root variant-based unification, which are available at `http://safe-tools.dsic.upv.es/cr-mgvu`.

All the experiments were conducted on a PC with a 3.3GHz Intel Xeon E5-1660 and 64GB RAM. We created a battery of 15 different unification problems for both the *exclusive-or* and the *abelian group* theories. These are among the most complicated cryptographic theories in protocol analysis that Maude-NPA [55], Tamarin [45] and AKISS [17] can hardly handle. Indeed, the *exclusive-or* and the *abelian group* theories cannot be specified in Maude using constructor symbols and we introduce arbitrary constructors f_1, f_2, f_3, f_4, f_5, where the subindex indicates the number of arguments. This is a common situation in crypto protocol analysis where the cryptographic properties are combined with many different additional constructor symbols. Experiments using other cryptographic theories, such as Diffie-Hellmann exponentiation, or more traditional programs, such as manipulating complex data structures, could also have been included but were discarded because the improvement is less remarkable.

For each problem and theory, we computed: (i) the unifiers using the standard `variant unify` command provided by the C++ core system of Maude;

	Unification problem	$\#_{maude}$	$\#_{fast}$	$\#_{cr}$	$\#_{cr+fast}$	$\mathscr{T}_{maude}$	$\mathscr{T}_{fast}$	$\mathscr{T}_{cr}$	$\mathscr{T}_{cr+fast}$
P_1	$V_1 \stackrel{?}{=} V_2 * V_3 * V_4$	57	1	1	1	50	95	1	1
P_2	$V_1 \stackrel{?}{=} f_3(V_2 * V_3, f_1(V_3 * V_4), f_2(V_2, f_1(V_4)))$	61	1	1	1	172	243	2	2
P_3	$V_1 * V_2 \stackrel{?}{=} V_3 * V_4$	57	8	41	8	9	89	83	131
P_4	$V_1 * V_2 \stackrel{?}{=} f_2(V_3, f_1(V_4 * V_5))$	28	4	4	4	12	18	8	10
P_5	$f_1(a) * f_1(V_1 * V_2) \stackrel{?}{=} f_1(b * V_3) * f_1(c * V_4)$	74	54	74	54	53	112	161	268
P_6	$f_1(V_1) \stackrel{?}{=} f_1(V_2 * V_3 * f_2(V_4, V_5))$	21	1	1	1	4	17	1	1
P_7	$f_2(V_1, V_2 * V_3 * V_4) \stackrel{?}{=} f_2(V_5 * f_1(V_6 * V_7), V_8)$	1596	1	1	1	3473	41592	9	9
P_8	$f_3(V_1, V_2, V_3) \stackrel{?}{=} f_3(f_1(V_4 * V_5), f_1(V_6 * V_7 * V_8), f_1(f_1(V_9)))$	399	1	1	1	507	3289	8	8
P_9	$f_4(V_1, V_2 * V_3, f_1(V_2 * V_4 * V_5), V_3) \stackrel{?}{=}$ $f_4(f_2(V_6, V_7) * V_6, V_8, V_9, f_1(f_1(V_{10})))$	492	14	1	1	122544	61184	14	14
P_{10}	$f_5(V_1, V_2 * V_3 * V_4, f_2(V_5, f_1(V_3 * V_4)), V_4, f_1(V_6 * V_7)) \stackrel{?}{=}$ $f_5(f_2(V_8, V_9), V_{10}, V_{11}, f_1(f_1(V_8)), V_{12})$	161	11	1	1	6780	9249	16	16
P_{11}	$f_1(V_1 * V_2) \stackrel{?}{=} f_2(V_3 * V_4 * V_5, f_2(V_4, V_5))$	0	0	0	0	985	125	1	1
P_{12}	$f_2(V_1, V_2 * V_3 * V_4) \stackrel{?}{=} f_3(V_5 * f_1(V_6 * V_7), V_8, V_9)$	0	0	0	0	2987	57	1	1
P_{13}	$f_3(V_1, V_2, V_3 * V_4) \stackrel{?}{=} f_2(f_1(V_5 * V_6 * V_7), f_1(f_1(V_8)))$	0	0	0	0	468	48	1	1
P_{14}	$f_4(V_1, V_2 * V_3, f_1(V_2 * V_4 * V_5), V_3) \stackrel{?}{=}$ $f_3(f_2(V_6, V_7) * V_6, V_8, f_1(f_1(V_9)))$	0	0	0	0	118028	53653	1	1
P_{15}	$f_5(V_1, V_2 * V_3 * V_4, f_2(V_5, f_1(V_3 * V_4)), V_6, f_1(V_7 * V_8)) \stackrel{?}{=}$ $f_4(f_2(V_9, V_{10}), V_{11}, f_1(f_1(V_9)), V_{12})$	0	0	0	0	6968	7033	1	1

