UNIVERSITÉ DE LORRAINE | BIBLIOTHÈQUES UNIVERSITAIRES

AVERTISSEMENT
DROIT D'AUTEUR

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document. Toute contrefaçon, plagiat ou reproduction illicite encourt une poursuite pénale.

Contact bibliothèque : ddoc-theses-contact@univ-lorraine.fr
(Cette adresse ne permet pas de contacter les auteurs)

Code de la Propriété Intellectuelle. articles L 122. 4
Code de la Propriété Intellectuelle. articles L 335.2- L 335.10

UNIVERSITÉ DE LORRAINE
34 COURS LÉOPOLD — CS 25233
54052 NANCY CEDEX
TÉL : 03 54 50 54 00
CONTACT@UNIV-LORRAINE.FR
WWW.UNIV-LORRAINE.FR

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Formal Verification in Practice: Real-World Case Study and Enhanced Support for AC Operators in Tamarin

THÈSE

présentée et soutenue publiquement le ***11/12/2025***

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Elise Klein

Composition du jury

Président :	David Baelde	Professeur, ENS Rennes, France
Rapporteurs :	Yannick Chevalier	Maître de conférences, Université de Toulouse, France
	Barbara Fila	Maîtresse de conférences, INSA de Rennes, France
Examinateurs :	Ioana Boureanu	Professeure, University of Surrey, Royaume-Uni
	Claudia Ignat	Directrice de recherche, Centre Inria de l'Université de Lorraine, France
Encadrants :	Jannik Dreier	Maître de conférences, Université de Lorraine, France
	Steve Kremer	Directeur de recherche, Centre Inria de l'Université de Lorraine, France

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Remerciements

Quand on fait une thèse, il arrive souvent qu'on se sente seul au monde. Surtout quand on est face à son manuscrit et que chaque page est une montagne. Mais parfois, on prend du recul et on remarque le soutien que l'on a reçu.

Ma première pensée va à mes encadrants, Steve et Jannik. Je n'aurais pas pu rêver mieux comme encadrants, vous avez toujours été là et vous vous êtes toujours adaptés à mon travail, même quand vos agendas de ministre n'étaient pas conciliants. Je n'ai pas été une doctorante facile et je m'en veux beaucoup. Cela m'est arrivé de vouloir tout envoyer balader, mais je me souvenais alors de tout ce que vous faisiez pour moi et je repartais. Merci pour tout.

Maïwenn, ma jumelle de thèse. On a commencé en même temps, avec les mêmes encadrants. On a pesté souvent toutes les deux sur un certain singe et sa base de code, mais ça faisait du bien. Tu étais mon rappel santé, j'étais ton rappel administratif, notre manière à nous de nous rappeler de penser un peu à nous en dehors de la thèse. On a partagé beaucoup de choses durant nos différents voyages, formations, simples après-midis... Et promis, la prochaine fois qu'on prendra l'avion ensemble, ça se passera mieux.

Et puis à vous les doctorants du bureau des doctorants, ou simplement tout ceux qui sont passé par le B272, merci à vous pour nos franches rigolades. Ca va me manquer d'écouter vos ragots, de planquer des post-it dans des endroits improbables, de décorer les portes avec des traits d'humour de tout genre, de scruter les habitudes des autres pour faire des bingos, de méditer devant la vue de la baie vitrée, de pester sur les écrans qui fonctionnent pas, de parler boulot (des fois)... Et tant d'autres choses qui font que mes années de thèse me sont précieuses malgré tout.

Je souhaite également remercier toute l'équipe Pesto pour m'avoir accueillie toutes ces années. Nos discussions plus ou moins caustiques à table ou en ligne étaient des plaisirs quotidiens.
Je remercie Isabelle pour la chaleur dont elle fait preuve envers tout le monde. Tu as sûrement sauvé beaucoup de mes journées.
Je remercie Loïc et Lionel de l'accueil qui ne manquaient jamais de m'ouvrir la porte le matin. Commencer la journée par une petite attention et un bonjour chaleureux, c'est simple mais indispensable.
Je remercie également les collègues de TELECOM Nancy, ma plus grande sources de ragots en tout genre. C'était tellement bien de travailler avec vous. J'ai adoré être une chipie avec toi, Christophe.

Quand on est en thèse, il ne faut pas non plus oublier qu'on a une vie personnelle. Merci à la team du dimanche soir de m'avoir obligée à sociabiliser ces dernières années. Continuez, s'il vous plait, je vous nourris en échange.

Avant de conclure, je souhaite également remercier mes parents. Vous m'avez suivi de loin, je pense que vous ne compreniez pas trop pourquoi je m'étais lancée là-dedans (non désolé, je ne menais pas de cyber-guerre contre les russes...), mais vous m'avez toujours fait confiance pour mes choix de vie et je ne vous remercierez jamais assez pour ça.

Le mot de la fin te revient, Alex. Le point d'orgue de ces remerciements ne peut aller que vers toi, qui m'a donné cette idée saugrenue de faire une thèse et qui a toujours toujours cru en moi, surtout quand je pensais être une incapable finie. Tu as donné beaucoup de ton temps pour me sortir la tête hors de l'eau (ou pour sauver mon ordinateur) et pour m'apprendre des trucs techniques digne des plus grands magiciens. Même si je suis parfois pénible, tu as toujours la patience pour me faire relativiser et pour m'aider à trouver des solutions. Je n'aurais jamais pu réussir sans ton soutien. Merci.

Contents

Contents iii

1. Introduction 1
1.1. Context: Protocol & Security . . . 1
1.1.1. The Riddle of the Kleptomaniac Messengers . . . 1
1.1.2. Cryptographic protocols . . . 2
1.1.3. Security Properties . . . 4
1.2. Symbolic Model . . . 6
1.2.1. Protocol model . . . 6
1.2.2. Dolev-Yao Intruder . . . 7
1.2.3. Properties and verification . . . 7
1.3. Contributions . . . 8
1.4. Outline . . . 9

I. Formal analysis using verification tools 11

2. Symbolic Analysis using Sapic$^+$ 12
2.1. Syntax & Semantics . . . 12
2.1.1. Messages as terms . . . 12
2.1.2. Protocols as processes . . . 16
2.1.3. Properties as first-order formulas . . . 20
2.2. Sapic$^+$ translation to other tools . . . 23
2.2.1. Tamarin Prover . . . 24
2.2.2. ProVerif . . . 25
2.2.3. DeepSec . . . 26

3. Application: the case of LAKE-EDHOC 27
3.1. Context . . . 27
3.1.1. Related work . . . 28
3.1.2. Contributions . . . 28
3.2. Presentation of the EDHOC protocol . . . 29
3.2.1. Protocol outline . . . 30
3.2.2. Claimed properties . . . 32
3.3. Models for automated verification . . . 34
3.3.1. Protocol and properties modeling . . . 34
3.3.2. Advanced primitive modelings . . . 37
3.4. Results of the analysis . . . 39
3.4.1. Methodology . . . 40
3.4.2. Summary of the automated analysis of draft 12 . . . 42
3.4.3. Weaknesses in the Key Derivation . . . 43
3.4.4. Transcript collision attacks . . . 45
3.4.5. Key and IV-reuse . . . 46
3.4.6. Privacy leak . . . 47
3.4.7. Non-repudiation . . . 47
3.4.8. Analysis of draft 14 . . . 48
3.5. Future work . . . 48

II. The challenge of Associative and Commutative symbols in Tamarin **49**

4. Focus on the Tamarin Prover **50**
4.1. Syntax & Semantics . . . 50
4.1.1. Messages as terms: reminder and singularities . . . 50
4.1.2. Protocols as rewrite systems . . . 52
4.1.3. Properties as first-order formulas . . . 57
4.2. Intruder deduction and automated reasoning . . . 58
4.2.1. Internal representation: Dependency graphs . . . 58
4.2.2. Intruder Rules . . . 60
4.2.3. Internal representation: Normal Dependency Graphs . . . 63
4.2.4. Automated reasoning . . . 64

5. Handling User-Defined AC Operators in Tamarin **69**
5.1. Context: AC symbols in Tamarin . . . 69
5.1.1. Related work . . . 70
5.1.2. Contributions . . . 70
5.2. The No Deconstruction Chain property . . . 72
5.3. Verifying the No Deconstruction Chain property . . . 73
5.3.1. Algorithm to check the NDC property . . . 74
5.3.2. Verifying deduction . . . 74
5.4. Pruning constructor chains . . . 79
5.4.1. Constructor chains issue . . . 79
5.4.2. Extending $\rightsquigarrow_p$ with $\mathbf{DG}_{+,\perp}$ . . . 80
5.5. Evaluation and Case studies . . . 83
5.5.1. Impact of $\mathbf{DG}_{+,\perp}$ . . . 84
5.5.2. Existing AC symbols . . . 85
5.5.3. Improvement of existing case studies . . . 86
5.5.4. Toy voting system based on distributed decryption . . . 87
5.5.5. Exponentiation Mixnet . . . 88
5.5.6. Summary . . . 89
5.6. Future Work . . . 90

6. Conclusion & Future work **91**
6.1. Future Work . . . 91

Bibliography **94**

Appendix **99**

A. Appendix for formal analysis background **100**
A.1. Model in Sapic$^+$ of $CRXOR$ protocol . . . 100
A.2. Model in ProVerif of $CRXOR$ protocol . . . 101
A.3. Model in Tamarin of $CRXOR$ protocol . . . 102

B. Appendix for EDHOC analysis **103**
B.1. Detailed analysis results . . . 103

C. Appendix for user-defined AC symbols in Tamarin **106**
C.1. Proof of Theorem 5.2.1 . . . 106
C.2. Modified Constraint-Solving Rules . . . 107
C.2.1. Extending $\rightsquigarrow_p$ with **N14** . . . 107
C.2.2. Extending $\rightsquigarrow_p$ with $\mathbf{DG}_{+,\perp}$ . . . 108

D. Résumé en français **111**
D.1. Contexte : Protocole & Sécurité . . . 111
D.1.1. L'énigme des messagers cleptomanes . . . 111
D.1.2. Protocoles cryptographiques . . . 112
D.1.3. Propriétés de sécurité . . . 114
D.2. Modèle symbolique . . . 115
D.2.1. Modèle de protocoles . . . 116
D.2.2. Adversaire de Dolev-Yao . . . 116
D.2.3. Propriétés et vérification . . . 117
D.3. Contributions . . . 118

1. Introduction

1.1. Context: Protocol & Security

1.1.1. The Riddle of the Kleptomaniac Messengers

To illustrate the various concepts discussed in this thesis, consider the following situation: Princess Alice and Prince Bob live in an imaginary world where communication is possible through postal service only. The messengers responsible for delivering letters and parcels are all very intrusive: they want to know the contents of everything entrusted to them, and if it serves their interests, they are not above keeping it for themselves. Alice and Bob, living in distant regions, are therefore forced to rely on these kleptomaniac messengers for their exchanges. However, they do not want their sweet words or precious gifts to be read by strangers or stolen. They thus need a strategy to avoid such a scenario, knowing that each of them possesses a padlock and its corresponding key.

This small logical riddle is a classic example: to solve the problem, Alice places her letter in a box, locks it with her padlock, and entrusts it to a messenger. Unable to open the box, the messenger delivers it to Bob. Bob, also unable to open the box, adds his own padlock and sends it back to Alice. Alice, who has the key to her own padlock, removes it and sends the box back to Bob. Now, only Bob's padlock remains on the box, and he can open it to read Alice's precious message.

This trick between the two lovers can be seen as a **protocol**. They have previously agreed on the steps to follow (illustrated in Figure 1.1):

1. Alice places her message in a box, locks it with her padlock, and sends it to Bob.
2. Bob receives a box with a padlock on it. He adds his own padlock and sends it back to Alice.
3. Alice receives a box with two padlocks. She removes her own padlock and sends the box back to Bob.
4. Bob removes his padlock and reads the content.

This actions sequence relies on several implicit assumptions: Alice and Bob each have a unique padlock, and therefore a distinct key; only the key that matches a padlock can open it; the padlocks cannot be forced or destroyed; keys cannot be reproduced without having been seen, etc.
Let us add three more assumptions: (1) the protocol set up by the couple is not secret, it is known to everyone (other people might want to protect their letter too!); (2) everyone in the kingdom has a key-padlock pair; (3) Alice and Bob can only recognize their own padlocks (they cannot be sure who owns the other padlock).

With these new assumptions, a particularly cunning messenger, whom we will call Charlie, manages to carry out the following attack by leveraging his knowledge of the protocol[1]. During an exchange between Alice and

1.1 Context: Protocol & Security 1
1.1.1 The Riddle of the Kleptomaniac Messengers 1
1.1.2 Cryptographic protocols . 2
1.1.3 Security Properties 4
1.2 Symbolic Model 6
1.2.1 Protocol model 6
1.2.2 Dolev-Yao Intruder 7
1.2.3 Properties and verification 7
1.3 Contributions 8
1.4 Outline 9

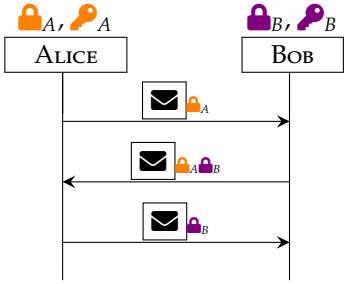


Figure 1.1.: Protocol used by Alice and Bob

1: assumption (1)

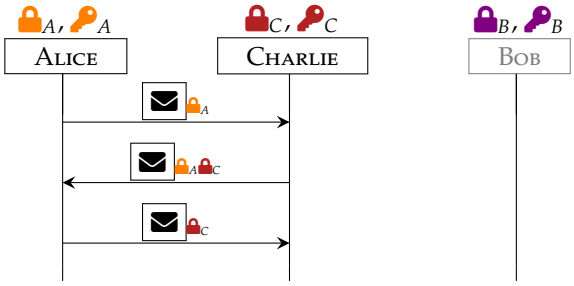


Figure 1.2.: Attack performed by Charlie

Bob, he acts as illustrated in Figure 1.2:

1. Charlie waits for Alice to hand him the locked box intended for Bob.
2. Instead of delivering it to Bob, he keeps the box, adds his own padlock[2], and sends it back to Alice.
3. Alice receives a box with two padlocks[3]. She removes her own and sends it back to Charlie to deliver it to Bob.
4. Charlie keeps the box: only his padlock remains, and he can open it and read or steal the contents.

2: Thanks to assumption (2), we know that Charlie has his own key-padlock pair.

3: According to assumption (3), Alice can recognize her own padlock but cannot identify the second one.

Alice followed the protocol to the letter, but Bob never had the box in his hands: Charlie has found an **attack**. There was an issue with **recipient authentication**, as Alice had no way of ensuring that she was truly sending her parcel to Bob, which also compromised the **confidentiality** of the content, since Charlie was able to access it.

1.1.2. Cryptographic protocols

As intuitively described in Section 1.1.1, a protocol is a set of rules describing how several entities exchange messages, and how they secure this exchange using *cryptographic primitives*.

In a contemporary and practical context, protocols play a central role at many levels of our daily lives. The EMV (Europay Mastercard Visa) protocol [1, 2] enables communication between the bank associated with a credit card and a payment terminal to authorize or deny a transaction. Messaging applications such as Signal [3] rely on protocols to define how messages are exchanged between multiple users over the Internet. Likewise, the TLS 1.3 protocol [4] and the authentication protocols of the 5G-AKA (Authentication and Key Agreement) standard [5, 6] are essential for the operation of mobile networks and secure internet access. Protocols can also play a role in democratic processes, notably through electronic voting systems such as Belenios [7] or that of Swiss Post [8].

These protocols are critical to the functioning of modern societies and must provide strong guarantees. For example, a credit card payment protocol must ensure that the transaction amount and the recipient account remain correct and cannot be altered during the process. A messaging protocol must preserve the confidentiality of conversations, while an electronic voting protocol must guarantee both the secrecy and the proper casting of each vote, and much more (anonymity, etc.). Such security guarantees are achieved through the use of **cryptographic primitives**.

[1]: Ruiter et al. (2012), *Formal Analysis of the EMV Protocol Suite*
[2]: Basin et al. (2021), *The EMV Standard: Break, Fix, Verify*
[3]: Katriel et al. (2020), *A Formal Security Analysis of the Signal Messaging Protocol*
[4]: Cremers et al. (2017), *A Comprehensive Symbolic Analysis of TLS 1.3*
[5]: Koutsos (2019), *The 5G-AKA Authentication Protocol Privacy*
[6]: Basin et al. (2018), *A Formal Analysis of 5G Authentication*
[7]: Cortier et al. (2019), *Belenios: a simple private and verifiable electronic voting system*
[8]: Cortier et al. (2022), *A privacy attack on the Swiss Post e-voting system*

If these guarantees are not properly ensured, vulnerabilities can be discovered and exploited by malicious actors, potentially leading to severe consequences. Numerous weaknesses have been identified in the EMV protocol [2, 9, 10], including one that allows bypassing PIN verification using a smartphone application, enabling unauthorized transactions without knowledge of the card's code. In another domain, security flaws were uncovered in the electronic voting system used for the 2022 French legislative elections, raising concerns about the integrity of the electoral process [11]. It is therefore essential to carefully design protocols and rigorously analyze them in order to detect potential vulnerabilities as early as possible.

[2]: Basin et al. (2021), *The EMV Standard: Break, Fix, Verify*
[9]: Ferradi et al. (2016), *When organized crime applies academic results: a forensic analysis of an in-card listening device*
[10]: Basin et al. (2023), *Inducing Authentication Failures to Bypass Credit Card PINs*
[11]: Debant et al. (2023), *Reversing, Breaking, and Fixing the French Legislative Election E-Voting Protocol*

When designing a protocol, several key aspects must be carefully considered, each requiring distinct areas of expertise. A few notable examples include:

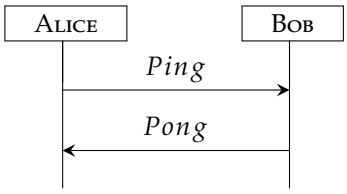
- **Cryptography**: Which encryption algorithms, hash functions, or signature schemes should be used? In short, which mathematical tools are most appropriate for the protocol's intended application domain?
- **Specification**: What sequence of messages should be exchanged between participants (how many, what kind of information, encrypted or not)? How many entities are involved, and what are their roles? At this stage, cryptographic primitives are assumed to be ideal; the focus is on the interaction logic among the actors.
- **Implementation**: How is the protocol implemented? In which language? On what types of machines? Under what performance constraints? Does the implementation strictly follow the specification, or does it introduce adaptations, potentially at the expense of some security guarantees ensured by the original specification?

Each of these elements is essential. This thesis focuses primarily on specification-related questions, that is, the underlying logic of the protocol.

To analyze a protocol, we must first describe it. Several standard methods exist for representing a protocol, depending on whether the goal is to provide a concise visual overview or a detailed formal specification. The first approach is a textual, technical documentation, similar to what was presented in the previous section, but with more comprehensive detail and often made publicly available. This type of specification is common for standardization documents such as RFCs (Request For Comments). The second method is a highly compact way of describing message exchanges, referred to as Alice & Bob notation. It provides a minimalistic view of the interactions between entities. For example, if Alice initiates an exchange by sending "Ping" and Bob replies with "Pong," we write:

$$\begin{aligned} Alice &\rightarrow Bob : Ping \\ Bob &\rightarrow Alice : Pong \end{aligned}$$

Finally, a protocol can also be represented visually using a diagram, which helps to clarify the flow of messages at a glance. This visual representation will be used throughout the body of this thesis to describe the protocols under study. For instance, the simple protocol above would be depicted as shown in the adjacent figure.



Running example: $CRXOR$ **protocol**

To illustrate the fundamental notions introduced in Chapters 2 and 4, we will use the following running example.
The $CRXOR$ protocol is a simple RFID[4] protocol that operates using so-called RFID chips and the *XOR operator* $\oplus$. The protocol involves two agents: the first, called the Initiator, and the second, the Responder. These two parties share a common *secret key* k. The Initiator begins by sending a *nonce* n_a (i.e., a fresh random value) to the Responder. The Responder replies with a new nonce n_b and a hash of the XOR of both nonces and the shared key k.
This protocol can be represented using Alice & Bob notation as follows:

$$\begin{aligned} Initiator &\rightarrow Responder: \quad n_a \\ Responder &\rightarrow Initiator: \quad \langle \mathsf{h}(n_a \oplus k \oplus n_b), n_b \rangle \end{aligned}$$

Figure 1.3.: $CRXOR$ protocol in Alice & Bob notation

In a typical application, the Initiator corresponds to an RFID reader device, and the Responder is the RFID chip. At the end of the exchange, the Initiator computes the XOR of its original nonce n_a, the shared key k, and the received nonce n_b, then applies the hash function h. If the result does not match the hash received from the Responder, the Initiator aborts the exchange.
This protocol contains a vulnerability, which will be discussed in Example 2.1.6.