Table 6.1: Experimental evaluation (exclusive-or)

(ii) the unifiers using the algorithm $CSU^{\widehat{\cap}}_{E\cup B}(t = t')$ of [59]; (iii) the unifiers using the algorithm $CSU^{\widehat{\cap}}_{E\cup B}(t = t')$ of Definition 21, and (iv) the unifiers using the algorithm $CSU^{\widehat{\cap}}_{E\cup B}(t = t')$ obtained from $CSU^{\widehat{\cap}}_{E\cup B}(t = t')$ by replacing $CSU^{\widehat{\cap},V_1,V_2}_{E\cup B}(t = t')$ with $CSU^{\widehat{\cap},V_1,V_2}_{E\cup B}(t = t')$ from [59]. Note that (ii), (iii), and (iv) are implemented at the metalevel of Maude. We measured both the number of computed unifiers and the time required for their computation.

Table 6.1 (resp. Table 6.2) shows the results obtained for the *exclusive-or* (resp. *abelian group)* theory. *T/O* indicates that a generous 4 hours *timeout* was reached without any response. The first column describes the unification problem, while the following $\#_{maude}$, $\#_{fast}$, $\#_{cr}$ and $\#_{cr+fast}$ columns show the number of computed unifiers for all four unification algorithms (i), (ii), (iii), (iv) described above and the columns $\mathscr{T}_{maude}$, $\mathscr{T}_{fast}$, $\mathscr{T}_{cr}$ and $\mathscr{T}_{cr+fast}$ show the time (in milliseconds) required to execute the unification command. Note that it is unfair to compare the performance between compiled code ($\mathscr{T}_{maude}$ column) and interpreted code ($\mathscr{T}_{fast}$, $\mathscr{T}_{cr}$ and $\mathscr{T}_{cr+fast}$ columns), i.e., the C++ core system of Maude and a Maude program using Maude's metalevel. However, our constructor-root unification algorithm is able to beat the compiled code in almost all the unification problems.

Tables 6.1 and 6.2 show that the *cr+fast* combination is the best choice, since it combines the benefits of both the *fast* unification algorithm of [59] and the new constructor-root unification algorithm *cr*. For the number of unifiers, *cr* always reported less unifiers than Maude except for problem P_{20}, where

	Unification problem	$\#_{maude}$	$\#_{fast}$	$\#_{cr}$	$\#_{cr+fast}$	$\mathcal{T}_{maude}$	$\mathcal{T}_{fast}$	$\mathcal{T}_{cr}$	$\mathcal{T}_{cr+fast}$
P_{16}	$V_1 \stackrel{?}{=} V_2 + V_3 + V_4$	3702	1	1	1	4344602	5034046	1	1
P_{17}	$V_1 \stackrel{?}{=} f_3(V_2 + V_3, f_1(V_3 + V_4), f_2(V_2, f_1(V_4)))$	3789	1	1	1	6956340	5413107	2	2
P_{18}	$V_1 + V_2 \stackrel{?}{=} V_3 + V_4$	3611	664	3313	664	36258	547115	253078	657746
P_{19}	$V_1 + V_2 \stackrel{?}{=} f_2(V_3, f_1(V_4 + V_5))$	376	8	52	8	26425	5083	366	2000
P_{20}	$f_1(a) + f_1(V_1 + V_2) \stackrel{?}{=} f_1(b + V_3) + f_1(c + V_4)$	316	193	316	193	10202	4175	3161	6976
P_{21}	$f_1(V_1) \stackrel{?}{=} f_1(V_2 + V_3 + f_2(V_4, V_5))$	158	1	1	1	426	1410	2	2
P_{22}	$f_2(V_1, V_2 + V_3 + V_4) \stackrel{?}{=} f_2(V_5 + f_1(V_6 + V_7), V_8)$	-	-	1	1	T/O	T/O	11	11
P_{23}	$f_3(V_1, V_2, V_3) \stackrel{?}{=}$ $f_3(f_1(V_4 + V_5), f_1(V_6 + V_7 + V_8), f_1(f_1(V_9)))$	-	-	1	1	T/O	T/O	11	13
P_{24}	$f_4(V_1, V_2 + V_3, f_1(V_2 + V_4 + V_5), V_3) \stackrel{?}{=}$ $f_4(f_2(V_6, V_7) * V_6, V_8, V_9, f_1(f_1(V_{10})))$	-	-	1	1	T/O	T/O	19	19
P_{25}	$f_5(V_1, V_2 + V_3 + V_4, f_2(V_5, f_1(V_3 + V_4)), V_4, f_1(V_6 + V_7)) \stackrel{?}{=}$ $f_5(f_2(V_8, V_9), V_{10}, V_{11}, f_1(f_1(V_8)), V_{12})$	-	-	1	1	T/O	T/O	24	24
P_{26}	$f_1(V_1 + V_2) \stackrel{?}{=} f_2(V_3 + V_4 + V_5, f_2(V_4, V_5))$	-	0	0	0	T/O	5594580	1	1
P_{27}	$f_2(V_1, V_2 + V_3 + V_4) \stackrel{?}{=} f_3(V_5 + f_1(V_6 + V_7), V_8, V_9)$	-	0	0	0	T/O	4399334	1	1
P_{28}	$f_3(V_1, V_2, V_3 + V_4) \stackrel{?}{=} f_2(f_1(V_5 + V_6 + V_7), f_1(f_1(V_8)))$	-	0	0	0	T/O	3757585	1	1
P_{29}	$f_4(V_1, V_2 + V_3, f_1(V_2 + V_4 + V_5), V_3) \stackrel{?}{=}$ $f_3(f_2(V_6, V_7) * V_6, V_8, f_1(f_1(V_9)))$	-	-	0	0	T/O	T/O	1	1
P_{30}	$f_5(V_1, V_2 + V_3 + V_4, f_2(V_5, f_1(V_3 + V_4)), V_6, f_1(V_7 + V_8)) \stackrel{?}{=}$ $f_4(f_2(V_9, V_{10}), V_{11}, f_1(f_1(V_9)), V_{12})$	-	-	0	0	T/O	T/O	1	1

Table 6.2: Experimental evaluation (abelian group)

both report the same number. However, both the *cr* and the *fast* algorithm are incomparable and *cr* reported less unifiers than *fast* in the unification problems P_9 and P_{10}, whereas *fast* reported less unifiers than *cr* in the unification problems $P_3, P_5, P_{18}, P_{19}, P_{20}$. As for the execution time, *cr* can beat both Maude and the *fast* algorithm for almost all the unification problems. Indeed, unification in the abelian group is so complex that neither Maude nor *fast* can terminate in most of the unification problems (e.g., $P_{22}, P_{23}, P_{24}, P_{25}$, and more), whereas *cr* did.

Our best contribution are the non-unifiable problems in the third block of Tables 6.1 and 6.2. Our new constructor-root unification algorithm immediately terminates, whereas neither Maude nor *fast* could, as shown in the unification problems $P_{11}, P_{12}, P_{13}, P_{14}, P_{15}, P_{26}, P_{27}, P_{28}, P_{29}, P_{30}$.

6.6 Conclusion and Future Work

The variant-based equational unification algorithm implemented in the most recent version of Maude, version 3.0, may compute many more unifiers than the necessary or may not be able to stop immediately. Constructor symbols are extensively used in computer science, but they have not been integrated into the variant-based equational unification procedure of Maude. In this paper, we have redefined the variant-based unification algorithm and our experiments

on some unification problems show an impressive speedup. Especially for non-unifiable problems, where many resources are wasted.

As far as we know, this is the only research line to reduce the number of variant unifiers. The closest work is to combine standard unification algorithms with variant-based unification, such as [50,51]. Note that the constructor variant unification of [91] is not connected to our work, since it is based on a new notion of *constructor variant*. This is just a step forward on developing new techniques for improving variant-based unification and we plan to reduce even more the number of variant unifiers.

Part III

Conclusions

CHAPTER 7

Conclusions and Future Work

In this chapter, we describe our conclusions and future work for each publication in this thesis.

- In Chapter 2, we were able to specify two APIs Yubikey & YubiHSM and analyze security properties automatically, going beyond the capabilities of previous work in the literature. This research raised different challenges to be solved, first, managing Lamport clocks, which give a partial ordering of events with minimal overhead, second, managing mutable memory in terms of synchronization messages, and, third, the inclusion of an ordering between some events. All this combined with the use of exclusive-or, which complicated things. In the future, we would like to analyze more APIs and even more challenging problems.