4: Radio Frequency Identification

The notions of hash functions and the XOR operator are explained in Section 1.1.3.

1.1.3. Security Properties

As previously mentioned, cryptographic protocols are expected to ensure certain **security properties**, which may vary depending on the protocol's application. The most common properties include:

- **Authentication**: Each participant must be able to verify the identity of the other. Referring back to the riddle introduced in Section 1.1.1, Alice needs assurance that her messages are indeed being delivered to Bob. For his part, Bob must be confident that the box he receives truly came from Alice and not from an attacker impersonating her. This notion is also essential in banking protocols, where the bank must ensure that a card is authorized to charge the requested amount from the corresponding account.
- **Confidentiality**: When a message is encrypted, it must remain secret and accessible only to authorized parties. This property is fundamental to nearly all cryptographic protocols: ensuring that private messages remain private in messaging apps, preventing access to bank account balances or PIN codes from intercepted messages of the protocol, preserving the secrecy of electronic votes, and so on.
- **Anonymity**: Participants may wish to keep their identity hidden, even from the authentication server. This is the case in systems like IBM's IDEMIX [12] or some electronic voting protocols. Anonymity is often coupled with *untraceability*, which ensures that multiple sessions cannot be linked to the same user.

[12]: Camenisch et al. (2010), *Specification of the identity mixer cryptographic library*

Other security properties are more specific to particular application domains. For instance, in the context of electronic voting, it may be necessary to ensure *eligibility*: the protocol must guarantee that only ballots cast by legitimate voters (based on age, nationality, registration, etc.) are counted, and that each voter can vote only once.
To achieve these guarantees, cryptographic protocols rely on **cryptographic primitives**, a few of which are described below:

- **Symmetric encryption**: This is analogous to the locked box in the riddle in Section 1.1.1. However, it requires that the communicating parties already share a secret key (e.g., obtained beforehand via another protocol). The sender encrypts the message with this key to produce a *ciphertext*, and the recipient decrypts it using the same key to recover the *plaintext*. Only participants possessing the secret key can decrypt messages encrypted with it.
- **Asymmetric encryption**: Similar in principle to symmetric encryption, but each participant possesses a key pair: a *public key*, which can be widely shared, and a *private key*, which is different and remains secret. A sender can encrypt a message using the recipient's public key, which only the recipient can decrypt using their private key. Returning to the messenger analogy, Alice would use one of Bob's padlocks (his public key) to secure her box, and only Bob, with his unique key, could open it, effectively blocking Charlie's attack.
- **Cryptographic hash functions**: These functions associate each input x with a unique output $\mathsf{h}(x)$, ideally in a way that makes it infeasible to find another input $x' \neq x$ such that $\mathsf{h}(x) = \mathsf{h}(x')$ (this would be a *collision*). While computing $\mathsf{h}(x)$ is easy, retrieving x from $\mathsf{h}(x)$ is computationally hard.
- **Digital signatures**: These enable a sender to prove the origin and authenticity of a message. Like asymmetric encryption, they rely on a public/private key pair. The sender signs the message using their private key (which is exclusively own by the signer), and anyone can verify the signature using the corresponding public key. A variant known as a *blind signature*, used notably in some voting systems, allows a message to be signed without revealing its content, similar to placing a letter inside an envelope and having someone sign it through the envelope.
- **Diffie-Hellman exponentiation**: This primitive is a modular exponentiation in a cyclic group, using a public generator g. It enables two parties to compute a shared key without revealing their respective secrets. For example, if Alice chooses a secret x and Bob chooses a secret y, they publicly exchange g^x and g^y. Then, each party can compute g^{x*y} on their side. This shared value can be used to derive a common secret key. The individual private values x and y remain unknown to eavesdroppers. Because of the computational Diffie-Hellman problem (given g^x and g^y it is hard to compute g^{x*y}), an attacker knowing only g^x and g^y cannot derive the shared value.
- **XOR operator**: The XOR (exclusive or) operation $\oplus$ is both associative and commutative and has the useful property that $x \oplus x = 0$, allowing values to be "canceled out". Let us revisit the messenger riddle: in the solution devised by the couple, we replace the key-lock pairs used by Alice and Bob with numerical values (a for Alice,

b for Bob). At the beginning, when Alice locks the box containing the message m, she instead computes $m \oplus a$. Bob receives this value and applies his own transformation, computing $m \oplus a \oplus b$. Alice then decrypts it by computing $m \oplus a \oplus b \oplus a = m \oplus b$. Finally, when Bob receives this last value, he computes $m \oplus b \oplus b = m$ and recovers Alice's original message.

1.2. Symbolic Model

One way to analyze the security of a cryptographic protocol is to adopt the symbolic approach of formal methods.

1.2.1. Protocol model

In a symbolic model, participants, messages, and cryptographic primitives are represented abstractly, focusing on their logical interactions rather than cryptographic details. Specifically, a message is modeled as a *term*, which is either a variable x, a name n or a function symbol applied on other terms. This constrasts with computational model, where messages are represented as sequences of bits (a *bitstring*), which would be closer to the implementation level. Similarly, cryptography is assumed to be perfect (called the *perfect cryptography assumption*), and its behavior is described using *equations* and *function symbols*. For example:

- The model of **symmetric encryption** involves two function symbols (`enc` for encryption and `dec` for decryption) and a single equation:

$$\mathtt{dec}(\mathtt{enc}(m, k), k) = m$$

  This means that if a message m is encrypted with a key k ($\mathtt{enc}(m, k)$), the only way to recover it is to decrypt it (`dec`) with the same key k.
- The model of the **XOR operator** uses two function symbols ($\oplus$ for the operator itself and `zero` to denote the constant 0, the neutral element), along with the following equations:

$$\begin{aligned} a \oplus (b \oplus c) &= (a \oplus b) \oplus c & a \oplus b &= b \oplus a \\ a \oplus a &= \mathtt{zero} & a \oplus \mathtt{zero} &= a \end{aligned}$$

  These equations capture, for any message, the associativity and commutativity of the operator, as well as its ability to "cancel" a message, as described earlier.

This abstraction of cryptographic primitives greatly facilitates automated verification using tools (see Section 1.2.3). This model can also be refined by adding more complex equations to better reflect real-world cryptographic behavior.

In this thesis, we describe two approaches to symbolically model a protocol: the first (Chapter 2) relies on the applied pi-calculus [13], and the second (Chapter 4) uses multiset rewriting rules [14]. Both formalisms allow us to describe the agents participating in the protocol, and how they interact.

[13]: Abadi et al. (2018), *The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication*

[14]: Cervesato et al. (1999), *A meta-notation for protocol analysis*

1.2.2. Dolev-Yao Intruder

Symbolic models not only allow us to model and analyze cryptographic protocols, but also to represent the threat model, i.e., the attacker we aim to defend against. The more capabilities the attacker has to interfere with the protocol, the harder it becomes to ensure security properties. A common distinction is made between a *passive* attacker and an *active* attacker.

A *passive* attacker, on the one hand, observes communications without interfering with their flow. They merely **eavesdrop** on message exchanges: they read unencrypted messages to extract information, and copy all messages (even those they cannot understand). Their goal is to gather sensitive data (passwords, keys, private content) in order to analyze or break cryptographic mechanisms, while remaining undetected.

An *active* attacker, on the other hand, interacts directly with the system and disrupts message exchanges. They may **intercept** and **modify** messages (for example, using information previously obtained through eavesdropping), as well as **delete**, **delay**, or **inject** new messages of their own. The attacker's goal may be to obtain secret data (keys, messages, etc.), impersonate a legitimate participant, or bypass security protections. A classic example of an active attack is the *man-in-the-middle* attack, where the attacker controls the network on which parties communicate and may alter the messages to impersonate one participant to the other.

In symbolic approaches, the attacker is typically modeled following the seminal work of Dolev and Yao [15] from the early 1980s. In the so-called Dolev-Yao model, the attacker is considered to be **active and powerful**, with full control over the network and all the aforementioned capabilities. The attacker can also operate on cryptographic primitives and modeled function symbols, but only within the constraints of the chosen equations.

[15]: Dolev et al. (1983), *On the security of public key protocols*

1.2.3. Properties and verification

Security properties, such as those presented in Section 1.1.3, are modeled using logical formulas. A security property is said to hold if the logical formula that models it is true for all possible executions of the protocol. This verification is always performed in the presence of an attacker. Since some tools handle an unbounded number of sessions, the verification problem becomes undecidable, and analyses may not terminate.

Automated, symbolic protocol analysis is a successful approach for finding attacks or proving their absence. Formal security proofs capturing precise security and threat model definitions allow to identify whole families of vulnerabilities. It indeed provides a good compromise, balancing the accuracy of the model and the level of automation through tool support, allowing to analyse large, real-world protocols while taking advanced security properties and threat models into account.

Verification tools

State-of-the-art verification tools, such as ProVerif [16, 17] and the Tamarin Prover [18, 19], are now capable of scaling to industrial-sized, real-world protocols.

These two tools are the most widely used protocol analyzers in this field of research and have been successfully applied to the analysis of protocols such as TLS [4, 20], authentication protocols from the 5G standard [6], the Signal instant messaging protocol [21], EMV payment protocols [2, 22], as well as the OPC-UA standard for industrial control systems [23].

DeepSec [24] is another symbolic tool, specialized in verifying equivalence properties, and designed specifically for an bounded number of sessions, unlike ProVerif and Tamarin.

Finally, we can mention Sapic$^+$ [25], a platform integrated into Tamarin, which centralizes a single protocol model and allows exporting it to all three aforementioned tools (ProVerif, Tamarin, and DeepSec), used as back-ends.

[18]: Schmidt et al. (2012), *Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties (extended version)*

[19]: Basin et al. (2022), *Tamarin: Verification of Large-Scale, Real-World, Cryptographic Protocols*

[4]: Cremers et al. (2017), *A Comprehensive Symbolic Analysis of TLS 1.3*

[20]: Bhargavan et al. (2022), *A Symbolic Analysis of Privacy for TLS 1.3 with Encrypted Client Hello*

[6]: Basin et al. (2018), *A Formal Analysis of 5G Authentication*

[21]: Kobeissi et al. (2017), *Automated Verification for Secure Messaging Protocols and Their Implementations: A Symbolic and Computational Approach*

[2]: Basin et al. (2021), *The EMV Standard: Break, Fix, Verify*

[22]: Radu et al. (2022), *Practical EMV Relay Protection*

[23]: Diemunsch et al. (2025), *A Comprehensive Formal Security Analysis of OPC UA*

[24]: Cheval et al. (2018), *DEEPSEC: Deciding Equivalence Properties in Security Protocols Theory and Practice*

[25]: Cheval et al. (2022), *Sapic$^+$: protocol verifiers of the world, unite!*

1.3. Contributions

We present in this thesis two main contributions:

- **Comprehensive formal analysis of the EDHOC protocol**: EDHOC (Ephemeral Diffie Hellman Over COSE) is a key exchange protocol proposed by IETF's Lightweight Authenticated Key Exchange (LAKE) working group. Its design focuses on messages of small sizes to be suitable for constrained IoT communication technologies. We provide an in-depth formal analysis of EDHOC-draft version 12 (published in October 2021), taking the different proposed authentication methods and various options into account. For our analysis we use the Sapic$^+$ protocol platform that allows to compile a single specification to 3 state-of-the-art protocol verification tools (ProVerif, Tamarin and DeepSec) and take advantage of the strengths of each of the tools. In our analysis we consider a large variety of compromise scenarios, and also exploit results that allow to model existing weaknesses in cryptographic primitives, relaxing the perfect cryptography assumption, common in symbolic analysis. While our analysis confirmed security for the most basic threat models, a number of weaknesses were uncovered in the current design when more advanced threat models were taken into account. These weaknesses have been acknowledged by the LAKE working group and the mitigations we propose (and prove secure) have been included in version 14 of the draft, published in May 2022.
- **Handling user-defined associative and commutative (AC) operators in Tamarin**: We extend the Tamarin Prover to allow users to specify AC symbols as part of their user-defined equational theory. To avoid termination issues, we propose a novel property that bounds certain deduction chains. We use Tamarin's deduction algorithm to verify whether this bound is indeed correct for a given equational theory, generalizing previous work. Moreover, our work

COSE is for CBOR Object Signing and Encryption. CBOR is for Concise Binary Object Representation

allowed to identify that Tamarin's built-in XOR used an optimization that may lead to incorrect results, and would have influenced our results as well. Simply disabling the optimization results in non-termination on a large fraction of examples from the Tamarin repository. We therefore design a new constraint reduction rule that discards some redundant constructor chains, and prove its correctness. We illustrate the effectiveness and efficiency of our approach using different examples, including XOR, multisets, re-encryption, Diffie-Hellman exponentiation, and distributed decryption. It turns out that our property holds for all these examples, and that, compared to Tamarin 's built-in support for XOR or multisets, our approach only incurs a small overhead on startup when checking the property, but otherwise has similar performance on most examples. Re-encryption and distributed decryption were previously out of scope, and our model for Diffie-Hellman exponentiation, albeit simpler than the built-in one, allows for the exponentiation symbol to be reused in further equations, which is not allowed for the built-in one.

1.4. Outline

This thesis is organized into two main parts. The first introduces formal analysis through symbolic models, focusing on the Sapic$^+$ platform and applying this approach to the analysis of a real-world protocol. The second part focuses on the Tamarin Prover, detailing its core concepts and internal mechanisms, and presenting a new feature implemented in the tool.

Formal analysis using verification tools Part I presents the foundations of formal analysis.
Chapter 2 introduces the core concepts of symbolic modeling using Sapic$^+$, a platform that centralizes protocol modeling and exports it to three state-of-the-art symbolic tools. This chapter details the modeling of messages, processes, and security properties, and provides an overview of the supported backends: ProVerif, Tamarin, and DeepSec.
Chapter 3 applies this concepts to analyze the LAKE-EDHOC protocol using Sapic$^+$. We describe the models used, the methodology followed, and the vulnerabilities uncovered during the analysis.

The challenge of Associative and Commutative symbols in Tamarin Part II focuses on the handling of Associative and Commutative (AC) symbols in the Tamarin Prover.
Chapter 4 builds on the concepts introduced in Chapter 2, refining them to reflect the specificities of Tamarin. It delves into the internal representation of protocols and the mechanisms for adversarial message deduction.
Chapter 5 introduces support for user-defined AC symbols in Tamarin, along with novel mechanisms to address the termination issues these operators typically pose. In particular, it presents the *No Deconstruction Chain* property, along with corresponding normal form condition, and a new constraint-reduction rules $\mathbf{DG}_{+,\perp}$. This chapter also compares

built-in and user-defined AC symbols, demonstrating improvements to existing case studies and enabling new scenarios that were previously impossible to model.

Part I.

Formal analysis using verification tools

Symbolic Analysis using Sapic$^+$

2.

Summary

This chapter introduces the fundamental concepts for formally modeling and analyzing cryptographic protocols using Sapic$^+$ [25]. These concepts are later applied in Chapter 3, where we analyze a real-world protocol called LAKE-EDHOC. We provide an overview of Sapic$^+$, including its syntax and semantics for modeling protocols and specifying the security properties to be verified. Finally, we shortly present the three tools to which Sapic$^+$ can translate: Tamarin, ProVerif, and DeepSec. For each tool, we briefly outline its specific features, advantages and limitations, and explain how they can be combined in a comprehensive analysis.

2.1 Syntax & Semantics . . . 12
2.1.1 Messages as terms 12
2.1.2 Protocols as processes . . 16
2.1.3 Properties as first-order formulas 20
2.2 Sapic$^+$ translation to other tools 23
2.2.1 Tamarin Prover 24
2.2.2 ProVerif 25
2.2.3 DeepSec 26

Sapic$^+$ is a protocol verification platform integrated with Tamarin that uses a dialect of the applied pi-calculus [13] to specify cryptographic protocols. A single Sapic$^+$ specification can be compiled to three state-of-the-art verification tools used as backends: Tamarin Prover, ProVerif, and DeepSec. In [25], Cheval *et al.* provide a correctness proof showing that results obtained using one backend can be soundly reused in the other ones, ensuring consistency and avoiding redundant work. This means that one tool may prove a security property (or discover an attack) that another tool cannot. This feature enables users to leverage the strengths of all three tools while maintaining a single, tool-agnostic model, thereby reducing complexity and potential errors associated with multiple specifications.

[13]: Abadi et al. (2018), *The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication*

[25]: Cheval et al. (2022), *Sapic$^+$: protocol verifiers of the world, unite!*

Sapic$^+$ is an extension of the original Sapic front-end [26], which only supported translation to Tamarin and offered fewer specification features.

Outline We describe and formalize in Section 2.1 the syntax and semantics of messages (Section 2.1.1), protocols (Section 2.1.2) and security properties (Section 2.1.3) used in formal verification, with specificities of Sapic$^+$. We provide in Section 2.2 a quick overview of the back-end tools to which Sapic$^+$ can translate.

2.1. Syntax & Semantics

2.1.1. Messages as terms

In formal verification, it is common to represent messages and cryptographic primitives using a *term algebra*, which emphasizes the structural composition of messages and abstracts from implementation details.

Terms and substitutions

To represent a message, we recursively build *terms* by applying **function symbols** to **atomic terms**.

Atomic terms represent either fresh values or variables. A fresh value n, denoted $\sim n$, models a nonce, a key or a random value in general and is represented by constants drawn from an infinite set of names, $\mathcal{N}$. We divide this set $\mathcal{N}$ into two subsets: $\mathcal{N}_{pub}$ for public names (which are thus known to the attacker) and $\mathcal{N}_{priv}$ for private names (typically used to represent secrets within the protocol). Variables represent unknown values, used as inputs to the protocol and instantiated during its execution. They are drawn from an infinite set of variables, $\mathcal{V}$. We denote by $vars(t)$ the set of variables occurring in the term t.

Function symbols represent cryptographic operations. We assume a signature Σ_{Op} that defines the set of operators and their arities, denoted, for example, as $\mathtt{f}/n$. A function symbol can be a constructor operator or a destructor operator. Accordingly, we assume that $\Sigma_{Op} = \Sigma^c_{Op} \cup \Sigma^d_{Op}$. A *destructor* is a function symbol whose application may fail and that can only be applied to terms built using constructor operators.

Definition 2.1.1 *Using variables from $\mathcal{V}$ and names from $\mathcal{N}$, we construct the set of terms $T_{\Sigma_{Op}}(\mathcal{V}, \mathcal{N})$, which is closed under application of operators in Σ_{Op}. We construct a term t recursively:*

$$t = \begin{cases} n & t \in \mathcal{N} \\ v & t \in \mathcal{V} \\ \mathtt{f}(t_1, \ldots, t_k) & \mathtt{f}/k \in \Sigma_{Op} \wedge t_1, \ldots, t_k \in T_{\Sigma_{Op}}(\mathcal{V}, \mathcal{N}) \end{cases}$$

A term t is ground when it contains no variables and we denote the set of ground terms by $T_{\Sigma_{Op}}(\mathcal{N})$, or simply $T_{\Sigma_{Op}}$.

Example 2.1.1 Consider the protocol presented in Figure 1.3, the corresponding signature is:

$$\Sigma_{Op} = \{\mathtt{h}/1, \oplus/2, \mathtt{zero}/0\}$$

The function symbol $\mathtt{h}/1$ models a hash function and is applied to a single message, hence its arity of 1. The symbol $\oplus/2$ denotes the XOR operator with arity 2, and $\mathtt{zero}/0$ represents the constant value zero, making it a function symbol of arity 0.
Let $N = \{n_a, n_b, k\} \subseteq \mathcal{N}$ and $V = \{m, n\} \subseteq \mathcal{V}$. Following the definition above, n_a and k are atomic terms, so $n_a \oplus n$ and $\mathtt{h}(n_a \oplus n) \in T_{\Sigma_{Op}}(V, N)$ and $\mathtt{h}(\mathtt{zero})$ is a ground term, as it contains no variables.

Now consider the symbolic model of symmetric encryption with the signature:

$$\Sigma_{Op} = \{\mathtt{enc}/2, \mathtt{dec}/2\}$$

Here, the function symbols represent encryption and decryption, respectively: $\mathtt{enc}/2$ models the encryption of a message with a key and is a constructor symbol, while $\mathtt{dec}/2$ models decryption with the corresponding key and is a destructor symbol, as decryption may fail if the key is incorrect.

Here, we focus only on the function symbols. The following sections explain how their properties are defined.