- In Chapter 3, we studied different approaches to overcome the problem that some cryptographic properties are difficult to handle by recent crypto tools. First, we provided a security protocol transformation, which relies on the notion of constructor term variant, that can safely get rid of all cryptographic properties exposed in the set of protocols we analyzed. The transformed protocols had a better analysis performance than the original ones. And second, we were able to specify and use the bilinear pairing theory, which is used in identity-based protocols.

Both our protocol transformation and the specification of bilinear pairing may be useful for other crypto tools. In the future, we would like to study other security protocols with cryptographic properties such as homomorphism.

- In Chapter 4, we expanded the capabilities of the crypto tool Maude-NPA by modeling the traveled distance time of a message between two participants. We encoded several security protocols of the family of Distance Bounding Protocols and, thanks to the connection of Maude-NPA to a Satisfiability Modulo Theories (SMT) that the Maude System support, we were to analyze several types of attacks, such as Mafia fraud. In the future, we would like to consider other families of protocols relying on distances or traveled time.

- In Chapter 5, we expanded the previous chapter to handle not only time but also space information. We used the Brands and Chaum protocol to show how this addition of a location in space is natural and equally subsumes our last specification of time. This new approach offers the possibility of describing secure localization protocols with complex location and time constraints. In the future, we would like to consider protocols that use Message Time of Arrival Codes (MTACs).

- In Chapter 6, we introduced a new variant-based equational unification algorithm that relies on the notion of constructor symbols. We considered unification problems for exclusive-or and abelian group theories which are the most complicated cryptographic theories in security protocol analysis showing an impressive speed-up in the experiments, especially for non-unifiable problems, where normally many resources are wasted. In the future, we would like to improve even more these techniques.

Bibliography

[1] The CVC4 SMT solver. Available online at: https://cvc4.github.io.

[2] Maude-NPA manual v3.1. Available online at: "http://maude.cs.illinois.edu/w/images/9/90/Maude-NPA_manual_v3_1.pdf".

[3] The Scyther-Proof Tool. Available online at: https://infsec.ethz.ch/research/software/scyther-proof.html.

[4] Wolfram Mathematica. Available online at: https://www.wolfram.com/mathematica.

[5] The Yices SMT solver. Available online at: https://yices.csl.sri.com/doc/index.html.

[6] Yubico AB. YubiKey Security Evaluation: Discussion of security properties and best practices. v2.0. Available online at: http://static.yubico.com/var/uploads/pdfs/Security_Evaluation_2009-09-09.pdf.

[7] Yubico customer list. Available online at: http://www.yubico.com/references.

[8] The Z3 SMT solver. Available online at: https://github.com/Z3Prover/z3.

[9] Specifications Overview, FIDO Alliance. Available online at: https://fidoalliance.org/specifications/overview/, Dec. 2015.

[10] S. S. Al-Riyami and K. G. Paterson. Tripartite authenticated key agreement protocols from pairings. In *IMA International Conference on Cryptography and Coding*, pages 332–359. Springer, 2003.

[11] M. Alpuente, S. Escobar, and J. Iborra. Termination of narrowing revisited. *Theoretical Computer Science*, 2009.

[12] M. Alpuente, S. Escobar, and J. Iborra. Modular termination of basic narrowing and equational unification. *Logic Journal of the IGPL*, 19(6):731–762, 2011.

[13] D. Aparicio-Sánchez, S. Escobar, C. Meadows, J. Meseguer, and J. Sapiña. Protocol analysis with time. In *International Conference on Cryptology in India*. Springer, 2020.

[14] G. Avoine, X. Bultel, S. Gambs, D. Gerault, P. Lafourcade, C. Onete, and J.-M. Robert. A terrorist-fraud resistant and extractor-free anonymous distance-bounding protocol. In *Proceedings of the ACM on Asia conference on computer and communications security*, pages 800–814, 2017.

[15] F. Baader and W. Snyder. Handbook of automated reasoning, chapter unification theory, 2001.

[16] K. Bae, S. Escobar, and J. Meseguer. Abstract logical model checking of infinite-state systems using narrowing. In *24th International Conference on Rewriting Techniques and Applications*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[17] D. Baelde, S. Delaune, I. Gazeau, and S. Kremer. Symbolic verification of privacy-type properties for security protocols with xor. In *IEEE 30th Computer Security Foundations Symposium (CSF)*, 2017.

[18] E. Barendsen. *Term rewriting systems*. Cambridge University Press, 2003.

[19] D. Basin, S. Capkun, P. Schaller, and B. Schmidt. Let's get physical: Models and methods for real-world security protocols. In *International Conference on Theorem Proving in Higher Order Logics*, pages 1–22. Springer, 2009.

[20] D. Basin, S. Capkun, P. Schaller, and B. Schmidt. Formal reasoning about physical properties of security protocols. *ACM Transactions on Information and System Security (TISSEC)*, 14(2):1–28, 2011.

[21] D. Basin, C. Cremers, J. Dreier, S. Meier, R. Sasse, and B. Schmidt. Tamarin Prover manual. Available online at: `https://tamarin-prover.github.io/manual/tex/tamarin-manual.pdf`.

[22] D. Basin, S. Mödersheim, and L. Vigano. Ofmc: A symbolic model checker for security protocols. *International Journal of Information Security*, 2005.

[23] F. Björck. Security DJ Blog: Increased security for Yubikey. Available online at: `http://web.archive.org/web/20100203110742/http://security.dj/?p=4`, Aug. 2009.

[24] F. Björck. Security DJ Blog: Yubikey Security Weaknesses. Available online at: `http://web.archive.org/web/20100725005817/http://security.dj/?p=154`, Feb. 2009.

[25] B. Blanchet. Modeling and verifying security protocols with the applied pi calculus and proverif. *Foundations and Trends® in Privacy and Security*, 1(1-2):1–135, 2016.

[26] C. Bouchard, K. A. Gero, C. Lynch, and P. Narendran. On forward closure and the finite variant property. In *International Symposium on Frontiers of Combining Systems*, pages 327–342. Springer, 2013.

[27] S. Brands and D. Chaum. Distance-bounding protocols. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 344–359. Springer, 1993.

[28] R. Burstall and J. Goguen. Algebras, theories and freeness: An introduction for computer scientists. In *Theoretical Foundations of Programming Methodology*, pages 329–350. Reidel, 1982. Marktoberdorf NATO Summer School, Advanced Study Institute Series, Volume C91.

[29] S. Capkun. Secure positioning and location-based security for iot and beyond. In *Proceedings of the Workshop on Attacks and Solutions in Hardware Security*, pages 81–81, 2018.

[30] S. Čapkun, L. Buttyán, and J.-P. Hubaux. Sector: secure tracking of node encounters in multi-hop wireless networks. In *Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*, pages 21–32, 2003.

[31] S. Capkun and J.-P. Hubaux. Secure positioning of wireless devices with application to sensor networks. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 3, pages 1917–1928, 2005.

[32] S. Capkun and J.-P. Hubaux. Secure positioning in wireless networks. *IEEE Journal on Selected Areas in Communications*, 24(2):221–232, 2006.

[33] A. Cholewa, J. Meseguer, and S. Escobar. Variants of variants and the finite variant property. Technical report, 2014.

[34] T. Chothia, J. De Ruiter, and B. Smyth. Modelling and analysis of a hierarchy of distance bounding attacks. In *27th {USENIX} Security Symposium)*, pages 1563–1580, 2018.

[35] M. Clavel, F. Durán, S. Eker, S. Escobar, P. Lincoln, N. Martı-Oliet, J. Meseguer, R. Rubio, and C. Talcott. Maude manual (version 3.2.1). *SRI International*, 2022.

[36] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude-A High-Performance Logical Framework: How to Specify, Program, and Verify Systems in Rewriting Logic*, volume 4350. Springer, 2007.

[37] K. Cohn-Gordon, C. J. F. Cremers, B. Dowling, L. Garratt, and D. Stebila. A formal security analysis of the signal messaging protocol. In *IEEE European Symposium on Security and Privacy, EuroS&P ,Paris, France*, pages 451–466, 2017.

[38] H. Comon-Lundh and S. Delaune. The finite variant property: How to get rid of some algebraic properties. In *International Conference on Rewriting Techniques and Applications*, pages 294–307. Springer, 2005.

[39] C. Cremers. *Scyther - Semantics and Verification of Security Protocols*. PhD thesis, Eindhoven University of Technology, 2006.

[40] C. Cremers. The Scyther Tool. Available online at: `https://people.cispa.io/cas.cremers/scyther/index.html`, 2014.

[41] A. Debant and S. Delaune. Symbolic verification of distance bounding protocols. In *Principles of Security and Trust-8th International Conference*, volume 11426, 2019.

[42] A. Debant, S. Delaune, and C. Wiedling. A symbolic framework to analyse physical proximity in security protocols. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[43] S. Delaune, S. Kremer, M. D. Ryan, and G. Steel. Formal analysis of protocols based on TPM state registers. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium, CSF, Cernay-la-Ville, France*, pages 66–80, 2011.