A **substitution** $\sigma : \mathcal{V} \to T_{\Sigma_{Op}}(\mathcal{V}, \mathcal{N})$ is a function from variables to terms. We homomorphically extend σ to terms and use postfix notations: $t\sigma$ is the term in which we replace each variable $x \in vars(t) \cap dom(\sigma)$[1] by $\sigma(x)$

1: $dom(\sigma)$ is the domain of the substitution, i.e., the set of variables such that $\sigma(x) \neq x$

and for $\mathtt{f}/k \in T_{\Sigma_{Op}}(\mathcal{V}, \mathcal{N})$, $\mathtt{f}(t_1, \ldots, t_k)\sigma = \mathtt{f}(t_1\sigma, \ldots, t_k\sigma)$. $t\sigma$ is named an *instance of* t. We define a substitution with the notation $\sigma = \{x \mapsto a\}$, meaning that $\sigma(x) = a$.

Example 2.1.2 Consider the same Σ_{Op}, N, V as in Example 2.1.1. We define two substitutions:

$$\sigma_1 = \{m \mapsto n_b, n \mapsto \mathtt{h}(k)\} \quad \sigma_2 = \{n \mapsto n_a\}$$

Hence $(m \oplus n)\sigma_1 = n_b \oplus \mathtt{h}(k)$ and $\mathtt{h}(n_a \oplus n)\sigma_2 = \mathtt{h}(n_a \oplus n_a)$

Equational theories and rewrite systems

To actually express that function symbols represent a given cryptographic primitive, we define their properties using an *equational theory* defined by a set of equations.

An *equation* is an unordered pair of terms $s, t \in T_{\Sigma_{Op}}(\mathcal{V})$ written $s = t$. For a set of equations E over Σ_{Op} the resulting *equational presentation* is $\mathscr{E} = (\Sigma_{Op}, E)$.

Definition 2.1.2 *An **equational theory**, written $=_{\mathscr{E}}$, is the smallest equivalence relation that contains all $t_1 = t_2 \in E$, is closed under substitution of variables and application of function symbols from Σ_{Op} and containing all instances of E. This relation is induced on constructor terms*[2].
We often drop the Σ_{Op} and write $=_E$ for the equational theory $=_{\mathscr{E}}$ when it is clear from the context.

2: A constructor term is a term containing no destructor symbol from Σ^d_{Op}.

Two terms s and t are equal modulo E iff $s =_E t$. For all operations on sets, sequences and multisets we use the subscript E to denote that this is to be considered modulo E [note 1].

[note 1]: For example, we write $\in_E$ for set membership modulo E.

The semantics of destructor operators is defined through a set of **rewrite rules** $\mathscr{E}^d$ [note 2] of the form $\mathtt{d}(t_1, \ldots, t_k) \to r$ where $\mathtt{d}/k \in \Sigma^d_{Op}$ and $t_1, \ldots, t_k \in T_{\Sigma^c_{Op}}(\mathcal{V})$. We say that the term $\mathtt{d}(t_1, \ldots, t_k)$ rewrites to the term r. The rewrite system defines how to apply the destructor operators of Σ_{Op}.

[note 2]: We call it a rewrite system.

We write $t_1 \to_E t_2$ when t_1 is built using only one destructor operator $\mathtt{d}/k$, and $t_1 = \mathtt{d}(t_1, \ldots, t_k)$ rewrites to t_2 modulo E: $t_1 \to_E t_2$ iff $\mathtt{d}(v_1, \ldots, v_k) \to v$, $v_i =_E t_i$ and $v =_E t_2$. Terms with several destructors are rewritten bottom-up, preserving rewriting destructor symbols applied to another destructor operator.

We only consider rewrite systems that are *convergent*. We denote $\to^*$ the transitive-reflexive closure of $\to$.

Definition 2.1.3 *A rewrite system $\mathscr{E}^d$ is **terminating** if there is no infinite sequence of terms t_i such that $t_i \to_E t_{i+1}$.*
*$\mathscr{E}^d$ is **confluent** if $\forall t_1, u_1, u_2 \in T_{\Sigma_{Op}}(\mathcal{V}, \mathcal{N})$ such that $t_1 \to^*_E u_1$ and $t_1 \to^*_E u_2$, there exists $t_2 \in T_{\Sigma_{Op}}(\mathcal{V}, \mathcal{N})$ such that $u_1 \to^*_E t_2$ and $u_2 \to^*_E t_2$.*
*$\mathscr{E}^d$ is **convergent** if it is terminating and confluent.*
If, for all u such that $v \to^*_E u$, $v =_E u$ and $t \to^*_E \cdots \to^*_E v$, then v is called the normal form of t and is denoted $t\downarrow_E$.

When a term contains a destructor symbol that does not rewrite[3], we say that the destructor fails. As we expect a protocol to only send valid messages, we define a predicate $Msg(t)$ evaluated to true if the term t is valid: for all subterms t_i of t, $t_i \in T_{\Sigma^c_{Op}}(\mathcal{V}, \mathcal{N})$, i.e., they are constructor terms.

3: Formally, $t\downarrow_E = \mathtt{d}(t_1, \ldots, t_k)$ and $\mathtt{d}/k \in \Sigma^d_{Op}$

Definition 2.1.4 *We lift* $=_E$ *to arbritrary terms:*

$$\forall t_1, t_2 \in T_{\Sigma_{Op}}(\mathcal{V}, \mathcal{N}).\ Msg(t_1) \wedge Msg(t_2) \wedge t_1\downarrow_E =_E t_2\downarrow_E \Rightarrow t_1 =_E t_2$$

Example 2.1.3 The equational theory for XOR operator $\mathscr{E}_{XOR}$ used in the protocol described in Section 1.1.2 includes:

$$a \oplus (b \oplus c) = (a \oplus b) \oplus c \qquad a \oplus b = b \oplus a$$
$$a \oplus a = \mathtt{zero} \qquad a \oplus \mathtt{zero} = a$$

The equations in the second line allow the operator to cancel a term. The equations in the first line express the associativity and the commutativity of the operator.

The equational theory for symmetric encryption $\mathscr{E}_{senc}$ consists of a single equation:

$$\mathtt{dec}(\mathtt{enc}(m, k), k) = m$$

This means that a message m encrypted with a key k can only be recovered by decrypting it with the same key. Using a different key k_1 will not yield the original message: $\mathtt{dec}(\mathtt{enc}(m, k), k_1) \neq_{\mathscr{E}_{senc}} m$

The equational theories supported by SAPIC$^+$ correspond to the union of those handled by TAMARIN and PROVERIF. In cases such as Diffie-Hellman (more fully supported in TAMARIN than in PROVERIF) an abstracted version of the theory is exported to the latter tool.

Additional term construction

In SAPIC$^+$, we define additional particular construction of terms.

Patterns We define the set of patterns $Pat \subseteq T_{\Sigma^c_{Op}}(\mathcal{V}, \mathcal{N})$ to be a subset of terms against which we can perform pattern matching in `let`[4] construction.

4: Defined in Section 2.1.2

Conditional formulas Conditional branchings (`if c then P else Q`[5] construction) in protocols are described using first-order formulas[6] for the condition `c`. It is of the form $t_1 = t_2$ and can contain quantifiers on variables. We interpret = in those formulas as $=_E$.

5: Defined in Section 2.1.2

6: See Section 2.1.3

Facts We define a set of facts Σ_{Fact}, that are terms built by applying fact symbols[7] on constructor terms. This set contains in particular the facts `K`, `In`, `Out`, used to log attacker's knowledge, inputs and outputs. We use Σ_{Fact} to annotate processes and log events[8].

7: Fact symbols are defined in a similar way to function symbols.

8: See Sections 2.1.2 and 2.1.3

$\langle P, Q \rangle$ ::=

(elementary processes)		*(extended processes)*	
0	*null process*	| event F; P	*event*
| new n; P	*binding of a fresh name*	| $!P$	*replication*
| $P \mid Q$	*parallel composition*		
| out(t_1, t_2); P	*output of t_2 on channel t_1*		
| in(t_1, x); P	*input on channel t of a term matching x*		
| if ϕ then P else Q	*conditional*		
| let $p = t$ in P else Q	*let binding*		

Figure 2.1.: Process syntax [25], with $n \in \mathcal{N}_{priv}$, $x \in \mathcal{V}$, $t, t_1, t_2 \in T_{\Sigma_{Op}}(\mathcal{V}, \mathcal{N})$, $p \in Pat$ and $\mathsf{F} \in \Sigma_{Fact}$.

2.1.2. Protocols as processes

In SAPIC$^+$, protocols are modeled as processes in a dialect of the applied pi-calculus [13]. The applied pi-calculus is a specification language for modeling cryptographic protocols as communicating, parallel programs. Its semantics takes into account that communications happen over an adversary-controlled network, as the Dolev-Yao adversary models we introduced in Section 1.2.2. The adversary can intercept any message and insert forged messages. To forge a message an adversary may build terms from $\mathcal{N}_{pub}$ and previously intercepted messages, possibly applying function symbols from Σ_{Op} on these.

[13]: Abadi et al. (2018), *The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication*

Syntax

Figure 2.1 illustrates the syntax of SAPIC$^+$. We distinguish between **elementary processes** and **extended processes**.

Elementary processes model finite protocols, i.e., those that involve a finite number of processes. The basic process is called the null process: it is denoted 0 and it figures the end of a process. For this reason, it is often implicit in practice. In finite protocols, fresh values are generated using the `new` instruction. Terms can be output on a channel with `out` and received with `in`. Channels are by default represented as public terms and are thus assumed to be known to the adversary, in accordance with the Dolev-Yao attacker model. However, private channels can be modeled using private terms. Elementary processes also support conditional branching using the `if ... then ... else ...` construct, as well as pattern matching with the `let ... in ...` construct. Finally, parallel execution is supported via the `|` symbol.

Extended processes model unbounded protocol sessions, in addition to all the capabilities of elementary processes. The `event` instruction is used to log information about the execution of a process. This information is modeled as a fact and serves as an annotation for specifying reachability properties. It has no effect on the actual execution of the process. Extended processes also support replication, denoted by `!`, which allows modeling an unbounded number of concurrent instances of a process running in parallel.

Example 2.1.4 Consider the protocol described in Section 1.1.2. Listings 2.1 and 2.2 present the SAPIC$^+$ models of the `Initiator` and `Responder` processes. In this example, the terms indexed with $\mathcal{V}$ represent variables, not names.

The `Initiator` first generates a nonce `~na`, which is annotated with the fact event `Challenge(~na)`. It then sends this nonce over a public channel (denoted with the public name `c`), which is accessible to both the responder and potential attackers. Note that the channel name can be omitted in the `out` instruction; if so, the channel is considered public by default. Next, the initiator is expected to receive a message $m_{\mathcal{V}}$ and a value $nb_{\mathcal{V}}$ from the same channel. (These two terms are variables to allow the adversary to send the messages it wants.) It computes the term `n` by applying the XOR operator to its nonce, the received value and its symmetric key `k`. Finally, it computes the hash of `n`, and if it matches the received message $m_{\mathcal{V}}$, it logs the fact `Alive(~na,~k)`, indicating that the initiator has successfully completed its session using nonce `~na` and key `k`.

The `Responder` begins by receiving a value $na_{\mathcal{V}}$ from the channel. It then generates its own nonce `~nb` and computes $na_{\mathcal{V}} \oplus$ `~k` $\oplus$ `~nb` which it binds to the term `n`. If the received value $na_{\mathcal{V}}$ is equal to zero, the responder terminates its session. Otherwise, it sends over the channel the hash of `n` along with its nonce `~nb`. It then logs the facts `NeqForExe(`$na_{\mathcal{V}}$`,zero)` (ensuring that the received nonce is not null) and `Response(~k)` (indicating that the responder has achieved its session by sending a message involving its symmetric key `~k`). The `out` instruction corresponds to the `in` instruction in the initiator's process.

Listing 2.3 shows the model of the main process of the protocol, describing how the subprocesses (i.e., the participants) interact. First, a symmetric key `~k` is generated and assumed to be shared among the two participants (represented by the `~k` parameter passed to both processes). Then, sessions of the `Initiator` and sessions of the `Responder` are run in parallel.

Note that this example considers only two agents. To model multiple pairs running in parallel, we could replicate the long-term key generation.

The complete model is available in Appendix A.1.

Listing 2.1: Initiator process of the running example

```
let Initiator(~k) =
  new ~na;
  event Challenge(~na);
  out(c,~na);
  in(c,<mV,nbV>);
  let n = ~na ⊕ ~k ⊕ nbV in
  if h(n) = mV then
    event Alive(~na,~k)
```

Listing 2.2: Responder process of the running example

```
let Responder(~k) =
 in(c,naV);
 new ~nb;
 let n = naV ⊕ ~k ⊕ ~nb in
 if naV = zero then 0
 else ( event Response(~k);
  event NeqForExe(naV,zero);
  out(c,<h(n),~nb>) )
```

```
process:
  new ~k;
  ! ( Initiator(~k) ||
      Responder(~k) )
```

Listing 2.3: General process of the running example

Remark 2.1.1 SAPIC$^+$ provides also a syntax for *stateful processes* that model protocols with global shared states:

$\langle P, Q \rangle$ ::=	*(stateful processes)*
| insert st, t; P	*set value of state st to t*
| delete st; P	*delete state st*
| lookup st as x in P else Q	*read the state*
| lock st; P	*lock a state*
| unlock st; P	*unlock a state*

The analysis presented in Chapter 3 does not involve stateful processes; therefore, we do not describe this type of process in detail. Interested readers may refer to [25] for further information.

Semantics

We define an operational semantics for SAPIC$^+$ processes to formally model the execution of processes in the presence of a Dolev-Yao attacker controlling the network. This semantics is also essential for expressing the security properties described in Section 2.1.3.

Definition 2.1.5 *A configuration $\mathcal{C} = (\mathcal{B}, \mathcal{P}, \sigma)$ is defined by:*

- *$\mathcal{B}$: the set of currently bound names.*
- *$\mathcal{P}$: the set of executable processes.*
- *σ: the substitution representing the current knowledge of the attacker. It is defined over an infinite set of variables $\mathcal{AX} = \{att_i\}_{i\in\mathbb{N}}$, each mapped to a term in the attacker's knowledge: $\sigma = \{att_1 \mapsto t_1, \ldots, att_n \mapsto t_n\}$, where all terms t_i are in normal form and originate from protocol outputs.*

Remark 2.1.2 As mentioned earlier, we exclude stateful processes in this work. With this notion, a configuration is defined as a tuple $(\mathcal{B}, \mathcal{S}, \mathcal{P}, \sigma, \mathcal{L})$; where $\mathcal{S}$ represents the current values of states and $\mathcal{L}$ the set of active locks. For more details, see [25].

A *recipe* is a term constructed by the adversary by applying function symbols on the terms in its knowledge or public names. We denote this as $r \in T_{\Sigma_{Op}}(\mathcal{AX}, \mathcal{N}_{pub})$.

The **semantics** is a labeled transition system between configurations, defined through a transition relation equipped with a label:

- $\longrightarrow$ is the transition with no label.
- $\xrightarrow{\mathsf{K}(t)}$ indicates that t is sent over the network and becomes known to the attacker.
- $\xrightarrow{\mathsf{In}(R,R')}$ represents a protocol input, where R' is a recipe.
- $\xrightarrow{\mathsf{Out}(R)}$ represents the sending of a term on channel R. If the adversary can compute R, it is allowed to add the sent term to its knowledge.
- $\xrightarrow{\mathsf{F}}$ is the transition labeled with $\mathsf{F} \in \Sigma_{Fact}$

We use the following notations:

- $\longrightarrow^*$ is the reflexive transitive closure of $\longrightarrow$
- $\overset{\mathsf{l}}{\Rightarrow}$ is shorthand for $\longrightarrow^* \xrightarrow{\mathsf{l}} \longrightarrow^*$, with $\mathsf{l} \in \Sigma_{Fact}$
- $\mathcal{C}_i \overset{\mathsf{l}_1 \ldots \mathsf{l}_n}{\Longrightarrow} \mathcal{C}_f$ holds if $\mathcal{C}_i \overset{\mathsf{l}_1}{\Rightarrow} \ldots \overset{\mathsf{l}_n}{\Rightarrow} \mathcal{C}_f$, where $\mathcal{C}_i, \mathcal{C}_f$ are two configurations and $\mathsf{l}_1 \ldots \mathsf{l}_n \in \Sigma_{Fact}$
- $P \overset{\mathsf{l}_1 \ldots \mathsf{l}_n}{\Longrightarrow} \mathcal{C}$ is shorthand for $(\emptyset, \{P\}, \emptyset) \overset{\mathsf{l}_1 \ldots \mathsf{l}_n}{\Longrightarrow} \mathcal{C}$

The transition rules for elementary and extended processes are presented in Figure 2.2. We highlight a few of them below:

- The 2nd transition rule states that when running P and Q in parallel, both processes are added to the multiset $\mathcal{P}$.
- The 6th transition rule specifies that when executing an output instruction of the form `out(`t_1`,`t_2`); P`, the adversary first attempts to compute t_1 based on its current knowledge to determine whether it can access the channel. If it can, and if t_2 is a valid message, t_2 is added to the adversary's knowledge. The `out` instruction is then replaced by P in the multiset of processes. The facts $\mathsf{Out}(R)$ and $\mathsf{K}(t_1)$ are logged.

$$
\begin{array}{rcl}
(\mathcal{B}, \mathcal{P} \cup^\sharp \{0\}, \sigma) & \longrightarrow & (\mathcal{B}, \mathcal{P}, \sigma) \\
(\mathcal{B}, \mathcal{P} \cup^\sharp \{P|Q\}, \sigma) & \longrightarrow & (\mathcal{B}, \mathcal{P} \cup^\sharp \{P, Q\}, \sigma) \\
(\mathcal{B}, \mathcal{P} \cup^\sharp \{!P\}, \sigma) & \longrightarrow & (\mathcal{B}, \mathcal{P} \cup^\sharp \{!P, P\}, \sigma) \\
(\mathcal{B}, \mathcal{P} \cup^\sharp \{\mathsf{new}\ n; P\}, \sigma) & \longrightarrow & (\mathcal{B} \cup \{n'\}, \mathcal{P} \cup^\sharp \{P\{n \mapsto n'\}\}, \sigma) \\
 & & \quad \text{if } n' \in \mathcal{N}_{priv} \text{ is fresh} \\
(\mathcal{B}, \mathcal{P}, \sigma) & \xrightarrow{\mathsf{K}(t)} & (\mathcal{B}, \mathcal{P}, \sigma) \\
 & & \quad \text{if } t =_E R\sigma \text{ for some } R \in T_{\Sigma_{Op}}(\mathcal{AX}, \mathcal{N}_{pub}) \\
(\mathcal{B}, \mathcal{P} \cup^\sharp \{\mathsf{out}(t_1, t_2); P\}, \sigma) & \xrightarrow{\mathtt{Out}(R),\mathsf{K}(t_1)} & (\mathcal{B}, \mathcal{P} \cup^\sharp \{P\}, \sigma \cup \{att_n \mapsto t_2\}) \\
 & & \quad \text{if } t_1 =_E R\sigma \text{ for some } R \in T_{\Sigma_{Op}}(\mathcal{AX}, \mathcal{N}_{pub}), \\
 & & \quad Msg(t_2) \text{ and } n = |\sigma| + 1 \\
(\mathcal{B}, \mathcal{P} \cup^\sharp \{\mathsf{in}(t, x); P\}, \sigma) & \xrightarrow{\mathtt{In}(R,R'),\mathsf{K}(\langle t, R'\sigma\rangle)} & (\mathcal{B}, \mathcal{P} \cup^\sharp \{P\{x \mapsto R'\sigma\}\}, \sigma) \\
 & & \quad \text{if } t =_E R\sigma, Msg(R'\sigma) \\
 & & \quad \text{for some } R, R' \in T_{\Sigma_{Op}}(\mathcal{AX}, \mathcal{N}_{pub}) \\
(\mathcal{B}, \mathcal{P} \cup^\sharp \{\mathsf{out}(t_1, t_2); P, \mathsf{in}(t, x); Q\}, \sigma) & \longrightarrow & (\mathcal{B}, \mathcal{P} \cup \{P, Q\{x \mapsto t_2\}\}, \sigma) \\
 & & \quad \text{if } t_1 =_E t \text{ and } Msg(t_2) \\
(\mathcal{B}, \mathcal{P} \cup \{\mathsf{if}\ \phi\ \mathsf{then}\ P\ \mathsf{else}\ Q\}, \sigma) & \longrightarrow & (\mathcal{B}, \mathcal{P} \cup \{P\}, \sigma) \\
 & & \quad \text{if } \phi \text{ holds} \\
(\mathcal{B}, \mathcal{P} \cup \{\mathsf{if}\ \phi\ \mathsf{then}\ P\ \mathsf{else}\ Q\}, \sigma) & \longrightarrow & (\mathcal{B}, \mathcal{P} \cup \{Q\}, \sigma) \\
 & & \quad \text{if } \phi \text{ does not hold} \\
(\mathcal{B}, \mathcal{P} \cup \{\mathsf{event}(\mathsf{F}); P\}, \sigma) & \xrightarrow{\mathsf{F}} & (\mathcal{B}, \mathcal{P} \cup \{P\}, \sigma) \\
(\mathcal{B}, \mathcal{P} \cup^\sharp \{\mathsf{let}\ p = t\ \mathsf{in}\ P\ \mathsf{else}\ Q\}, \sigma) & \longrightarrow & (\mathcal{B}, \mathcal{P} \cup^\sharp \{P\tau\}, \sigma) \\
 & & \quad \text{if } p\tau =_E t \text{ and } \tau \text{ is grounding for } p \\
(\mathcal{B}, \mathcal{P} \cup^\sharp \{\mathsf{let}\ p = t\ \mathsf{in}\ P\ \mathsf{else}\ Q\}, \sigma) & \longrightarrow & (\mathcal{B}, \mathcal{P} \cup^\sharp \{Q\}, \sigma) \\
 & & \quad \text{if for all } \tau, p\tau \neq_E t
\end{array}
$$

Figure 2.2.: SAPIC$^+$ operational semantics, standard operations [25]. $\cup^\sharp$: multiset union

- The 9th and 10th transition rules describe conditional branching. When the process is of the form `if ... then ... else`, it is removed from $\mathcal{P}$ and replaced by P if the condition evaluates to true, or by Q otherwise.
- The 11th transition rule corresponds to the `event(F)` instruction. This rule does not affect execution (no sets are modified) and the process continues as normal. The fact F is simply logged.