[44] J. Dreier, C. Duménil, S. Kremer, and R. Sasse. Beyond subterm-convergent equational theories in automated verification of stateful protocols. In *International Conference on Principles of Security and Trust*, pages 117–140. Springer, 2017.

[45] J. Dreier, L. Hirschi, S. Radomirovic, and R. Sasse. Automated unbounded verification of stateful cryptographic protocols with exclusive or. In *IEEE 31st Computer Security Foundations Symposium (CSF)*, 2018.

[46] F. Durán, S. Eker, S. Escobar, N. Martí-Oliet, J. Meseguer, R. Rubio, and C. Talcott. Programming and symbolic computation in maude. *Journal of Logical and Algebraic Methods in Programming*, 2020.

[47] F. Durán, S. Lucas, and J. Meseguer. Termination modulo combinations of equational theories. In *International Symposium on Frontiers of Combining Systems*. Springer, 2009.

[48] F. Durán and J. Meseguer. On the church-rosser and coherence properties of conditional order-sorted rewrite theories. *The Journal of Logic and Algebraic Programming*, 2012.

[49] F. Durán, J. Meseguer, and C. Rocha. Ground confluence of order-sorted conditional specifications modulo axioms. *Journal of Logical and Algebraic Methods in Programming*, 2020.

[50] A. K. Eeralla, S. Erbatur, A. M. Marshall, and C. Ringeissen. Rule-based unification in combined theories and the finite variant property. In *International Conference on Language and Automata Theory and Applications*. Springer, 2019.

[51] S. Erbatur, D. Kapur, A. M. Marshall, P. Narendran, and C. Ringeissen. Unification and matching in hierarchical combinations of syntactic theories. In *International Symposium on Frontiers of Combining Systems*. Springer, 2015.

[52] S. Escobar, D. Kapur, C. Lynch, C. Meadows, J. Meseguer, P. Narendran, and R. Sasse. Protocol analysis in maude-npa using unification modulo homomorphic encryption. In *Proceedings of the 13th international ACM SIGPLAN symposium on Principles and practices of declarative programming*, 2011.

[53] S. Escobar, C. Meadows, and J. Meseguer. A rewriting-based inference system for the nrl protocol analyzer and its meta-logical properties. *Theoretical Computer Science*, 2006.

[54] S. Escobar, C. Meadows, and J. Meseguer. Equational cryptographic reasoning in the maude-nrl protocol analyzer. *Electronic Notes in Theoretical Computer Science*, 2007.

[55] S. Escobar, C. Meadows, and J. Meseguer. Maude-npa: Cryptographic protocol analysis modulo equational properties. In *Foundations of Security Analysis and Design V*. Springer, 2009.

[56] S. Escobar, C. Meadows, J. Meseguer, and S. Santiago. State space reduction in the maude-nrl protocol analyzer. *Information and Computation*, 2014.

[57] S. Escobar, C. Meadows, J. Meseguer, and S. Santiago. Symbolic protocol analysis with disequality constraints modulo equational theories. In *Programming Languages with Applications to Biology and Security*. Springer, 2015.

[58] S. Escobar and J. Meseguer. Symbolic model checking of infinite-state systems using narrowing. In *Proceedings of the 18th International Conference on Rewriting Techniques and Applications (RTA'07)*, Lecture Notes in Computer Science, 2007.

[59] S. Escobar and J. Sapiña. Most general variant unifiers. *arXiv preprint arXiv:1909.08241*, 2019.

[60] S. Escobar, R. Sasse, and J. Meseguer. Folding variant narrowing and optimal variant termination. *The Journal of Logic and Algebraic Programming*, 2012.

[61] F. J. T. Fábrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Why is a security protocol correct? In *Proceedings. IEEE Symposium on Security and Privacy*, 1998.

[62] F. J. T. Fábrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of computer security*, 1999.

[63] A. González-Burgueño, D. Aparicio-Sánchez, S. Escobar, C. A. Meadows, and J. Meseguer. Formal verification of the yubikey and yubihsm apis in maude-npa. In *LPAR 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Awassa, Ethiopia*. EasyChair, 2018.

[64] A. González-Burgueño, S. Santiago, S. Escobar, C. A. Meadows, and J. Meseguer. Analysis of the IBM CCA Security API Protocols in Maude-NPA. In *Security Standardisation Research - First International Conference, SSR. Proceedings*, 2014.

[65] A. González-Burgueño, S. Santiago, S. Escobar, C. A. Meadows, and J. Meseguer. Analysis of the PKCS#11 API Using the Maude-NPA Tool. In *Security Standardisation Research - Second International Conference, SSR, Tokyo, Japan, Proceedings*, 2015.

[66] J. D. Guttman. Security goals and protocol transformations. In *Joint Workshop on Theory of Security and Applications*. Springer, 2011.

[67] G. P. Hancke and M. G. Kuhn. An rfid distance bounding protocol. In *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM'05)*. IEEE, 2005.

[68] J.-P. Jouannaud, C. Kirchner, and H. Kirchner. Incremental construction of unification algorithms in equational theories. In *International Colloquium on Automata, Languages, and Programming*. Springer, 1983.

[69] J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal of Computing*, 1986.

[70] A. Joux. A one round protocol for tripartite diffie–hellman. In *International algorithmic number theory symposium*. Springer, 2000.

[71] D. Kapur and P. Narendran. Matching, unification and complexity. *ACM SIGSAM Bulletin*, 1987.

[72] C. H. Kim, G. Avoine, F. Koeune, F.-X. Standaert, and O. Pereira. The swiss-knife rfid distance bounding protocol. In *International conference on information security and cryptology*. Springer, 2008.

[73] Y. Kim, A. Perrig, and G. Tsudik. Communication-efficient group key agreement. In *IFIP International Information Security Conference*. Springer, 2001.

[74] R. Künnemann. *Foundations for analyzing security APIs in the symbolic and computational model. Available online at: https://tel.archives-ouvertes.fr/tel-00942459/file/Kunnemann2014.pdf*. Theses, École normale supérieure de Cachan - ENS Cachan, 2014.

[75] R. Künnemann and G. Steel. YubiSecure? formal security analysis results for the Yubikey and YubiHSM. In *Revised Selected Papers of the 8th Workshop on Security and Trust Management (STM)*, Lecture Notes in Computer Science, Pisa, Italy, 2012. Springer.

[76] R. Küsters and T. Truderung. Using proverif to analyze protocols with diffie-hellman exponentiation. In *22nd IEEE Computer Security Foundations Symposium*, 2009.

[77] R. Küsters and T. Truderung. Reducing protocol analysis with xor to the xor-free case in the horn theory based approach. *Journal of Automated Reasoning*, 2011.

[78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 1978.

[79] P. Leu, M. Singh, M. Roeschlin, K. G. Paterson, and S. Čapkun. Message time of arrival codes: A fundamental primitive for secure distance measurement. In *IEEE Symposium on Security and Privacy (SP)*, 2020.

[80] S. Lucas and J. Meseguer. Normal forms and normal theories in conditional rewriting. *Journal of Logical and Algebraic Methods in Programming*, 2016.

[81] S. Mauw, Z. Smith, J. Toro-Pozo, and R. Trujillo-Rasua. Distance-bounding protocols: Verification without time and location. In *IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018.

[82] C. Meadows. The nrl protocol analyzer: An overview. *The Journal of Logic Programming*, 1996.

[83] C. Meadows, R. Poovendran, D. Pavlovic, L. Chang, and P. Syverson. Distance bounding protocols: Authentication logic analysis and collusion attacks. In *Secure localization and time synchronization for wireless sensor and ad hoc networks*. Springer, 2007.

[84] S. Meier, C. Cremers, and D. Basin. Strong invariants for the efficient construction of machine-checked protocol security proofs. In *23rd IEEE Computer Security Foundations Symposium*, 2010.

[85] S. Meier, B. Schmidt, C. Cremers, and D. Basin. The tamarin prover for the symbolic analysis of security protocols. In *International Conference on Computer Aided Verification*. Springer, 2013.

[86] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical computer science*, 96(1), 1992.

[87] J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *Proc. WADT'97*. Springer, 1998.

[88] J. Meseguer. Twenty years of rewriting logic. *The Journal of Logic and Algebraic Programming*, 2012.

[89] J. Meseguer. Strict coherence of conditional rewriting modulo axioms. *Theoretical Computer Science*, 2017.

[90] J. Meseguer. Symbolic reasoning methods in rewriting logic and maude. In *International Workshop on Logic, Language, Information, and Computation*. Springer, 2018.

[91] J. Meseguer. Variant-based satisfiability in initial algebras. *Science of Computer Programming*, 2018.

[92] J. Meseguer. Generalized rewrite theories, coherence completion, and symbolic methods. *Journal of Logical and Algebraic Methods in Programming*, 2020.