Definition 2.1.6 *A **trace** is a sequence of labels $[l_1 \dots l_n]$[9], along with the attacker knowledge σ of the resulting configuration.*
We define the set of traces of a process P:

$$traces^{pi}(P) = \{(tr, \sigma) \mid \exists\, \mathcal{B}, \mathcal{P}.\ P \overset{tr}{\Rightarrow} (\mathcal{B}, \mathcal{P}, \sigma)\}$$

9: We use afterwards a more compact notation, tr

We denote $\overset{\text{tr}}{\Rightarrow}_{\vdash}$ the relation where In and Out facts are removed from the labels and $\overset{\text{tr}}{\Rightarrow}_{\approx}$ the relation where we only keep those two labels. Hence, we denote the corresponding traces set by $traces^{pi}_{\vdash}(P)$ and $traces^{pi}_{\approx}(P)$.

Example 2.1.5 Let us go back to the running example. We illustrate the different transition rules and the notion of traces by showing an execution of the protocol. The initial configuration is $\mathcal{C}_0 = (\emptyset, \{\text{new } k; Init | Resp\}, \emptyset)$
(For the example, we remove the replication for simplicity.)

$$
\begin{aligned}
\mathcal{C}_0 &\longrightarrow (\{k\}, \{Init|Resp\}, \emptyset)\\
&\longrightarrow (\{k\}, \{Init, Resp\}, \emptyset)\\
&\longrightarrow (\{k, n_a\}, \{Init_1, Resp\}, \emptyset)\\
&\xrightarrow{\text{Challenge}(n_a)} (\{k, n_a\}, \{Init_2, Resp\}, \emptyset)\\
&\xrightarrow{\text{Out}(c),\text{K}(c)} (\{k, n_a\}, \{Init_3, Resp\}, \{att_0 \mapsto n_a\})\\
&\xrightarrow{\text{In}(c,n_a),\text{K}(\langle c,n_a\rangle)} (\{k, n_a\}, \{Init_3, Resp_1\}, \{att_0 \mapsto n_a\})\\
&\longrightarrow (\{k, n_a, n_b\}, \{Init_3, Resp_2\}, \{att_0 \mapsto n_a\})\\
&\longrightarrow (\{k, n_a, n_b\}, \{Init_3, Resp_3\}, \{att_0 \mapsto n_a\})\\
&\xrightarrow{\text{Response}(k),\text{NeqForExe}(n_a,\texttt{zero})} (\{k, n_a, n_b\}, \{Init_3, \text{out}(c, \langle \text{h}(n_a \oplus k \oplus n_b), n_b\rangle)\}, \{att_0 \mapsto n_a\})\\
&\xrightarrow{\text{Out}(c),\text{K}(c)} (\{k, n_a, n_b\}, \{Init_3\}, \{att_0 \mapsto n_a, att_1 \mapsto \langle h_n, n_b\rangle\})^{10}\\
&\xrightarrow{\text{In}(c,\langle h_n,n_b\rangle),\text{K}(\langle c,\langle h_n,n_b\rangle\rangle)} (\{k, n_a, n_b\}, \{Init_4\}, \{att_0 \mapsto n_a, att_1 \mapsto \langle h_n, n_b\rangle\})\\
&\longrightarrow (\{k, n_a, n_b\}, \{\text{if h}(n) = h_n \text{ then event Alive}(n_a, k)\}, \{att_0 \mapsto n_a, att_1 \mapsto \langle h_n, n_b\rangle\})\\
&\longrightarrow (\{k, n_a, n_b\}, \{\text{event Alive}(n_a, k)\}, \{att_0 \mapsto n_a, att_1 \mapsto \langle h_n, n_b\rangle\})\\
&\xrightarrow{\text{Alive}(n_a,k)} (\{k, n_a, n_b\}, \emptyset, \{att_0 \mapsto n_a, att_1 \mapsto \langle h_n, n_b\rangle\})
\end{aligned}
$$

where:
$Init = \{\text{new } n_a; Init_1\}$
$Init_1 = \{\text{event Challenge}(n_a); Init_2\}$
$Init_2 = \{\text{out}(c, n_a); Init_3\}$
$Init_3 = \{\text{in}(c, \langle h_n, n_b\rangle); Init_4\}$
$Init_4 = \{\text{let } n = n_a \oplus k \oplus n_b \text{ in}$
$(\text{if h}(n) = h_n \text{ then event Alive}(n_a, k))\}$
$Resp = \{\text{in}(c, n_a); Resp_1\}$
$Resp_1 = \{\text{new } n_b; Resp_2\}$
$Resp_2 = \{\text{if } n_a = \texttt{zero} \text{ then } 0 \text{ else}$
$(\text{event Response}(k); \text{event NeqForExe}(n_a, \texttt{zero});$
$\text{out}(c, \langle h_n, n_b\rangle))\}$
$Resp_3 = \{\text{event Response}(k);$
$\text{event NeqForExe}(n_a, \texttt{zero}); \text{out}(c, \langle h_n, n_b\rangle)\}$

10: $h_n = \text{h}(n_a \oplus k \oplus n_b)$

The trace of this execution is:

$$
\left(\text{tr} = \left\{
\begin{array}{l}
\text{Challenge}(n_a),\\
\text{Out}(c),\\
\text{K}(c),\\
\text{In}(c, n_a),\\
\text{K}(\langle c, n_a\rangle),\\
\text{Response}(k),\\
\text{NeqForExe}(n_a, \texttt{zero}),\\
\text{Out}(c),\\
\text{K}(c),\\
\text{In}(c, \langle h_n, n_b\rangle),\\
\text{K}(\langle c, \langle h_n, n_b\rangle\rangle), \text{Alive}(n_a, k)
\end{array}
\right\}, \sigma = \left\{
\begin{array}{l}
att_0 \mapsto n_a,\\
att_1 \mapsto \langle h_n, n_b\rangle
\end{array}
\right\}\right)
$$

2.1.3. Properties as first-order formulas

Security properties are expressed as properties over the traces of a protocol. SAPIC$^+$ provides a first-order logic to specify such properties.

Reachability property

A reachability property is a formula defined over all traces of a protocol $traces^{pi}(P)$. It is typically used to express security goals such as confidentiality or authentication. These properties are modeled using the facts associated with the `event` instructions within the protocol, along with a temporal logic.

The temporal logic used in SAPIC$^+$ to express reachability properties is the same as in the TAMARIN Prover [18]: an expressive two-sorted first-order logic that distinguishes between terms, of sort *msg*, and time points, of sort *temp*. The purpose of the *temp* sort is to enable reasoning about the ordering of events, for example to express that one event occurs before another one.

[18]: Schmidt et al. (2012), *Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties (extended version)*

Definition 2.1.7 ([25], Definition 1) *An atomic trace formula is either:*

- $\bot$ *false*
- $t_1 \approx t_2$ *a term equality (modulo the equational theory)*
- $i \lessdot j$ *a time point ordering (i occurs before j)*
- $i \doteq j$ *a time point equality*
- $F@i$ *an action for a fact* $F \in \Sigma_{Fact}$ *and a time point* i (F *occurs in the* i^{th} *element of the trace)*

A ***trace formula*** *is a first-order formula over atomic trace formulas.*

We equip the two sorts s with a domain $dom(s)$ and we define two sets of variables for each sort, $\mathcal{V}_{msg}$ and $\mathcal{V}_{temp}$.

$$dom(s) = \begin{cases} T_{\Sigma_{Op}}(\mathcal{N}) & s = msg \\ \mathbb{Q} & s = temp \end{cases}$$

We define a *valuation* as a function $\theta : \mathcal{V} \to T_{\Sigma_{Op}}(\mathcal{N}) \cup \mathbb{Q}$[11] respecting sorts, i.e., $\theta(\mathcal{V}_s) \subset dom(s)$. We homomorphically extend θ to terms and use postfix notations, as we do for substitutions.

11: Where $\mathbb{Q}$ denotes the set of rational numbers.

We denote $idx(\mathsf{tr})$ the set of positions in a trace tr, and tr_i the multiset of facts of tr at position i. Those positions are useful to model a time ordering between the facts in a trace.

Definition 2.1.8 ([25], Definition 2) *The* ***satisfaction relation*** $(tr, \theta) \models \phi$ *between a trace* tr*, a valuation* θ *and a trace formula* ϕ *is defined as follows:*

$(tr, \theta) \models \bot$	*never*
$(tr, \theta) \models F@i$	*iff* $\theta(i) \in idx(tr)$ *and* $F\theta \in_E tr_{\theta(i)}$
$(tr, \theta) \models i \lessdot j$	*iff* $\theta(i) < \theta(j)$
$(tr, \theta) \models i \doteq j$	*iff* $\theta(i) = \theta(j)$
$(tr, \theta) \models t_1 \approx t_2$	*iff* $t_1\theta =_E t_2\theta$
$(tr, \theta) \models \neg\phi$	*iff not* $(tr, \theta) \models \phi$
$(tr, \theta) \models \phi_1 \wedge \phi_2$	*iff* $(tr, \theta) \models \phi_1$ *and* $(tr, \theta) \models \phi_2$
$(tr, \theta) \models \exists x : s.\phi$	*iff* $(tr, \theta[x \mapsto u]) \models \phi$ *for some* $u \in dom(s)$

The semantics of the remaining logical connectives and quantifiers is defined as usual. [18]

When ϕ is a closed formula, we can denote $\mathsf{tr} \models \phi$, the satisfaction of the formula being independent of the valuation. Given a set of traces TR, we denote $\mathsf{TR} \models \phi$ if $\forall \mathsf{tr} \in \mathsf{TR}.\ \mathsf{tr} \models \phi$ and by extension we denote $P \models \phi$ for $traces^{pi}_{\vdash}(P) \models \phi$.

Since SAPIC$^+$ uses the same logic as TAMARIN for reachability properties, it also supports the satisfiability relation; that is, checking whether a given property holds on a specific trace and valuation. This is denoted by $\models^{\exists}$, where $\mathsf{TR} \models \phi$ iff $\mathsf{TR} \not\models^{\exists} \neg\phi$. Such a relation is useful for sanity checking or verifying executability properties. However, SAPIC$^+$ can translate them only into TAMARIN.

Example 2.1.6 Consider the running example. We define the property `alive` stating that whenever an `Alive(x,y)` event is logged at some time point `i`, there must exist a corresponding `Response(y)` event somewhere in the trace. This property is modeled as follows:

```
lemma alive:
  All x y #i. Alive(x,y)@#i ==> Ex #j. Response(y)@#j
```

The `#` symbol denotes a variable of sort *temp* and the `@` symbol indicates the position of a fact within the trace.
Let tr_0 be the trace presented in Example 2.1.5, and let σ_0 be the corresponding attacker knowledge. We can say that tr_0 satisfies the `alive` property. Since no counterexample trace can be found, we conclude that $P_{CRXOR} \models$ `alive`.

We also define another property `recentalive_tag`, using a satisfiability relation: we seek to determine whether there exists a trace in which the events are not correctly ordered, i.e., where `Challenge` occurs first, `Response` second, and `Alive` last. This property is modeled using the `exists-trace` construct:

```
lemma recentalive_tag:
  exists-trace
    not ( All x y #i. Alive(x,y)@#i ==>
          Ex #j #k. Challenge(x)@#k & Response(y)@#j
          & #k < #j & #j < #i )
```

In tr_0, the three facts are ordered correctly. However, there exists another trace tr_a where this is not the case.
Since the process in Listing 2.3 defines both `Initiator` and `Responder` as replicated, we can replicate the `Initiator` process after tr_0. This results in a new trace and updated attacker knowledge:

$$\mathsf{tr}_1 = \mathsf{tr}_0 \cup \{\mathsf{Challenge}(n'_a), \mathsf{Out}(c), \mathsf{K}(c)\}$$

$$\sigma_1 = \sigma_0 \cup \{att_2 \mapsto n'_a\}$$

At this point, the attacker knows n_a, $\langle \mathsf{h}(n_a \oplus k \oplus n_b), n_b \rangle$ and n'_a. They can decompose the pair and compute the recipes $\mathsf{h}(n_a \oplus k \oplus n_b)$ and n_b. With this additional knowledge, the adversary produces $n'_b = n_a \oplus n_b \oplus n'_a$ and constructs the message $m_a = \langle \mathsf{h}(n_a \oplus k \oplus n_b), n'_b \rangle$ which is then sent to the `Initiator`. This results in the final trace:

$$\mathsf{tr}_a = \mathsf{tr}_1 \cup \{\mathsf{In}(c, m_a), \mathsf{K}(\langle c, m_a \rangle), \mathsf{Alive}(n'_a, k)\}$$

In tr_a, the order of the facts is `Response` first, then `Challenge` and finally `Alive`. Therefore, $\mathsf{tr}_a \models^{\exists}$ `recentalive_tag`.

The `Initiator` accepts the message because it computes $\mathsf{h}(n'_a \oplus k \oplus n'_b)$, which equals $\mathsf{h}(n_a \oplus k \oplus n_b)$ due to the equational theory of the XOR operator.

Equivalence property

An equivalence property is useful for proving anonymity or any property requiring that the adversary cannot distinguish between two processes. Since reachability properties alone are not sufficient for this purpose, we introduce the notion of *trace equivalence* to model indistinguishability. This relies on an intermediate concept known as *static equivalence*:

Definition 2.1.9 (Static equivalence ([25], Definition 3)) *Two substitutions* $\sigma_1, \sigma_2 : \mathcal{AX} \rightarrow T_{\Sigma^c_{Op}}(\mathcal{N})$ *are statically equivalent, written* $\sigma_1 \sim_E \sigma_2$, *iff*

$$\forall M, N \in T_{\Sigma_{Op}}(\mathcal{AX}, \mathcal{N}_{pub}).\ M\sigma_1 =_E N\sigma_1 \Leftrightarrow M\sigma_2 =_E N\sigma_2$$

This models the indistinguishability of two sequences of terms.

In other words, a passive adversary is not able to learn information to distinguish two sequences of terms if they are statically equivalent.

We apply this definition on the attacker's knowledge: two protocols are trace equivalent if the substitutions representing the adversarial knowdledge are statically equivalent for all possible traces.

Definition 2.1.10 (Trace equivalence ([25], Definition 4)) *Let* P *and* Q *be two processes.* P *is a trace included in* Q, *written* $P \sqsubseteq_E Q$, *iff*

$$\forall tr, \sigma_P.\ \exists \sigma_Q.\ (tr, \sigma_P) \in traces^{pi}_{\approx}(P) \Rightarrow (tr, \sigma_Q) \in traces^{pi}_{\approx}(Q) \wedge \sigma_P \sim_E \sigma_Q$$

P and Q are trace equivalent, written $P \approx_E Q$, *iff* $P \sqsubseteq_E Q$ *and* $Q \sqsubseteq_E P$

Informally, an arbitrary attacker running in parallel cannot distinguish whether it interacts with P or with Q.

In SAPIC+, such properties are translated into PROVERIF and DEEPSEC, but not into TAMARIN. The reason is that the notion of equivalence supported by TAMARIN differs from the ones used by PROVERIF and DEEPSEC: it relies on a form of *observational equivalence* between rewrite systems (see Section 4.1.3), which is distinct from trace equivalence. Moreover, there is no formal proof of correctness for translating trace equivalence for processes into observational equivalence for rewrite systems. As a result, equivalence properties cannot currently be exported from SAPIC+ to TAMARIN. PROVERIF verifies a strong process equivalence for indistinguishability properties. As this notion implies trace equivalence, the translation to PROVERIF is possible.

2.2. SAPIC+ translation to other tools

Once a protocol has been modeled in SAPIC+ using the concepts introduced in the previous section, it is exported to the three back-end tools used to carry out the proofs: TAMARIN Prover [18], PROVERIF [17], and DEEPSEC [24]. The translation process is detailed in [25] and will not be further discussed in this document.

The value of this platform lies in maintaining a single model while benefiting from the strengths of each back-end tool, including their specific features, such as oracles in TAMARIN. Moreover, the three tools

[18]: Schmidt et al. (2012), *Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties (extended version)*

[17]: Blanchet et al. (2022), *ProVerif with Lemmas, Induction, Fast Subsumption, and Much More*

[24]: Cheval et al. (2018), *DEEPSEC: Deciding Equivalence Properties in Security Protocols Theory and Practice*

[25]: Cheval et al. (2022), *SAPIC+: protocol verifiers of the world, unite!*

can be run in parallel on the same model: the proof is considered complete as soon as one tool succeeds. This allows for rapid initial feedback after modeling and helps determine which tool is most suitable for verifying a given property.

The following sections provide an overview of the back-end tools, along with preliminary guidance on how to combine them during analysis. A complete methodology is presented in Section 3.4.1.

2.2.1. TAMARIN Prover

We do not describe protocols as processes in TAMARIN, but rather using multiset rewriting rules of the form $id : l -[\ a\ ]\!\!\rightarrow r$, where id is a unique identifier, and l, a, and r are multisets of facts. A rule is applicable when all facts in l are available, resulting in the production of the facts in r. The multiset a is used solely for logging information in the trace, serving a similar purpose to the `event` instruction. These rules model both the behavior of the protocol participants (called protocol rules) and the capabilities of the attacker (known as message deduction rules). TAMARIN supports an unbounded number of protocol sessions.

TAMARIN models messages as terms, as described in Section 2.1.1, with some slight differences: the distinction between constructors and destructors is not made at the term level, but through message deduction rules. TAMARIN supports equational theories that are convergent and satisfy the finite variant property [27] (Definition 4.1.2). Users can define custom function symbols or rely on built-in theories modeling common cryptographic primitives, such as Diffie–Hellman exponentiation, multiset operations, or the XOR operator.

[27]: Comon-Lundh et al. (2005), *The finite variant property: How to get rid of some algebraic properties*

Security properties are defined as in Section 2.1.3 for reachability properties. The lemmas in Example 2.1.6 are also valid in TAMARIN. In addition, one can define restrictions using formulas, in order to consider only the traces that satisfy these constraints.

TAMARIN can operate in either automated or interactive proof modes. In interactive mode, the user selects the steps the tool should follow in its proof tree and can visualize the proof state using graphs. This mode is particularly useful when the automated mode fails to terminate. TAMARIN also enables the user to define complex heuristics using oracles, which guide the tool through the proof tree based on detailed patterns.

The strength of TAMARIN lies in its ability to precisely model cryptographic primitives such as Diffie–Hellman exponentiation or hash functions. The tool also guarantees full correctness of its proofs, albeit at the cost of more frequent termination issues, especially in the automated mode. In such cases, TAMARIN offers an interactive mode that allows the user to manually guide the proof process, supported by graphical tools to analyze the protocol and develop a proof strategy. While PROVERIF and DEEPSEC are often sufficient for simpler analyses, a good strategy is to use them to obtain initial results, then switch to TAMARIN for analyzing more complex lemmas that require richer equational theories, not supported by the other tools.

This tool is presented in more detail in Chapter 4.

2.2.2. PROVERIF

Similarly to TAMARIN, PROVERIF models messages as terms and cryptographic primitives using arbitrary equational theories, provided they are convergent and satisfy the finite variant property. While PROVERIF does not support associative and commutative symbols, it does allow linear equations, i.e., equations where each variable appears at most once on both sides. This feature enables an approximation of Diffie-Hellman exponentiation using the equation $(g^x)^y = (g^y)^x$, where g is a constant. Moreover, unlike TAMARIN, PROVERIF supports destructor symbols.

PROVERIF models protocols using a dialect of the applied pi-calculus, which is very similar to the one used in SAPIC$^+$. By default, PROVERIF expects a typed version of the applied pi-calculus, where each object is assigned a type that can be defined by the user. If no type is specified, the default is `bitstring`. Types are only used during the parsing of the model; they are not mandatory, but are very useful when designing the protocol as they help identify modeling errors.

Example 2.2.1 We export the models presented in Listings 2.1 to 2.3 into PROVERIF. The generated code is shown in the following listing. All terms are typed as `bitstring` and the communication channel is typed as `channel`. The full model, including function, event and equation declarations, is available in Appendix A.2.

```
let Initiator(k:bitstring)=
    new na:bitstring;
    event eChallenge(na);
    out(c,na);
    in(c,(m:bitstring, nb:bitstring));
    let (=h(xor(k,xor(na,nb))))=m in
        (event eAlive(na,k)).

let Responder(k:bitstring)=
    in(c,na:bitstring);
    new nb:bitstring;
    let (=na)=zero in
        (0)
    else(event eNeqForExe(na,zero);
         event eResponse(k);
         out(c,(h(xor(k,xor(nb,na))),nb))).

process
    new k:bitstring;
    ! (Initiator(k) | Responder(k))
```

Reachability properties in PROVERIF are expressed using (correspondence) *queries*, as illustrated in Listing 2.4. These specify that each time an instance of the event $\mathsf{Fact}_1(t_1, \ldots, t_n)$ occurs, a corresponding event $\mathsf{Fact}_2(t'_1, \ldots, t'_k)$ must have occurred earlier in the trace. A query can also be declared as *injective*, meaning that each instance of $\mathsf{Fact}_1(t_1, \ldots, t_n)$ must correspond to a distinct instance of $\mathsf{Fact}_2(t'_1, \ldots, t'_k)$.

```
query event(Fact1(t1,...,tn)) ==>
      event(Fact2(t'1,...,t'k))
```

Listing 2.4: Query example in PROVERIF

Example 2.2.2 The generated query for the lemma `alive` presented in Example 2.1.6 is:

```
(*alive*)
query x:bitstring, y:bitstring, i:time, j:time;
```

```
(event(eAlive(x,y))@i) ==> (event(eResponse(y))@j)
```

All variables in the left-hand side of the query are universally quantified, while those on the right-hand side are existentially quantified. PROVERIF also supports the use of time points (typed as `time`) to model temporal ordering on events in the trace.

Equivalence properties are modeled using equivalence queries, which attempt to prove a strong form of equivalence known as diff-equivalence. This notion implies the trace equivalence.

To carry out its proofs, PROVERIF translates processes into Horn clauses, which are a specific form of first-order formulas. This translation introduces strong and sound abstractions that often accelerate the verification process compared to TAMARIN. However, these abstractions may lead to inconclusive results: PROVERIF may neither find a proof nor detect an attack, in which case it outputs the conclusion `cannot be proved`.

The main strength of PROVERIF lies in its powerful abstractions, which enable fast results and enhanced automation. However, it can sometimes fail to produce conclusive results: it may find a derivation but be unable to reconstruct a corresponding attack, possibly due to overly strong abstractions, thereby complicating the analysis.

Remark 2.2.1 SAPIC$^+$ supports GSVERIF, a front-end for PROVERIF. It introduces new mechanism that reduce the number of cases where PROVERIF fails to find either a proof or an attack. This front-end follows the idea of verifying whether *"ϕ or some action that should occur only once occurred twice"*. More details can be found in [28].

2.2.3. DEEPSEC

DEEPSEC models protocols as processes in a manner similar to PROVERIF, but with more limitations: it does not support replication, and therefore only handles a bounded number of protocol sessions. DEEPSEC provides a decision procedure, guaranteeing theoretical termination, provided sufficient time and memory resources are available. It supports a class of subterm-convergent (i.e., where the right-hand side is either a ground term or a strict subterm of the left-hand side) destructor equational theories, which is strictly included within the class of equational theories supported by PROVERIF.

DEEPSEC is a specialized tool for analyzing indistinguishability properties, particularly trace equivalence, and thus cannot be used to prove reachability properties. Since PROVERIF attempts to prove diff-equivalence, a strictly stronger property than trace equivalence, DEEPSEC can sometimes successfully prove equivalence in cases where PROVERIF fails to reach a conclusion.

3. Application: the case of LAKE-EDHOC

Summary

This chapter presents a formal analysis of the EDHOC protocol (drafts 12 and 14), conducted using the SAPIC$^+$ platform introduced in the previous chapter. This work is a collaboration with Charlie Jacomme, Steve Kremer, and Maïwenn Racouchot, and was published at USENIX in 2023 [29]. Our analysis leverages the strengths of three backend tools (TAMARIN, PROVERIF, and DEEPSEC) by combining their capabilities. As part of our methodology, we also incorporate several advanced models of cryptographic weaknesses, going beyond the assumption of perfect cryptography. Our analysis uncovered vulnerabilities in the protocol, which we responsively disclosed (along with proposed mitigations) to the IETF Working Group responsible for designing EDHOC. Our feedback has been reflected in more recent drafts of the protocol.

3.1 Context 27
3.1.1 Related work 28
3.1.2 Contributions 28
3.2 Presentation of the EDHOC protocol 29
3.2.1 Protocol outline 30
3.2.2 Claimed properties 32
3.3 Models for automated verification 34
3.3.1 Protocol and properties modeling 34
3.3.2 Advanced primitive modelings 37
3.4 Results of the analysis . . 39
3.4.1 Methodology 40
3.4.2 Summary of the automated analysis of draft 12 42
3.4.3 Weaknesses in the Key Derivation 43
3.4.4 Transcript collision attacks 45
3.4.5 Key and IV-reuse 46
3.4.6 Privacy leak 47
3.4.7 Non-repudiation 47
3.4.8 Analysis of draft 14 48
3.5 Future work 48

3.1. Context

EDHOC (Ephemeral Diffie Hellman Over COSE) is a key exchange protocol designed by IETF's Lightweight Authenticated Key Exchange (LAKE) Working Group (WG) to be used on constrained devices in the Internet of Things (IoT). This kind of environment leads to strict requirements regarding implementation or use: for instance, bandwidth is limited along with the memory and therefore the design must prioritize messages of small sizes. Given today's immense number of IoT devices, security concerns are of crucial importance.

The LAKE WG was formed in 2019 and the first draft, version 00, of EDHOC was published in July 2020. Since then, 23 drafts have been released, leading to the publication of the proposed standard RFC 9528 [30] in March 2024. EDHOC is a public-key based authenticated Diffie Hellman (DH) key exchange protocol. It allows for different authentication methods, either based on signatures or static long-term DH keys. It also supports a specific version aiming at Post-Quantum security, replacing the DH key derivation by a Key Encapsulation Mechanism (KEM).

As introduced in Section 1.2, formal methods have achieved significant success in identifying attacks or proving their absence in real-world protocols. Notable examples include the formal analysis of TLS [4, 20], authentication protocols from the 5G standards [6], and the Signal instant messaging protocol [21].

Building on these successes, Vučinić *et al. invite the formal analysis community to study the protocol and contribute the results in both the symbolic and computational models* in a short paper summarizing the design of EDHOC [31].

Our work initially focused on version 12 of the draft [32], published in October 2021. When version 14 was released in May 2022, during the

[30]: Selander et al. (2024), *Ephemeral Diffie-Hellman Over COSE (EDHOC)*

[4]: Cremers et al. (2017), *A Comprehensive Symbolic Analysis of TLS 1.3*
[20]: Bhargavan et al. (2022), *A Symbolic Analysis of Privacy for TLS 1.3 with Encrypted Client Hello*

[6]: Basin et al. (2018), *A Formal Analysis of 5G Authentication*

[21]: Kobeissi et al. (2017), *Automated Verification for Secure Messaging Protocols and Their Implementations: A Symbolic and Computational Approach*

[31]: Vučinić et al. (2022), *Lightweight Authenticated Key Exchange With EDHOC*

[32]: Selander et al. (2021), *Ephemeral Diffie-Hellman Over COSE (EDHOC)*

course of our analysis, we extended our methodology to include this version as well.

3.1.1. Related work

Bruni *et al.* [33] performed a first formal analysis of EDHOC on draft 00 using the Tamarin prover. Their results and proposals were taken into account by the WG in draft 05. This analysis is however outdated as the protocol significantly changed since then. Further, it did not cover the four authentication methods (presented in Section 3.2) at the same time, nor the KEM-based variant for Post-Quantum security. Also, the analysis in [33] did not verify anonymity properties. In [25], Sapic$^+$ was used to analyse EDHOC draft 07. This model was rather simple and did not cover the four authentication methods nor the KEM-based variant and the analysis did not result into any feedback to the WG. Moreover, the symbolic models of [25, 33] both assumed perfect cryptography as usual in symbolic models, and covered less key compromise scenarios. As we demonstrate, this assumption can miss significant attacks. We therefore enrich our model with state-of-the-art advanced primitive models [34–36] as explained before, as well as a large number of possible compromise scenarios. These extensions were indeed crucial to uncover many of the weaknesses.

A systematic exploration of different compromise and threat scenarios using symbolic tools has been performed in several other works. Basin and Cremers [37] formalize and explore different notions of key and state compromise in authenticated key exchange protocols. A similar methodology has been applied to the protocols form the Noise framework [38] and second-factor authentication protocols [39] to automatically compute the strongest threat model under which a protocol is secure. We are the first to combine the Sapic$^+$ methodology of leveraging multiple tools with such a systematic threat model analysis, as well as the combinations of advanced primitive models, such as the hash models presented in [36].

[33]: Norrman et al. (2020), *Formal Analysis of EDHOC Key Establishment for Constrained IoT Devices*

[25]: Cheval et al. (2022), *Sapic$^+$: protocol verifiers of the world, unite!*

[34]: Cremers et al. (2019), *Prime, Order Please! Revisiting Small Subgroup and Invalid Curve Attacks on Protocols using Diffie-Hellman*

[35]: Jackson et al. (2019), *Seems Legit: Automated Analysis of Subtle Attacks on Protocols that Use Signatures*

[36]: Cheval et al. (2023), *Hash Gone Bad: Automated discovery of protocol attacks that exploit hash function weaknesses*

[37]: Basin et al. (2014), *Know Your Enemy: Compromising Adversaries in Protocol Analysis*

[38]: Girol et al. (2020), *A Spectral Analysis of Noise: A Comprehensive, Automated, Formal Analysis of Diffie-Hellman Protocols*

[39]: Jacomme et al. (2021), *An Extensive Formal Analysis of Multi-Factor Authentication Protocols*

3.1.2. Contributions

In this chapter, we present a comprehensive formal analysis of the EDHOC protocol: we provide a detailed model of version 12 and perform an analysis that combines several recent developments in formal methods.

- *Detailed models of the protocol, properties and primitives* (Section 3.3). We provide a detailed formal specification of version 12 of EDHOC. Our models include the four possible authentication methods, the KEM based version, and several optional checks. We also formally state the security properties claimed by the designers, see Section 3.3.1. Finally, we enrich our model with several state-of-the-art advanced models of cryptographic primitives, notably more precise models for signatures [35], DH groups [34], and hashes [36]. The latter aim to provide more realistic symbolic models of cryptography, as we explain in Section 3.3.2. To the best of our knowledge this is the first case-study to combine all of these advanced models.

- *Combination of tools*. We used the SAPIC$^+$ protocol verification platform [25], described in Chapter 2. In our analysis (Section 3.4), we exported and proved the model first with PROVERIF [17] which allows generally for faster results, then we used TAMARIN [18] which allows for more precise modeling of DH exponentiation and hash functions. We moreover exploited TAMARIN's interactive mode to find a hash transcript collision attack. In addition, we exported the model to DEEPSEC [24] for analyzing an anonymity property.
- *Methodology for a modular and comprehensive analysis* (Section 3.4.1). We followed a methodology that allowed us to analyze the protocol for a large combination of threat models, and protocol variants: we consider different threat models (key compromise and leakage), and optional protocol checks for each security property and systematically explore all of them, identify minimal threat scenarios for properties to hold, as well as maximal threat scenarios that violate a property. Moreover, all models are generated from a single file to increase maintainability, in particular to allow easy adaptation to future versions.
- *Attacks and mitigations* (Sections 3.4.2 to 3.4.7). Applying our methodology on version 12 of EDHOC, we highlight several weaknesses in the design and propose mitigations. All our findings were communicated to the LAKE WG and acknowledged. Possible mitigations were discussed and included in the version 14[1] of EDHOC, published in May 2022.
- *First results on version 14* (Section 3.4.8). We applied our methodology on version 14 and easily obtained a complete analysis of this version by leveraging the analysis pipeline set up for version 12. Our analysis confirms the security of the proposed fixes and ensures that they do not trigger other flaws.

[17]: Blanchet et al. (2022), *ProVerif with Lemmas, Induction, Fast Subsumption, and Much More*

[18]: Schmidt et al. (2012), *Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties (extended version)*

[24]: Cheval et al. (2018), *DEEPSEC: Deciding Equivalence Properties in Security Protocols Theory and Practice*

1: Draft 13 is a duplicate of draft 12 upon its expiration.

Outline We present version 12 of EDHOC in Section 3.2 along with its claimed properties. We then explain how we model the protocol, properties and weaknesses in cryptographic primitives in Section 3.3. Finally, we describe our methodology and results in Section 3.4, before presenting our future work.

Models and reproducibility All our formal models are available online at [40], along with the source code of the TAMARIN version that includes the SAPIC$^+$ platform needed to run them. Standard versions of the DEEPSEC and PROVERIF tools are sufficient and not provided at [40]. Alternatively, we provide a docker with the complete tool chain preinstalled that can be obtained and browsed using the following commands:

[40]: Jacomme et al. (2022), *LAKE EDHOC models*

```
$ docker pull protocolanalysis/lake-edhoc:draft-14
$ docker run -it protocolanalysis/lake-edhoc:draft-14 bash
```

3.2. Presentation of the EDHOC protocol

EDHOC is a MAC-then-Sign Diffie Hellman key exchange protocol designed to be suitable for IoT usage. Authentication is provided using long-term public keys: the initiator I and the responder R can each either use a static long-term DH key (STAT) or a public signature key

(SIG), resulting in four possible *methods* (Table 3.1). The protocol can be completed in three messages but allows an optional fourth message for additional security, i.e., key confirmation for the initiator.

Responder \ Initiator	SIG	STAT
SIG	Method 0	Method 2
STAT	Method 1	Method 3

Table 3.1.: Authentication methods usable with EDHOC

3.2.1. Protocol outline

The EDHOC protocol consists of four messages, the last one being optional (and expected to be replaced by one at the application layer):

1. I sends session setup information along with its identifiers and public ephemeral key share;
2. R responds with its own identifiers, public ephemeral key share and authenticates its credentials;
3. I authenticates its credentials;
4. Optionally, R explicitly confirms computation of the session key.

We now explain the protocol in more details. We suppose that the initiator I and responder R have long-term signature keys (sk_I, pk_I and sk_R, pk_R) or long-term, static DH keys (I, g^I and R, g^R). As an illustration, we display in Figure 3.1 the method 0 version, i.e., when both the initiator and responder use signature keys, without the optional fourth message.

Pseudo-random keys Before detailing each message, we introduce the intermediary pseudo-random keys (PRK) that are computed during each session from the ephemeral shared key (and possibly static DH shares) and used to derive encryption and MAC keys:

- PRK_{2e} – encryption in the second message;
- PRK_{3e2m} – encryption in the third message and MAC in the second message;
- PRK_{4x3m} – encryption in the fourth message and of application data and MAC in the third message. This is also the final key stored by both parties.

How each PRK is computed is method dependent and summarized in Table 3.2.

Meth.	PRK_{2e}	PRK_{3e2m}	PRK_{4x3m}
0	$\texttt{kdf}(G^{XY})$	PRK_{2e}	PRK_{3e2m}
1		$\texttt{kdf}(\text{PRK}_{2e}, G^{XR})$	
2		PRK_{2e}	$\texttt{kdf}(\text{PRK}_{3e2m}, G^{IY})$
3		$\texttt{kdf}(\text{PRK}_{2e}, G^{XR})$	

Table 3.2.: Derivation of intermediary keys

First message The initiator sends a message to the responder with the setup information (the method to be used, and the proposed cipher suite, i.e., a set of algorithm to be used for hashes, encryption, *etc.*), the public ephemeral key share G^X, a connection identifier C_I (with no cryptographic purpose) and some external authorization data EAD_1 (whose precise format is application dependent and left unspecified in [32]).

[32]: Selander et al. (2021), *Ephemeral Diffie-Hellman Over COSE (EDHOC)*

Second message In case R agrees on the method and proposed cipher suite, R generates an ephemeral key share Y and computes the shared ephemeral key G^{XY}, and a transcript hash TH_2 by hashing the first message, its ephemeral key G^Y and connection identifier C_R. Basically, R responds by sending its ephemeral public key share G^Y. Authentication is ensured by a mac MAC_2 (computed with PRK_{3e2m}). In methods 0 or 2, it is also ensured by an additional signature (with sk_R) on relevant

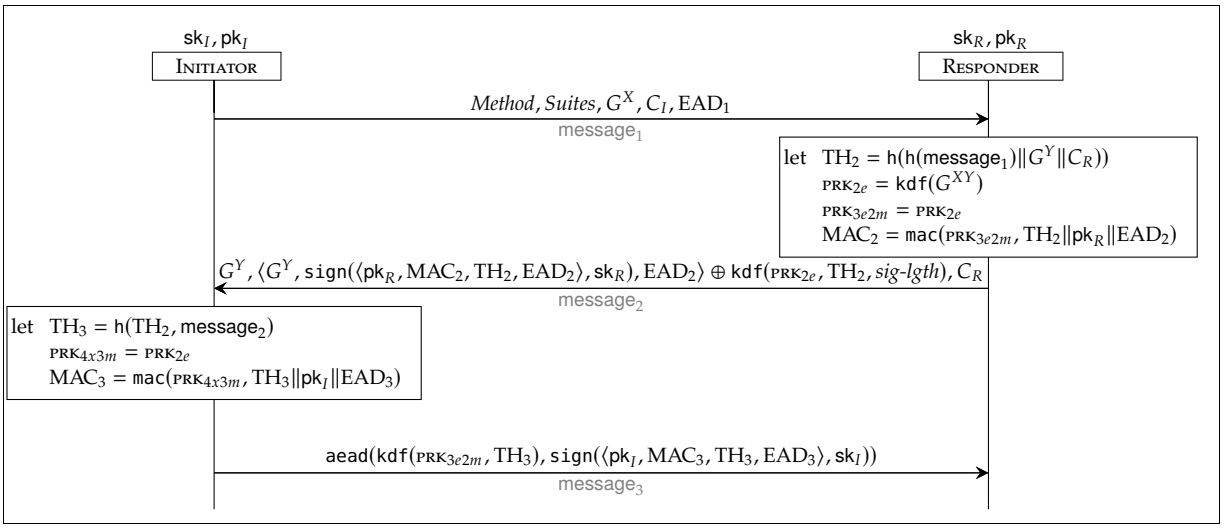


Figure 3.1.: EDHOC protocol, method 0: both initiator and responder use signature keys

data (pk_R, MAC_2, TH_2, and external authorization data EAD_2). Note that when only a mac is used, PRK_{3e2m} is derived from G^{XR}, i.e., involving the static, long-term DH key share R. The signature or mac is moreover encrypted using exclusive or with a key derived from PRK_{2e}.

Third message Once $\mathsf{message}_2$ is received, I computes the session key G^{XY} and the decryption key derived from PRK_{2e}. The signature or mac in $\mathsf{message}_2$ allows verifying R's credentials. To construct the third message I computes a mac MAC_3 on the new transcript hash TH_3, on its credentials (pk_I or G^I) and external data EAD_3 using PRK_{4x3m}. MAC_3, along with its content is then protected (depending on the method) by an additional signature and encrypted using an authenticated encryption scheme (aead) with a key derived from PRK_{3e2m}.

When R receives $\mathsf{message}_3$, it will decrypt it and verify the signature or mac to authenticate the initiator. At this point, the protocol can stop with the session key PRK_{4x3m} shared between both participants.

Fourth message The responder may send back a final message containing external authorized data encrypted with the session key to act as an acknowledgment.

A post-quantum secure variant It is noted in [32] that the proposed protocol is not post-quantum secure, in particular due to the use of a DH key exchange, relying on the hardness of the discrete logarithm. It is suggested that the protocol can be made post-quantum secure by relying on a *key encapsulation mechanism* (KEM): the resulting protocol is obtained using method 0 (instantiated with post-quantum secure signatures), and replacing the DH by a KEM-based key exchange. A detailed presentation of the KEM-based version of EDHOC is depicted in Figure 3.2.