[93] S. Mödersheim and L. Vigano. The open-source fixed-point model checker for symbolic analysis of security protocols. In *Foundations of Security Analysis and Design V*. Springer, 2009.

[94] J. Munilla and A. Peinado. Distance bounding protocols for rfid enhanced by using void-challenges and analysis in noisy channels. *Wireless communications and mobile computing*, 2008.

[95] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The kerberos network authentication service (v5). Technical report, 2005.

[96] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving sat and sat modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(t). *J. ACM*, 2006.

[97] V. Nigam, C. Talcott, and A. A. Urquiza. Towards the automated verification of cyber-physical security protocols: Bounding the number of timed intruders. In *European Symposium on Research in Computer Security*. Springer, 2016.

[98] V. Nigam, C. Talcott, and A. A. Urquiza. Symbolic timed observational equivalence. *arXiv preprint arXiv:1801.04066*, 2018.

[99] D. Oswald, B. Richter, and C. Paar. Side-channel attacks on the yubikey 2 one-time password generator. In *Research in Attacks, Intrusions, and Defenses - 16th International Symposium, RAID, Rodney Bay, St. Lucia, Proceedings*, 2013.

[100] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 1998.

[101] A. Perrig, D. Song, R. Canetti, J. Tygar, and B. Briscoe. Timed efficient stream loss-tolerant authentication (tesla): Multicast source authentication transform introduction. *Request For Comments*, 2005.

[102] K. B. Rasmussen and S. Capkun. Realization of rf distance bounding. In *USENIX Security Symposium*, 2010.

[103] A. Riesco. Using big-step and small-step semantics in maude to perform declarative debugging. In *International Symposium on Functional and Logic Programming*. Springer, 2014.

[104] V. Rusu. Combining theorem proving and narrowing for rewriting-logic specifications. In *International Conference on Tests and Proofs*. Springer, 2010.

[105] S. Santiago, S. Escobar, C. A. Meadows, and J. Meseguer. Effective sequential protocol composition in maude-npa. *CoRR*, abs/1603.00087, 2016.

[106] R. Sasse, S. Escobar, C. Meadows, and J. Meseguer. Protocol analysis modulo combination of theories: A case study in maude-npa. In *International Workshop on Security and Trust Management*. Springer, 2010.

[107] P. Schaller, B. Schmidt, D. Basin, and S. Capkun. Modeling and verifying physical properties of security protocols for wireless networks. In *22nd IEEE Computer Security Foundations Symposium*, 2009.

[108] B. Schmidt, R. Sasse, C. Cremers, and D. Basin. Automated verification of group key agreement protocols. In *IEEE Symposium on Security and Privacy*, 2014.

[109] V. Shmatikov and M.-H. Wang. Secure verification of location claims with simultaneous distance modification. In *Annual Asian computing science conference*. Springer, 2007.

[110] S. Skeirik and J. Meseguer. Metalevel algorithms for variant satisfiability. *Journal of Logical and Algebraic Methods in Programming*, 2018.

[111] E. Tushkanova, A. Giorgetti, C. Ringeissen, and O. Kouchnarenko. A rule-based system for automatic decidability and combinability. *Science of Computer Programming*, 2015.

[112] L. Vamanu. Formal analysis of Yubikey. Master's thesis, École normale supérieure de Cachan (August 2011). Available oscynline atline at: `http://n.ethz.ch/~lvamanu/download/YubiKeyAnalysis.pdf`.

[113] F. Yang, S. Escobar, C. Meadows, J. Meseguer, and P. Narendran. Theories of homomorphic encryption, unification, and the finite variant property. In *Proceedings of the 16th International Symposium on Principles and Practice of Declarative Programming*, 2014.

[114] F. Yang, S. Escobar, C. A. Meadows, and J. Meseguer. Strand spaces with choice via a process algebra semantics. *CoRR*, 2019.

[115] F. Yang, S. Escobar, C. A. Meadows, J. Meseguer, and S. Santiago. Strand spaces with choice via a process algebra semantics. In J. Cheney and G. Vidal, editors, *Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming, Edinburgh, United Kingdom, September*. ACM, 2016.

[116] Yubico. The YubiKey Manual. Usage, configuration and introduction of basic concepts. Version: 3.4. Available online at: `https://goo.gl/zk5fbK`.

[117] Yubico. YubiHSM Manual v1.5. Available online at: `https://www.yubico.com/wp-content/uploads/2015/04/YubiHSM-Manual_1_5_0.pdf`.