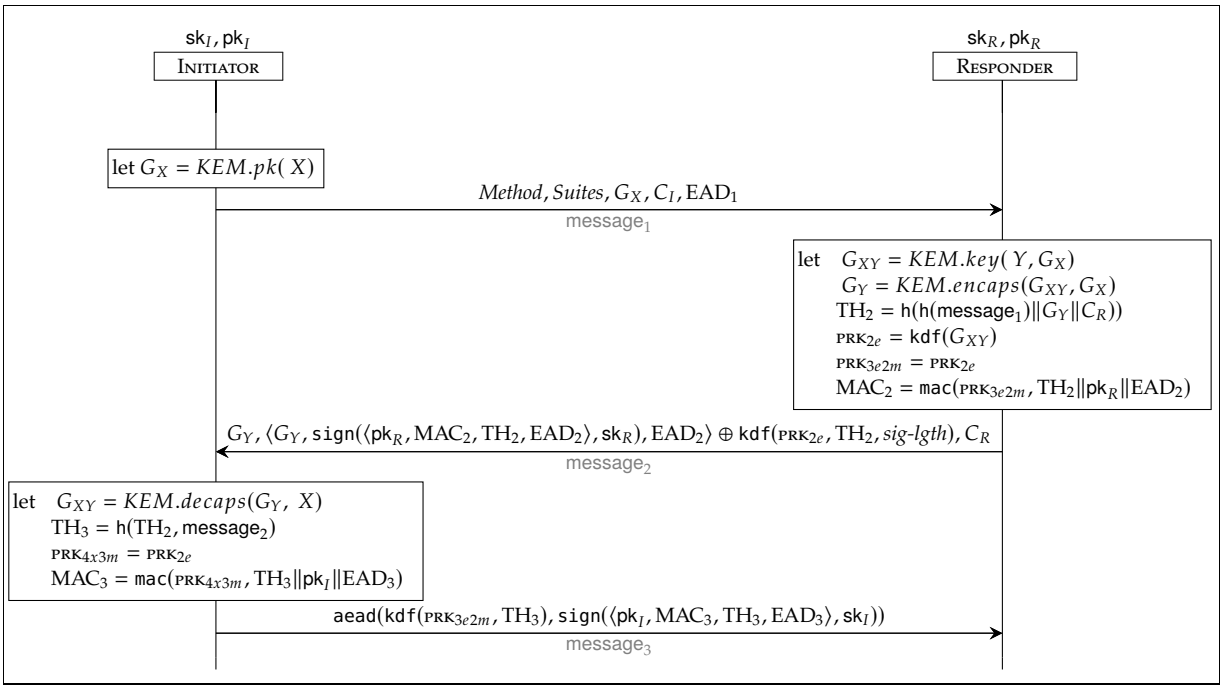


Figure 3.2.: EDHOC protocol, KEM method

3.2.2. Claimed properties

In [32], a number of properties are claimed that we present in this section and formalize and analyze later.

Confidentiality

Among the claimed properties, we have identified four confidentiality ones. From the responder point of view, a completed EDHOC exchange should provide confidentiality of the key, and the responder is also ensured that the initiator computed the same key.

Definition 3.2.1 (Explicit Key Confirmation) *[32, p. 44]*
After verifying $message_3$*, the Responder is assured that the Initiator has calculated the key* PRK_{4x3m} *(explicit key confirmation) and that no other party than the Responder can compute the key.*

However, from the initiator point of view only the confidentiality part is ensured as the fourth message is optional.

Definition 3.2.2 (Implicit Key Authentication) *[32, p. 44]*
After sending $message_3$*, the Initiator is assured that no other party than the Responder can compute the key* PRK_{4x3m}*.*

Confidentiality of the final session key must hold under strong forms of compromise: either compromise of other session keys or future compromise of the long-term keys.

Definition 3.2.3 (**Session key independence**) *[32, p. 43]*
Compromise of one session key does not compromise other session keys.

Definition 3.2.4 (**Forward secrecy**) *[32, p. 43]*
Compromise of the long-term keys (private signature or static DH keys) does not compromise the security of completed EDHOC exchanges.

Authentication

In addition to the Explicit Key Confirmation, which has already a flavor of authentication, EDHOC should also guarantee authentication of some of the exchanged data.

Definition 3.2.5 (**Authenticated transcript hash**) *[32, p. 8]*
Transcript hashes (hashes of message data) TH_2, TH_3, TH_4 (are) used for key derivation and as additional authenticated data.

Definition 3.2.6 (**Authenticated data**) *[32, p. 42]*
EDHOC adds an explicit method type and expands the message authentication coverage to additional elements such as algorithms, external authorization data, and previous messages.

Similar to confidentiality, some authentication should also be preserved even in case of compromises.

Definition 3.2.7 (**Key compromise impersonation**) *[32, p. 43]*
Compromising the private authentication keys of one party lets an active attacker impersonate that compromised party in EDHOC exchanges with other parties but [...] does not let the attacker impersonate other parties in EDHOC exchanges with the compromised party.

Identity protection

EDHOC has also been designed to offer anonymity guarantees, but mostly to the initiator. Indeed, a responder obviously leaks its identity to any initiator.

Definition 3.2.8 (**Identity protection**) *[32, p. 42]*
EDHOC protects the credential identifier of the Initiator against active attacks and the credential identifier of the Responder against passive attacks.

Non-repudiation

Depending on the method, a participant should not be able to deny having participated in a session. The draft specifies that, the other user could present to a judge its own ephemeral private key as an evidence. Note that this implies storing the ephemeral keys, therefore violating some requirements of the protocol.

Definition 3.2.9 (Non-repudiation) *[32, p. 44]*
In EDHOC authenticated with signature keys, the Initiator could theoretically prove that the Responder performed a run of the protocol by presenting the private ephemeral key, and vice versa.

3.3. Models for automated verification

3.3.1. Protocol and properties modeling

Protocol model

Overview of the protocol model. Our model takes an unbounded number of participants into account. Each participant can engage into multiple protocol sessions, acting either as the initiator, or the responder. Moreover, we model the four possible methods, depending on whether long-term signature keys or long-term DH keys are used for authentication. As explained above, the adversary chooses the intended communication partner for the initiator, and the method to be used. The attacker can also generate its own key material and choose to participate in a given session. Therefore, our model includes

- sessions between honest participants, in particular a same agent taking both the role of the initiator and the responder; hence, reflection or selfie style attacks are in the scope of the model;
- sessions where an honest initiator engages with the attacker;
- sessions where the attacker takes the role of the initiator and engages with an honest responder.

We additionally modeled the KEM-based variant: in method 0, the DH based key exchange can be replaced by a KEM-based one. Indeed, post-quantum security could be obtained by choosing both the signature and KEM schemes to be post-quantum secure.

Key compromise. We model a variety of (dynamic) key compromise scenarios. At any moment, the attacker can decide to compromise a key. In our model this is achieved by making compromise events available as parallel processes. We consider the following events, followed by the secret key outputs.

- `Compromise(k)`: compromise of either the long-term signature or long-term DH key;
- `LeakShare(x)`: compromise of the ephemeral DH key share; note, that for convenience we always raise two events, one with argument `g^s` and one with argument `s`, but we always leak the secret part `s` of the share;
- `LeakSessionKey(key)`: compromise of the session key PRK_{4x3m}.

As we will see below, our security properties will depend on whether some keys have been compromised.

Limitations. Currently, our model lacks some features. We do not model the *cipher suite negotiation*. The suite is simply a constant chosen by the attacker. However, we do verify authentication and agreement of the chosen suite. Our model also omits the *key update mechanism*, which allows achieving more efficient forward secrecy by updating the session key PRK_{4x3m} rather than re-running a new protocol session. Finally, our model does not include the optional fourth message, whose purpose is to provide explicit key confirmation to the initiator. These extensions would add complexity to our already rather extensive and large existing models.

Confidentiality properties

We analyse the four confidentiality properties discussed in Section 3.2.2. Interestingly, we are able to state a single strong confidentiality property for each of the participants that implies all four properties. We state the property for the initiator as `secretI`, and the property for the responder as `secretR` in a similar way:

```
lemma secretI:
All pkI pkR k3 k4 X GY #i #j #k.
 AcceptI(pkI,pkR,k3,k4,X,GY)@#i & KU(k4)@#j & Honest(pkR)@#k
  ==> Ex #t. Compromise(pkR)@#t & #t < #i |
      Ex #t. LeakSessionKey(k4)@#t |
      Ex #t. LeakShare(GY)@#t |
      Ex #t. LeakShare(X)@#t

lemma secretR:
All pkI pkR k4 Y GX #i #j #k.
 AcceptR(pkI,pkR,k4,Y,GX)@#i & KU(k4)@#j & Honest(pkI)@#k
  ==> Ex #t. Compromise(pkI)@#t & #t < #i |
      Ex #t. LeakSessionKey(k4)@#t |
      Ex #t. LeakShare(Y)@#t |
      Ex #t. LeakShare(GX)@#t
```

Intuitively, `secretI` (and similarly `secretR`) states that *if the attacker learns the key established (according to I) with an honest responder, then the responder's long-term key compromised before or the session key was leaked or the responder share was leaked or the initiator share was leaked.*.

This property indeed encodes all four confidentiality properties. *Forward secrecy* is encoded by requiring the initiator's long-term key to be compromised *before* the key is accepted by the initiator. In other words, if the compromise happens after the key is accepted, and the attacker learns the session key, the property is violated (or one of the other disjunct holds).

Session key independence is ensured as we only consider leakage of the current session key `k4`. Leakage of session keys from different sessions does not directly satisfy the property, and so the attacker is free to leak them. As the ephemeral key shares allow to recompute the session keys (at least in some methods) they are handled as the session key.

Finally, `secretI` encodes *Implicit Key Authentication* because the `AcceptI` is placed next to sending $\mathsf{message}_3$. Similarly, in `secretR`, the `AcceptR` event

is placed after having verified $\mathsf{message}_3$ to ensure *Secrecy after Explicit Key Confirmation*.

Authentication properties

Similarly, we model authentication properties in [40]. For instance, the following property states entity authentication for the responder, and explicit key confirmation.

[40]: Jacomme et al. (2022), *LAKE EDHOC models*

```
All pkI pkR k4 Y GX #i #k.
 AcceptR(pkI,pkR,k4,Y,GX)@#i & Honest(pkI)@#k
  ==> Ex #t X GY k3. #t < #i & AcceptI(pkI,pkR,k3,k4,X,GY)@#t |
      Ex #t. Compromise(pkI)@#t |
      Ex #t. LeakShare(Y)@#t
```

We basically state that each time the responder `pkR` accepts a session with initiator `pkI`, resulting in session key `k4`, the initiator previously accepted a session with the same parameters, or the attacker compromised keys that break the property trivially. Note that this property also implies Key Compromise Impersonation (KCI), as an attacker can compromise `pkR` without directly satisfying the property.

One may note that this is a non-injective authentication property: as stated the property does not protect against a replay attack, i.e., a responder might accept two sessions while the initiator accepted only one. Therefore we additionally prove uniqueness of the responder's accept event, i.e., there are no two accept events for the same initiator, responder, and session key. This is a stronger property that entails injectivity of the authentication. A symmetric authentication property is modeled for the initiator.

In addition, we also prove *transcript authentication* and authentication of other data such as algorithms, external authorization data, and previous messages. The modeling is the same as above except that the accept events are replaced by `AcceptRData(...)` and `AcceptIData(...)` which contain arguments `THx,EAD_x,mx` that represent the transcript hashes, the external authorization data, and previous messages.

Identity protection

We can model identity protection as an *anonymity* property, expressed as an equivalence $P_1 \approx P_2$ where P_b (for $b \in [\![1, 2]\!]$) is defined as

```
! ( new sk1; out(pk(sk1)); new sk2; out(pk(sk2));
    ! in(pkR); Init(sk1,pkR) |
    ! in(pkR); Init(sk2,pkR) |
    ! Resp(sk1) | ! Resp(sk2) |
    ! Init('skb,pk(sk1))
  )
```

This process allows to create any two pairs of users. If we consider that `Init` takes as parameters its secret signing key and the responder public key it will contact, each user can act as the initiator starting a session with an attacker chosen responder `pkR` or as a responder. We finally add a test session that depends on the *challenge bit* b: `'skb`, i.e., either `sk1` or `sk2`

initiates a *test session* with `sk1` (here, `sk1` is chosen arbitrarily; we could have chosen `sk2`). The equivalence expresses that the adversary cannot tell these two processes apart.

We note a few important details: in the test session the attacker may not choose the responder's identity. Otherwise, the attacker could act as the responder and trivially break the equivalence. Moreover, in our verification we forbid compromise of `sk1`, as otherwise, again, the attacker could act as the responder and trivially know b. Interestingly, we may however allow compromise of `sk2`.

Non-repudiation

Non-repudiation is more tricky to specify. The goal is to provide evidence to an external party (that we call a *judge*) that a designated party indeed participated in a session. The property can be split in two parts:

- *Soundness:* if evidence is accepted by a judge then the designated party did participate in the session.
- *Completeness:* if a party did participate in a session, then the other participant can present evidence to the judge that will be accepted.

We focus on soundness: this property should guarantee that the attacker cannot provide false evidence that the responder participated in a session. To show that the responder is guaranteed soundness, we use an additional judge process. The judge inputs a non-repudiation proof `proofnr` and checks its validity. If the check succeed, the judge raises an event `WasActiveR(pkr,derivedKey,proofnr)`: the proof `proofnr` claims that the responder `pkR` derived key `derivedKey`. We then verify the property:

```
All pkr derivedKey proofnr #i #j.
 WasActiveR(pkr,derivedKey,proofnr)@#i & Honest(pkr)@#j
  ==> Ex #k. DerivedRShared(pkr,derivedKey)@#k |
      Ex #k. Compromise(pkr)@#k
```

where `DerivedRShared(pkr,derivedKey)` indicates that the responder `pkr` computed the shared, ephemeral DH key `derivedKey`. A similar modeling is used for the initiator.

We also model an *injective* variant of this property: the judge will not accept two distinct evidence proving participation in a same session.

3.3.2. Advanced primitive modelings

As explained above, symbolic models generally idealize cryptography. However, recent work [34–36] has relaxed this idealization by explicitly modeling existing weaknesses that exist in concrete primitives. In this work, we aim at making our analysis as precise as possible, building on such more precise models whenever they apply to the concrete algorithms supported by EDHOC. We thus include state-of-the-art models of cryptographic primitives, and notably capture the following aspects.

[34]: Cremers et al. (2019), *Prime, Order Please! Revisiting Small Subgroup and Invalid Curve Attacks on Protocols using Diffie-Hellman*
[35]: Jackson et al. (2019), *Seems Legit: Automated Analysis of Subtle Attacks on Protocols that Use Signatures*
[36]: Cheval et al. (2023), *Hash Gone Bad: Automated discovery of protocol attacks that exploit hash function weaknesses*

Precise signature models (Sig$^{\ell}$, Sig$^{\ell}$-proof). The ES256 signature scheme is malleable, i.e., given a signature on a message it is possible to forge a different signature for the same message. The edDSA [41] signature scheme allows to (dishonestly) generate keys for which the verification of a signature will always succeed. A symbolic modeling of these and other weaknesses, that is amenable to automation in Tamarin, is proposed in [35]. In [35], two models are actually proposed: a first model explicitly adds equations for specific weaknesses, while the second allows any behaviour that does not contradict unforgeability assumptions of the signature. The first model, that we denote by Sig$^{\ell}$, is better suited for attack finding; the second model aims at security proofs, and we refer to it as Sig$^{\ell}$-proof.

[41]: Josefsson et al. (2017), *Edwards-Curve Digital Signature Algorithm (EdDSA)*

Precise DH models (DH$^{\ell}$). Similarly, more precise models have been proposed for DH exponentiation. First, there exists an identity element e in the ECDH group such that $e^x = e$ for all x. Moreover, in all curves supported in [32], there exist small subgroups of elements $(h_1, \ldots, h_n)$ [34] such that:

- many collisions of the form $h_1^x = h_2^y$ exist;
- given g^x, for all z we will have with non-negligible probability that $(h_1 g^x)^z = g^{xz}$.

Models for automating the verification with these additional capabilities have been proposed in [34] for Tamarin. We refer to these modelings as DH$^{\ell}$.

Precise hash functions (Hash$^{\ell}$). Modeling hash functions in the symbolic model is reminiscent of the random oracle model, which is unrealistic. Some hash functions allow for weaknesses as:

length extension	it exists x, y such that $h(x\|\|y) = h(h(x)\|\|y)$,
chosen-prefix collisions	for all p_1, p_2, the attacker can compute c_1, c_2 such that $h(p_1\|\|c_1) = h(p_2\|\|c_2)$[2]

2: See [42] for a survey.

These two weaknesses have been found to lead to real attacks on protocols, where [43] have exploited such weaknesses in SHA-1 to trigger transcript collision attacks over TLS.

[43]: Bhargavan et al. (2016), *Transcript Collision Attacks: Breaking Authentication in TLS, IKE and SSH*

As described in [32], EDHOC supports SHA-256 that allows length extensions, but for which there is no feasible chosen-prefix collision attack yet. Nevertheless, we consider it valuable to future-proof the protocol against such weaknesses if they were to appear, especially in the context of IoT where it may be even more difficult to deprecate an algorithm. We also note that practitioners often consider second-preimage resistance[3] to be a sufficient property. Chosen-prefix collisions do *not* contradict second-preimage collisions and it might therefore be tempting, e.g., for efficiency reasons, to consider hash functions that are only second-preimage resistant, but not necessarily collision resistant.

[32]: Selander et al. (2021), *Ephemeral Diffie-Hellman Over COSE (EDHOC)*

3: Given x, it is hard to find y such that $H(y) = H(x)$

In [36], a more precise symbolic model that accounts for length extension and chosen prefix collisions has been proposed and automated in both Tamarin and ProVerif. We refer to this precise hash model as Hash$^{\ell}$.

Precise encryption models ($\oplus^{\ell}$, AEAD^{ℓ}). The XOR encryption used in $\mathsf{message}_2$, which encrypts a triplet, is obviously malleable. We therefore added tailored equations that allow controlled modifications of each of the three elements:

- We define two function symbols and their corresponding equations to model encryption and decryption using a XOR operator.
  Note that we do not use the built-in XOR theory provided by Tamarin, even if it is more precise (e.g., with the associativity and commutativity of the XOR operator). First, it is not allowed to modify the equational properties of built-in symbols, as we do in the next point. Second, this XOR theory is not supported by ProVerif or DeepSec, preventing us from leveraging the strengths of these tools in our analysis for a malleable XOR.
- We define three additional functions and their corresponding equations to model the malleability of the XOR encryption. As mentioned, XOR encrypts a triplet in the EDHOC protocol. Each equation models the ability to modify one component of the triplet:

```
functions: mxorsign/1,mxorid/2,mxoread/2,sign/3,xord/0,id/2
equations:
  mxorsign(encxor(<x,sign(a,b,c),z>,k)) =
           encxor(<x,sign(a,xord,c),z>,k)
  mxorid(nid,encxor(<id(a,b),y,z>,k)) = encxor(<nid,y,z>,k)
  mxoread(nead,encxor(<x,y,z>,k)) = encxor(<x,y,nead>,k)
```

```
functions: encxor/2, decxor/2
equations:
  decxor(encxor(m,x),x) = m,
  decxor(encxor(m,x),m) = x
```

We denote this model by $\oplus^{\ell}$.

The AEAD used in $\mathsf{message}_3$ may only provide integrity of the plaintext and not of the ciphertext. This can be the case for the Mac-Then-Encrypt construction (even though not currently supported in [32]), or when the encoding and decoding of the message containing the encryption is not fully deterministic (implementation dependent). We therefore model the possibility to re-randomize an AEAD ciphertext, and denote this more precise model of AEAD as AEAD^{ℓ}.

It is interesting to note that although the models from [35] and [34] were designed for Tamarin, it was straightforward to port them to ProVerif and fully integrate them inside the Sapic^{+} platform.

3.4. Results of the analysis

Our analysis confirms the strong design of the EDHOC protocol as most claimed properties are satisfied when we consider *basic* threat models. However, when exploring more advanced threat models on detailed protocol models we discovered multiple weaknesses.

We present in the following the details of the identified weaknesses as well as mitigations to strengthen the protocol. While most weaknesses were found using automated analysis, we also manually identified some additional weaknesses. Those were notably discovered when exploring what were the limitations of our models w.r.t. to the draft, questioning each limitation leading to see which points of the standard required additional inspection.

Remark 3.4.1 **Disclosure process.** We reported each of the identified weakness to IETF's LAKE working group, along with recommendations and concrete proposals to mitigate them. The weaknesses were reported as git issues, followed by discussions and potential pull-requests. Due to the lack of up-to-date implementations of the draft, we could not verify the presence of these weaknesses in a concrete implementation. However, each weakness was discussed with the WG: most were found relevant and the discussions lead to the integration of major changes to the key derivation, and several other fixes in the editor version of the draft and later to draft 14.

3.4.1. Methodology

We describe here how we analysed the models described in Section 3.3, in a comprehensive and modular way. We also took maintainability of our models into account, which notably allowed for a swift update from the models of draft 12 to draft 14.

Modular threat models As explained in Section 3.3, even in basic models, we consider a Dolev-Yao attacker that completely controls the network, and can manipulate messages. Moreover, the attacker chooses the parameters of honest agents' sessions: notably, the attacker decides the identity of the peer, and the attacker can actively participate using its own set of identities. The attacker can also compromise long term secret keys.

This core threat model is extended by multiple attacker capabilities in a modular way: compromise of ephemeral shares, denoted by $\mathsf{DHShare}^{\not f}$, compromise of session keys denoted by $\mathsf{SessKey}^{\not f}$, and the advanced primitive models of Section 3.3 are atomic capabilities that can be combined. Our aim is to explore the different security properties by considering all possible combinations of those attacker capabilities.

In addition, we also implement modularly two additional checks that the protocol *may* perform:

- DH-Check – Agents refuse incoming neutral DH element;
- Cred-Check – Agents refuse sessions with their own identity as the peer.

To implement DH-Check, an agent checks whether the peer's ephemeral key share, G^X or G^Y, is the identity element. Note that when using $\mathsf{DH}^{\not f}$, DH-Check is insufficient to forbid the low order points of the elliptic curves, but we did not model a more advanced check as the WG preferred to avoid implementing even the basic DH-Check. Cred-Check is an optional check for Trust-On-First-Use (TOFU): the initiator should verify that the responder's identity is not equal to its own.

We summarize atomic attacker capabilities in Table 3.3. Our goal is to verify all possible combinations of these atomic capabilities. However, we can be more efficient and avoid the verification of redundant checks, similar to [36, 39]. The core idea is that the scenarios form a lattice ordered by inclusion, and proving that a property holds in a particular threat model implies that it also holds in all weaker scenarios. Conversely, if a property is violated inside a given threat model, say $\mathsf{Hash}^{\not f}$, it will also be violated in all stronger threat models, i.e., all threat models that contain $\mathsf{Hash}^{\not f}$. More redundant verifications can be avoided as for instance the $\mathsf{Sig}^{\not f}$ capability is weaker than the $\mathsf{Sig}^{\not f}$-proof.

We aim to identify for each property the maximal, i.e., the strongest, threat models for which the property hold, as well as the minimal, i.e., the weakest, ones for which there is an attack while pruning any redundant verifications.

[36]: Cheval et al. (2023), *Hash Gone Bad: Automated discovery of protocol attacks that exploit hash function weaknesses*
[39]: Jacomme et al. (2021), *An Extensive Formal Analysis of Multi-Factor Authentication Protocols*

Attacker Capabilities	
$\mathsf{Sig}^{\not f}$	precise signature (for attack finding)
$\mathsf{Sig}^{\not f}$-proof	precise signature (stronger guarantees)
$\mathsf{DH}^{\not f}$	precise DH with small subgroups
$\mathsf{AEAD}^{\not f}$	rerandomizable cyphertexts model
$\mathsf{Hash}^{\not f}$	Hash with length-extension and chosen prefix collisions
$\oplus^{\not f}$	malleable xor encryption
$\mathsf{SessKey}^{\not f}$	compromise of session keys
$\mathsf{DHShare}^{\not f}$	compromise ephemeral shares
Protocol Optional Checks	
DH-Check	neutral DH element check
Cred-Check	own identity check

Table 3.3.: Summary of the atomic scenarios

Using Sapic$^+$ for maintainability A core concern in our models was to factorize the code base as much as possible, both to avoid mistakes and ease maintainability with respect to updates of the draft. Our process relies on Sapic$^+$, the corresponding backends ProVerif, Tamarin and DeepSec, and the templating engine Jinja2:

1. Using Jinja2, we generate from a single file both the DH and KEM based protocols. Moreover, for the DH version, all four methods are derived from a shared template.
2. Using Sapic$^+$ and its basic preprocessing capabilities to enable or disable an attacker capability, we generate the file for a given list of atomic threat models and a target verification tool.
3. We finally perform the verification on the generated file using the corresponding tool.

Given a protocol variant, threat model, and a security property, a single terminal command allows to streamline this process and launch the verification. From this, we designed a few small scripts that batch run the verifications on all relevant threat model combinations.

Comprehensive verification with Sapic$^+$ As a benefit from using Sapic$^+$, we can choose the most suited tool for each verification: ProVerif and Tamarin are used for authentication, confidentiality and non-repudiation properties (modeled as reachability properties), and DeepSec and ProVerif for anonymity (modeled as an equivalence property). Overall, ProVerif tends to be faster, while Tamarin offers a more precise model of DH exponentiation, that notably includes inverse group elements, and thus provides stronger guarantees. Even though we did not encounter this case in our analysis, this implies that for a given threat model ProVerif could prove a property while Tamarin finds an attack.

While developing the model, we rely on ProVerif to perform efficient sanity checks and ensure that the protocol executes correctly. Interestingly, the correction result of Sapic$^+$ [25] ensures that these sanity checks obtained by ProVerif (through Sapic$^+$) carry over to the Tamarin and DeepSec models. Hence we avoid running them in these other tools, which can be cumbersome.

[25]: Cheval et al. (2022), *Sapic$^+$: protocol verifiers of the world, unite!*

To perform a comprehensive verification and explore all possible threat models, we then rely on the efficiency of ProVerif to first quickly comb through all the scenarios. Whenever ProVerif finds an attack in a given scenario, there is no need to check it with the more precise DH theory of Tamarin. For maximal scenarios in which a given property holds, we use Tamarin with a longer time out to strengthen the trust in this proof.

In more details, our global approach is as follows.

1. Run ProVerif on all threat model combinations with a timeout of 30 minutes over both the DH and the KEM variants.
2. Extract minimal threat models required for an attack, and maximal ones for a security proof. This allows us to extract a small set of interesting scenarios that can be verified by a script we prepared for reproducibility with ProVerif;
3. We finally use Tamarin with a long timeout of 24 hours to increase the trust on all maximal, i.e., strongest, threat models identified by ProVerif in the previous step.

As anonymity properties are harder to check, we do not consider any of the advanced primitive models and first verify the KEM version for a bounded number of sessions with DeepSec. If the property holds, we verify it for an unbounded number of sessions with ProVerif for both the KEM and DH protocols (but restricted to method 0 for efficiency reasons).

This full process was originally developed for draft 12 and allowed us to report several weaknesses and propose improvements to the WG, most of which were integrated into draft 14. We were then able to verify draft 14 very efficiently, by

(i) updating a single model,
(ii) running the batch ProVerif script of step 2, and
(iii) the batch Tamarin script of step 3.

Note that the same analysis and methodology could be done without Sapic$^+$ by writing manually tool specific models. However, Sapic$^+$ greatly sped up the analysis, e.g., it only required writing a single model (while the modeling languages of Tamarin and ProVerif/DeepSec fundamentally differ), sanity checks of the model are significantly faster in ProVerif. In particular, using only ProVerif, the DH model would have been less precise; using only Tamarin would complicate the anonymity analysis (support for equivalence, in particular in the presence of restrictions, is less mature in Tamarin) and exploration of the whole threat model lattice (as verification takes more time). Finally, it is likely that reaching our level of maintainability would be impossible with separate models.

We believe that the infrastructure we developed is of independent interest and could be reused out of the box for other protocol analyses. For instance, our models [40] are split into multiple parts with separate header files for easy reuse of the advanced primitive models.

[40]: Jacomme et al. (2022), *LAKE EDHOC models*

3.4.2. Summary of the automated analysis of draft 12

We summarize in Table 3.4 highlights of our analysis showing secure and insecure scenarios. Overall, most properties hold in the core threat model but we found multiple weaknesses in advanced threat models for which we propose fixes to strengthen the protocol. We provide in Table 3.5 a list of the main attacks we identified. We also specify the required threat model, the tool used to find the attack and the verification time. Moreover, we indicate whether the weakness was fixed in the draft 14 following our proposals. The extensive results of our analysis, specifying the exact lemmas and considered threat models, on both draft 12 and 14, can be found in Appendix B.1.

All our experiments were performed on a 64 core Intel(R) Xeon(R) CPU E5-4650L 0 @ 2.60GHz server with 756GB of RAM. Overall, the initial batch verification on draft 12 required 273 single threaded ProVerif calls for an accumulated runtime of 20.6 hours (verified in less than an hour with a 64 core parallelization). We extract from this batch verification the 50 maximal and minimal threat scenarios, that correspond to 3h of ProVerif accumulated runtime.

Table 3.4.: Summary of the automated analysis results over draft 12

Property	Threat model					
	Basic	AEAD$^{\not z}$	DH$^{\not z}$	DHShare$^{\not z}$ + SessKey$^{\not z}$	Hash$^{\not z}$ + DH$^{\not z}$	KEM variant
Confidentiality	✓	✓	✓	✓	×	✓
Implicit& Explicit Key Auth.	✓	✓	✓	×	✓	✓
Transcript Auth.	✓	×	✓	✓	×	✓
Algo Auth.	✓	✓	✓	✓	×	✓
Session key uniqueness	✓	✓	×	✓	×	×
Non-repudiation soundness	✓	✓	~	✓	~	✓
Identity protection	×	×	×	×	×	×

✓: property satisfied ×: violation of property ~: unclear security

3.4.3. Weaknesses in the Key Derivation

In [32], the final key material consists of the key PRK_{4x3m}, built from the ephemeral shared DH secret, and the final transcript hash TH_4. An optional key exporter is suggested that binds PRK_{4x3m} and TH_4. However, several attack traces found by our tool illustrate weaknesses in this key derivation.

[32]: Selander et al. (2021), *Ephemeral Diffie-Hellman Over COSE (EDHOC)*

Leaking ephemeral secrets breaks authentication Although one might expect authentication to rely only on the secrecy of the long-term keys, a violation of entity authentication was reported by our tool when we compromise both the ephemeral DH shares and the session key computed by the agents. This threat model is motivated by the use case where a device was fully compromised, except for the long term authentication keys that may be stored securely inside a Trusted Execution Environment, as proposed in [32, p. 48].

The attack corresponds to a Machine-in-the-Middle attack in which the attacker impersonates an initiator I to a responder R, although I did not initiate any session with R. Interestingly, this violation only occurs with methods others than 0. Indeed, even with access to all the ephemeral secrets, the attacker cannot forge a signature. In all other methods, the core issue is the use of the final session key, which is leaked, as the MAC key, allowing the attacker to forge the MAC of the last message.

For readability, we describe in Figure 3.3 the attack reported by the tool on a simplified version of EDHOC (omitting message fields that are not relevant). We suppose that I initiates a session with the attacker. The attacker then simply forwards message_1 to R. Then, using I's leaked ephemeral share X and G^Y, the attacker can compute a valid message_2 with their own identity ID_A. The initiator replies with message_3 that would not be accepted by R, as the transcripts do not match. However, after the leak of the session key, the attacker forges a MAC on the expected transcript. R accepts the forged message, and successfully finishes a session supposed to be initiated by I, though I never started a session with R.

This attack highlights an interesting property that a key exchange protocol should guarantee when an implementation with a TEE (Trusted Execution

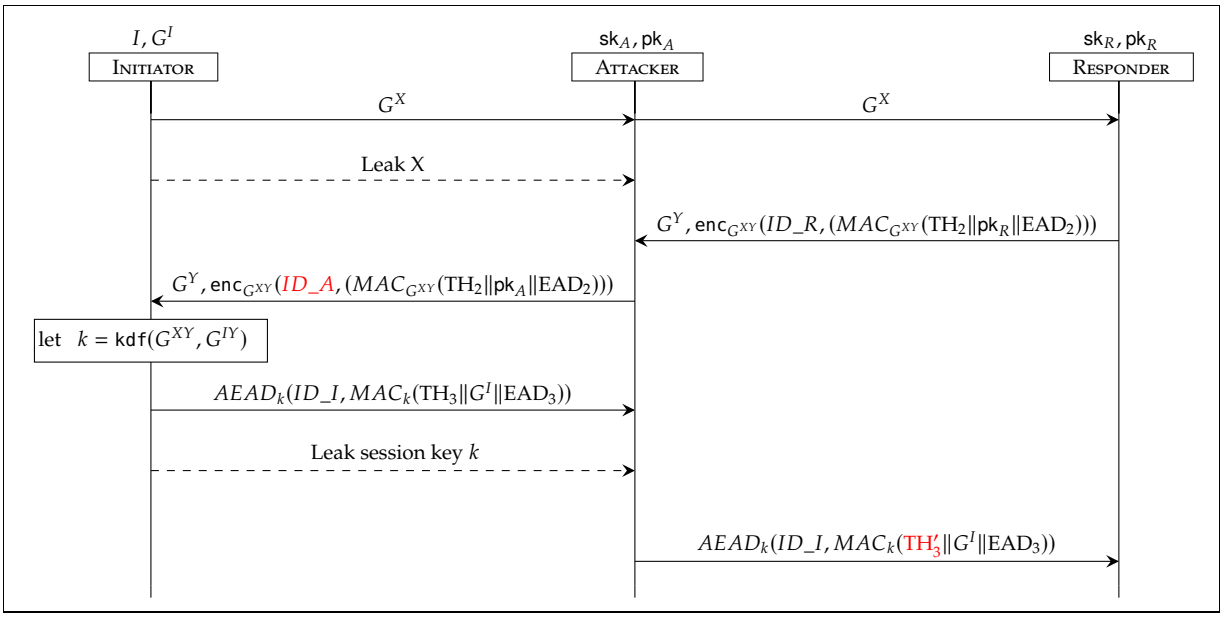


Figure 3.3.: Attack authentication in the compromise device but secure TEE setting

Environment) is considered: entity authentication should only rely on the long term authentication secret of an agent and tolerate compromise of all ephemeral secrets.

Weak data authentication It is claimed in [32] that an additional key confirmation step implies authentication of all the data, including TH_4. However, the use of the recommended key confirmation is not enforced, and only suggested. Recall that the key material consists of PRK_{4x3m} and TH_4; a natural key confirmation could only rely on PRK_{4x3m} in which case TH_4 is not authenticated.

This behaviour was highlighted by the fact that the authentication of TH_4 could be broken in multiple, implementation dependent ways, mostly because the ciphertext contained in $\mathsf{message}_3$ is directly included in the transcript to build TH_4. TH_4 authentication could then be broken when:

- the used AEAD scheme only provides integrity of the plaintext and not of the ciphertext (as it is the case for Mac-then-Encrypt);
- the encoding of $\mathsf{message}_3$ is not fully deterministic;
- when a party recomputes the last message sent from the internal protocol state, as specified in [32, Appendix E, p. 73], but using a randomized signature: this leads to two distinct, but valid versions of $\mathsf{message}_3$.

Out of those three possible behaviours, the first one was reported automatically by the tool in the weak AEAD threat model as an attack on data authentication, which lead us to investigate the other two possibilities.

Session key is weaker than exported keys In the threat model with low order DH points (Section 3.3.2), our tool reported that a dishonest party can actually control the key and force it to the neutral DH element. In the KEM variant, a dishonest responder can even fully control the

Table 3.5.: Summary of our attacks and action takens

Attack type	Requirements	Found by	Action
Initiator Impersonation	Ephemeral share and Session key leaks	ProVerif (846 s)	✓(draft 14)
Secrecy & Auth. breach & Downgrade attack	Hash Chosen-prefix collisions and no neutral DH check	Tamarin (16 h)	✓(draft 14)
Final transcript mismatch	Leak session key or Non deterministic encoding or Leak share and Malleable Sig.	ProVerif (56 s)	✓(draft 14)
Party Controlled Session key	No neutral DH check or KEM variant	ProVerif (49 s)	✓(draft 14)
Identity leak	Initiator refuses to exchange with its identity	DeepSec (1 s)	To be clarified
Duplicated non-repudiation	Malleable Sig.	ProVerif (81 s)	Judged irrelevant
AEAD Key/IV reuse	Message recomputation from stored state	Manual	✓(draft 14)

key. While session key uniqueness was not a property claimed by the standard, there was no clear reason why the exported keys would offer this property but not the session key.

We defined and explicitly checked in our analysis for this property, that we called session key uniqueness (this property is also referred to as contributiveness or attacker key control).

Our proposal We propose to fix all the previous weaknesses by adding a final key derivation as part of the protocol that would depend both on PRK_{4x3m} and TH_4. Such a session key, intuitively derived as $\mathtt{kdf}(\text{PRK}_{4x3m}, \mathsf{TH}_4)$, has the following benefits:

- the session key differs from the MAC key;
- a key confirmation of the session key implies authentication of all the data;
- a dishonest party cannot control the final value of the session key.

An additional benefit of this proposal is that the final state of the protocol is simplified: only the session key needs to be stored, instead of a key and a transcript. We considered alternatives but preferred the above solution: notably they would increase the complexity of implementations, or solve less issues at once. One option was to enforce key uniqueness by checking that the ephemeral DH shares are not low order points of the curve in use.

3.4.4. Transcript collision attacks

Using Tamarin, which has a better support for modeling chosen prefix collision attacks, we found that EDHOC was vulnerable to transcript collision attacks. This is a Machine-in-the-Middle (MitM) attack where the attacker intercepts both ephemeral key shares, and replaces them with the neutral element of the DH group. Then, both parties will have a shared secret equal to this neutral element. Such a MitM attack is generally avoided as parties will not agree on the transcript hash. However, the attacker is able to use the fields (of unbounded length) C_R, C_I or EAD to inject chosen data which allows to create a collision on the hash of

the transcripts. Similar attacks were discussed on TLS, IKE and SSH in [43].

[43]: Bhargavan et al. (2016), *Transcript Collision Attacks: Breaking Authentication in TLS, IKE and SSH*

Recall that a chosen-prefix collision allows for any p_1, p_2 to compute c_1, c_2 such that $h(p_1||c_1) = h(p_2||c_2)$. We detail the attack below by displaying the *expected* transcript computation T_E, followed by I's transcript T_I and the R's transcript T_R, where we denote by e the neutral DH element and c_1,c_2 the bytes used to obtain a chosen prefixed collision.

$$\begin{aligned} T_E &:= \text{method} \parallel \text{Suites} \parallel G^X \parallel C_I \parallel EAD_1 \parallel G^Y \parallel C_R \\ T_I &:= \quad 0 \quad \parallel \text{Suites}_I \parallel G^X \parallel C_I \parallel EAD_1 \parallel \ \mathrm{e} \ \parallel c_1 \parallel g^y \parallel C_R \\ T_R &:= \quad 0 \quad \parallel \text{Suites}_I \parallel \ \mathrm{e} \ \parallel C_I \parallel \quad c_2 \quad \parallel G^Y \parallel C_R \end{aligned}$$

Due to the chosen prefix collisions, we indeed have that

$$h(0||\text{Suites}_I||G^X||C_I||EAD_1||e||c_1) = h(0||\text{Suites}_I||e||C_I||c_2)$$

and appending to both sides $(G^Y||C_R)$ preserves the collision due to the length extension property. A variant of this attack also allows to perform a downgrade attack. While no hash function currently in the standard does have this weakness, we deemed it important to try to future-proof the protocol against such attacks, especially in the IoT context where it can be difficult to deprecate an algorithm. We proposed several solutions to the WG, which after discussion opted for an alternative which is to change the order of the arguments inside the hash computation. We verified which order would allow to avoid such attacks, and the hash transcript is now computed as:

$$h(G^Y||C_R||h(method||Suites||G^X||C_I||EAD_1))$$

The attack was initially found automatically on the KEM version in 16h using Tamarin. On the DH version of the protocol we can easily find the attack using Tamarin's interactive mode.

3.4.5. Key and IV-reuse

While modeling the protocol and browsing the standard, we manually identified the following weakness. As EDHOC may be deployed on unreliable networks, it may be necessary to re-send the last message.

Definition 3.4.1 (Packet loss resilience) *[32, Appendix E, p. 73] An EDHOC implementation MAY keep the protocol state to be able to recreate the previously sent EDHOC message and resend it.*

Consider $\mathsf{message}_3$, which is of the form $\mathsf{aead}(SIG, IV, K)$, where IV and K are derived from PRK_{4x3m}. We observe that the computation of the IV and the key K are deterministic w.r.t. to the state of the protocol. However, if a randomized signature scheme is used, such as ECDSA[4], the plaintext is also randomized. A well-known security issue of AEADs [44, Section 3.1] is that an IV must only be used once for a given ciphertext and key; otherwise no confidentiality nor integrity property may be assumed. Thus, recomputing the message actually leads to a nonce reuse for AEADs, and must not be allowed when using randomized signatures.

4: Elliptic Curve Digital Signature Algorithm

Following our recommendation, such behaviour is now forbidden in draft 14 and the security risk is mentioned. Note that we did not try to capture this behaviour in our models of draft 12 as this behaviour completely disappeared in draft 14.

3.4.6. Privacy leak

The initiator's identity should be protected against active attackers that do not own a public key that I is willing to exchange with. However, to mitigate a form of selfie attacks, a party may check that it is not receiving one of its own identities (modeled by Cred-Check). When modeling this particular feature, we observed that an active attacker can then test whether a receiver and an initiator share the same identity by observing the failure of the exchange due to the aforementioned test. In turn, as an active attacker can learn the identity of any responder, it can also learn the identity of any initiator.

It was clarified with the WG that this mitigation was intended for the Trust-On-First-Use setting, in which anybody can learn the identity of initiators, as they are willing to exchange with anybody. However, this lead to the observation that in the classical setting, where each party has a list of trusted identities, an active attacker can test whether an identity is allowed or not. This leads to a privacy leak when the list of trusted identities for I is only missing I's identity.

Details regarding the verification results for anonymity are given in Appendix B.1. As our privacy study relies on DeepSec which does not support the DH exponentiation, our results are only for the KEM based method. In ProVerif we additionally studied the DH version, but only for method 0 because of scalability issues.

3.4.7. Non-repudiation

We specified non-repudiation (soundness) as an injective property. This models that one proof presented to the judge should correspond to a single session. We identified several weaknesses that could make the non-repudiation guarantees unclear. Notably, it could be tricky to implement a judge that would need to count the number of sessions performed by a party because:

- first, if the parties accept low-order points or the identity group element, many sessions may share the same PRK_{4x3m};
- second, if the signature is malleable (i.e., given one signature for a message, it is possible to create a second distinct signature for the same message), multiple different proofs can be produced for the same session.

In addition, we witnessed that a dishonest agent could make itself repudiable by choosing a weak signature key for which verification will succeed over any message for some signature schemes and notably ED25519. After discussions, the WG considered that such considerations were side cases that did not need to be detailed in the draft.

3.4.8. Analysis of draft 14

Overall, we introduced in draft 14 a major change in the key derivation by adding a final key, we updated the order of the transcript elements, and we avoid an IV and key reuse over an AEAD. We performed again the analysis of the most significant scenarios with ProVerif, confirming that previous weaknesses are mitigated by our changes. In addition, we used Tamarin with a long time-out of 24 hours and 8 cores per job to try to obtain stronger guarantees over some scenarios. While many scenarios produced time-outs, we were able to strengthen the guarantees for 13 advanced threat models for an accumulated Tamarin runtime of 82 hours.

3.5. Future work

This work focuses on drafts 12 and 14 of the EDHOC protocol. Since then, several newer drafts have been released, up to draft 23. A computational analysis of draft 23 has been conducted [45], prior to the final release of the standard in March 2024.

[45]: Ferreira (2024), *Computational Security Analysis of the Full EDHOC Protocol*

Now that EDHOC has been standardized, our goal is to keep our models aligned with the final version of the standard, providing a reference model in the same spirit as a reference implementation. The process of updating the models should greatly benefit from the modular architecture that we set up. There are also multiple small ways to improve our models, notably by adding the optional message four or the integrated session key update mechanism, but that may come at the cost of a loss of automation and a reduction of the scope of threat models considered.

In addition, the LAKE working group is designing a new authentication method for the protocol based on pre-shared keys [46]. This upcoming extension to the standard will require formal analysis, and we plan to incorporate it into our reference model.

[46]: López Pérez et al. (2024), *EDHOC is a New Security Handshake Standard: An Overview of Security Analysis*

Part II.

The challenge of Associative and Commutative symbols in Tamarin

Focus on the Tamarin Prover 4.

Summary

This chapter focuses on the Tamarin Prover [18]. We first recall the fundamental concepts used to model messages and security properties, as introduced in Chapter 2, and clarify the particularities of the tool in each of these areas and for modeling protocols. We also explore the internal mechanisms used by Tamarin to conduct its proofs, with particular attention to how the capabilities of the Dolev-Yao attacker are internally represented through intruder rules. These concepts form the foundation for the work presented in Chapter 5.

4.1 Syntax & Semantics . . . 50
4.1.1 Messages as terms: reminder and singularities . 50
4.1.2 Protocols as rewrite systems 52
4.1.3 Properties as first-order formulas 57
4.2 Intruder deduction and automated reasoning . . . 58
4.2.1 Internal representation: Dependency graphs 58
4.2.2 Intruder Rules 60
4.2.3 Internal representation: Normal Dependency Graphs 63
4.2.4 Automated reasoning . . . 64

Outline In this chapter, we recall in Section 4.1 the essential notations used for the syntax and semantics, while highlighting the key differences specific to Tamarin in comparison with Sapic$^+$. In Section 4.2, we describe the internal mechanisms of the tool used during the proof to conduct the deductions made by the attacker and the automated reasoning.

4.1. Syntax & Semantics

The core syntax and semantics of Tamarin are similar to those of Sapic$^+$, introduced in Chapter 2, except for modeling protocols.

4.1.1. Messages as terms: reminder and singularities

In Tamarin, messages are modeled as terms in an order-sorted term algebra equipped with an equational theory.

All **terms** are assigned a top sort *msg*, which encompasses two incomparable subsorts:

- sort *fr*: models nonces, keys, and other random values,
- sort *pub*: models publicly known values such as agent names.

For each sort s, there is a countable but infinite set of variables, denoted $\mathcal{V}_s$. The set of all variables is called $\mathcal{V}$. If $x \in \mathcal{V}_s$, we write $x : s$. We write also $\sim x$ for $x : fr$ and $\$x$ for $x : pub$. Similarly, for each sort, there is a countably infinite set of names $\mathcal{N}_s$, and $\mathcal{N}$ denotes the set of all names.

We continue to write $vars(t)$ for the set of variables occurring in the term t, as introduced in Section 2.1.1.

Function symbols model cryptographic operations. We assume a signature Σ_{Op} defining the operators and their arities. This set is partitioned into two subsets: Σ_{pub} for public symbols (usable by the adversary for constructing recipes) and Σ_{priv} for private symbols (reserved for the protocol).

We also adopt standard notations for *positions* within terms.

Definition 4.1.1 *A position p in a term t is a finite sequence of integers, where the empty sequence is denoted by $[]$. We write $t|_p$ for the subterm of t at position p, where*

1. *if $p = []$, then $t|_p = t$,*
2. *if $p = [i] \cdot p'$ [1] for a positive integer i and a sequence p', and $t = f(t_1, \ldots, t_n)$ for $f/n \in \Sigma_{Op}$ and $1 \leq i \leq n$, then $t|_p = t_i|_{p'}$,*
3. *otherwise $t|_p$ is not defined and p is not a valid position.*

1: the concatenation of $[i]$ and p'

Example 4.1.1 Consider the protocol introduced in Figure 1.3. The Responder sends to the Initiator the term $t = \langle \mathsf{h}(n_a \oplus (k \oplus n_b)), n_b \rangle$. Let p be the position $[1, 1, 2, 1]$ and p' the position $[2]$. Then $t|_p = k$ and $t|_{p'} = n_b$. The position $[3]$ is not a valid position.
Subterms of t include k and n_b, as well as $\mathsf{h}(n_a \oplus (k \oplus n_b))$ and $n_a \oplus (k \oplus n_b)$.

The semantics of function symbols in Σ_{Op} are defined via equations, forming an **equational theory**. Additionally, TAMARIN supports several **built-in theories** for common cryptographic primitives. A built-in theory encapsulates the necessary function symbols along with their associated equations. However, using those function symbols in user-defined theories is forbidden as this might break the correctness of hard-coded optimizations. Figure 4.1 lists the built-in theories available in TAMARIN.

- Diffie-Hellman exponentiation
- Symmetric/Asymmetric encryption
- Hashing
- Signing (and revealing versions)
- Bilinear pairing
- XOR
- Multiset
- Natural numbers

Figure 4.1.: Built-in theories supported by TAMARIN

Example 4.1.2 Appendix A.3 presents the complete TAMARIN model of the $CRXOR$ protocol. The function symbols and equations used are imported entirely from built-in theories:

```
builtins: xor,hashing
```

This commands adds $\{\oplus/2, \mathsf{zero}/0, \mathsf{h}/1\}$ to Σ_{Op} and introduces the following equations to the equational theory:

$$x \oplus y = y \oplus x\,,\ (x \oplus y) \oplus z = x \oplus (y \oplus z)\,,\ x \oplus 0 = x\,,\ x \oplus x = 0\,,\ x \oplus y \oplus x = y$$

Equational theories in TAMARIN are required to satisfy the *finite variant property* (FVP) [47].

[47]: Comon-Lundh et al. (2005), *The Finite Variant Property: How to Get Rid of Some Algebraic Properties*

Definition 4.1.2 (Finite Variant Property [47]) *Given a convergent*[2] *equational theory with the FVP, for any term t, one can compute a finite set of terms $t_1, \ldots, t_n$ (each t_i being a normalized instance of t) such that for any substitution σ, there exists $i \in \{1, \ldots, n\}$ and a substitution ϕ satisfying $t\sigma\downarrow_E = t_i\phi$.*

2: Definition 2.1.3

This property enables efficient symbolic protocol analysis by using a precomputation to replace the equational theory. A complete set of variants (i.e., terms as all t_i described in Definition 4.1.2) modulo E for a term t is denoted $\lceil t \rceil^E$.

Example 4.1.3 Consider the term $t = x \oplus y$ and the equational theory $\mathscr{E}_{XOR}$[3]. The set $\lceil t \rceil^{\mathscr{E}_{XOR}}$ includes (with the corresponding substitution):

3: $x \oplus y = y \oplus x\,,\ (x \oplus y) \oplus z = x \oplus (y \oplus z)\,,\ x \oplus 0 = x\,,\ x \oplus x = 0\,,\ x \oplus y \oplus x = y$

0	$\{x \mapsto a, y \mapsto a\}$
$b \oplus c$	$\{x \mapsto a \oplus b, y \mapsto a \oplus c\}$
a	$\{x \mapsto a \oplus b, y \mapsto b\}$
b	$\{x \mapsto a, y \mapsto a \oplus b\}$
y	$\{x \mapsto 0\}$
x	$\{y \mapsto 0\}$

All x, y, a, b, c are variables.

Given two terms $s, t \in T_{\Sigma_{Op}}(\mathcal{V}, \mathcal{N})$, a substitution σ is a *unifier* modulo the equational theory $\mathcal{E}$ if $s\sigma =_{\mathcal{E}} t\sigma$. We denote $unif_{\mathcal{E}}(s =^? t)$ the complete set of unifiers of s and t modulo $\mathcal{E}$.

Example 4.1.4 Consider the terms $t_1 = n_a \oplus k \oplus n_b$ and $t_2 = n'_a \oplus k \oplus n'_b$ (with n'_b a variable and n_a, k, n_b, n'_a names), and the equational theory $\mathcal{E}_{XOR}$. The substitution $\sigma = \{n'_b \mapsto n_a \oplus n_b \oplus n'_a\}$ is an unifier modulo $\mathcal{E}_{XOR}$ of t_1, t_2:

$$\begin{aligned} t_2\sigma &= n'_a \oplus k \oplus n_a \oplus n_b \oplus n'_a \\ &=_{\mathcal{E}_{XOR}} n'_a \oplus n'_a \oplus n_a \oplus k \oplus n_b \\ &=_{\mathcal{E}_{XOR}} 0 \oplus n_a \oplus k \oplus n_b \\ &=_{\mathcal{E}_{XOR}} n_a \oplus k \oplus n_b = t_1 = t_1\sigma \end{aligned}$$

4.1.2. Protocols as rewrite systems

In Tamarin, security protocols are modeled using *labeled multiset*[4] *rewriting rules*. These rules operate on multisets of *facts*, which represent both the behavior of the protocol participants and the capabilities of the adversary.

4: We use the superscript $^\sharp$ to denote operations on multisets such as $\cup^\sharp$.

Syntax

Facts represent the current state of the system. They are constructed by applying elements from the fact signature Σ_{Fact} to terms:

$$\mathcal{F} = \{\mathsf{F}(t_1, \ldots, t_n) \mid t_i \in T_{\Sigma_{Op}}(\mathcal{V}, \mathcal{N}), \mathsf{F}/n \in \Sigma_{Fact}\}$$

We use facts to model all elements of the protocol. The set of facts $\mathcal{F}$ always include certain internal facts that represent:

- the adversary's knowledge (`K`),
- the freshness of atomic terms (`Fr`),
- messages sent (`Out`) or received (`In`) over the network.

The remaining facts can be freely defined by the user to model any information deemed useful, such as the secrecy of a key and the honesty of a session (as action facts), or the internal state of the protocol (as state facts).

$\mathcal{F}$ is partitioned into *linear* and *persistent* facts: linear facts may be consumed only once, while persistent facts may be consumed arbitrarily often. Persistent facts are prefixed with the symbol `!`. The set of multisets of facts is denoted by $\mathcal{F}^\sharp$ and the set of multisets of ground facts is written $\mathcal{G}^\sharp$. A fact is ground when built on ground terms and we lift this notion to sets, lists, etc. of facts.

The function $set(\cdot)$ converts a multiset into a set.

Definition 4.1.3 *Labeled multiset rewriting rules are written*

$$id : l -[\ a\ ]\!\rightarrow r \quad or \quad id\ \frac{l_1 \quad \dots \quad l_k}{r_1 \quad \dots \quad r_n}\ a_1, \dots, a_m$$

where id is a unique identifier and $l = \{l_1, \dots, l_k\}$, $a = \{a_1, \dots, a_m\}$, and $r = \{r_1, \dots, r_n\}$ are multisets of facts.
We say its name *is $name(ri) = id$, its* premises *are $prems(ri) = l$, its* conclusions *$concs(ri) = r$, and its* actions *$acts(ri) = a$.*

Each application of a rule transforms a multiset of facts S into a new multiset. Intuitively, when applying a rule on S, all linear facts from the premises of the rule are consumed, i. e., removed from S, and the facts from the conclusion are added to S. The actions a serve as labels on the transition, and are later used in the definition of security properties.

Given a set of multiset rewriting rules R, their ground instances are denoted $ginsts(R)$. We denote by $lfacts(l)$ the multiset of linear facts and by $pfacts(l)$ the set of persistent facts in l.

Example 4.1.5 We model the $CRXOR$ protocol (described in Figure 1.3) using the rules shown in Listings 4.1 to 4.4.

The first rule, `Setup`, generates a fresh `~k` using the linear fact `Fr`. This nonce represents the symmetric key shared between the two participants at the beginning of the session. The rule then produces two persistent facts: `!InitLTK(~k)` and `!RespLTK(~k)`, allowing the initiator and responder to later access the symmetric key as many times as needed. The rule is annotated with the action `OnlyOnce`, which is used (along with a restriction[5]) to ensure that only one key is generated per session.

The second rule models the initiator's first behavior: it generates a fresh nonce `na` and sends it via the `Out` fact. It also produces the linear fact `Initiated(~na)`, indicating that the initiator knows the nonce `na` at this state of the system. The rule is annotated with the action `Challenge(~na)`, which will later be used to express a security property.

The third rule models the responder's behavior in response to the challenge. It first receives a value $na_{\mathcal{V}}$ (treated by the responder as a variable, not a fresh term), using the `In` fact. It then looks up the shared long-term key `k` using the persistent fact `!RespLTK`, and generates a fresh nonce `nb` using the `Fr` fact. Finally, the responder outputs the pair of `nb` and the hash of the XOR of the three values via the `Out` fact. The rule is annotated with several actions:

- `Response(~k)` and `NeqForExe(`$na_{\mathcal{V}}$`,zero)` for use in a security property, and
- `Neq(`$na_{\mathcal{V}}$`,zero)` as restriction, preventing the rule from being applied when the received value is null.

Finally, the last rule represents the completion of the session for the initiator. It retrieves the key `k` using `!InitLTK`, inputs the pair of a message $m_{\mathcal{V}}$ and the value $nb_{\mathcal{V}}$, and consumes the fact `Initiated(~na)`, ensuring that this rule can only be applied after the initiator's previous rule. This rule is annotated with the action `Alive(~na,~k)` for expressing a security property, and with the action `Eq(h(~na ⊕ ~k ⊕ `$nb_{\mathcal{V}}$`),`$m_{\mathcal{V}}$`)` as a restriction, ensuring that the received message has the expected form.

```
rule Setup:
  [ Fr(~k) ]
--[ OnlyOnce() ]->
  [ !InitLTK(~k), !RespLTK(~k) ]
```

Listing 4.1: $CRXOR$ protocol: setup
5: See paragraph **Trace Restrictions** for details

```
rule initiator:
  [ Fr(~na) ]
--[ Challenge(~na) ]->
  [ Out(~na), Initiated(~na) ]
```

Listing 4.2: $CRXOR$ protocol: initiator beginning

```
rule responder:
  [In(na_V),Fr(~nb),!RespLTK(~k)]
--[ Response(~k)
  , Neq(na_V,zero)
  , NeqForExe(na_V,zero) ]->
  [ Out(<h(na_V⊕~k⊕~nb),~nb>) ]
```

Listing 4.3: $CRXOR$ protocol: responder

```
rule initiator2:
  [ Initiated(~na), !InitLTK(~k),
    In(<m_V,nb_V>) ]
--[ Alive(~na,~k)
  , Eq(h(~na⊕~k⊕nb_V),m_V) ]->
  []
```

Listing 4.4: $CRXOR$ protocol: initiator ending

Types of rules In Tamarin, we work with three types of multiset rewriting rules.

The first type is a special built-in rule that generates fresh values, called the *fresh* rule:

$$\text{Fresh} : -[]\rightarrow \mathsf{Fr}(\sim n)$$

This rule does not require any premises and can be applied whenever a random fresh nonce $\sim n$ is needed. Tamarin guarantees that each application of the fresh rule produces a unique value (no two fresh rule instances produce the same value). It is also the only rule allowed to produce a Fr fact in its conclusion.

The second type of rules is the set of *message deduction rules*, denoted *MD* and shown in Figure 4.2. These rules model a Dolev-Yao-style adversary that has full control over the network and can apply cryptographic operators to known values.

$$\begin{aligned} MD = \{\ & \mathsf{Out}(x)-[]\rightarrow!\mathsf{K}(x),\ !\mathsf{K}(x)-[\ \mathsf{K}(x)\ ]\rightarrow\mathsf{In}(x), \\ & \mathsf{Fr}(x:fr)-[]\rightarrow!\mathsf{K}(x:fr),\ []-[]\rightarrow!\mathsf{K}(x:pub)\ \} \\ \cup\ \{\ & !\mathsf{K}(x_1),\ldots,!\mathsf{K}(x_n)-[]\rightarrow!\mathsf{K}(\mathsf{f}(x_1,\ldots,x_n))\mid \mathsf{f}/n\in\Sigma_{Op}\ \} \end{aligned}$$

- Messages sent by protocol participants are placed into Out facts and stored in the adversary's knowledge via K facts.
- Messages are sent to participants as In facts.
- The adversary can create its own random values and knows all public values.
- It can apply functions symbols from the signature to known messages via the rules in the third line of *MD*.

Figure 4.2.: Message Deduction Rules *MD*

In these rules, we do not explicitly deal with the equations modeling the properties of cryptographic operators as all terms are considered modulo the equational theory.

The third type of rules are the *protocol rules*:

Definition 4.1.4 ([18]) *A protocol rule is a multiset rewriting rule* $l-[\ a\ ]\rightarrow r$ *satisfying:*

- ***P1*** *It does not contain fresh names,*
- ***P2*** *K and Out facts do not occur in l,*
- ***P3*** *K, In and Fr facts do not occur in r,*
- ***P4*** $vars(r) \subseteq vars(l) \cup \mathcal{V}_{pub}$

A protocol is defined as a finite set of protocol rules.

Condition **P1** and the restriction on Fr facts in **P3** ensure that only the fresh rule is able to generate fresh names. The restrictions on others facts in **P2** and **P3** enforce that any network communication must go through the adversary's knowledge via *MD*. Condition **P4** ensures that no free variables appear in a conclusion, so that the origin of any generated value is always known.
All rules presented in Listings 4.1 to 4.4 are protocol rules.

Semantics

After modeling the protocols with labeled rewriting rules, we formalize an execution as a sequence of **states**, representing the adversary's knowledge, the messages on the network, information about freshly generated values,

and the protocol's state [18], connected through a **global transition system**.

[18]: Schmidt et al. (2012), *Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties (extended version)*

Transitions The formal semantics of a set of multiset rewriting rules R is defined by a *labeled transition relation*.

Definition 4.1.5 ([18]) *A* ***labeled transition relation*** $\rightarrow_R \subseteq \mathcal{G}^\sharp \times \mathcal{G}^\sharp \times \mathcal{G}^\sharp$ *is defined by the step rule*

$$\frac{ri = id : l \text{-}[\, a \,]\!\rightarrow r \in_E ginsts(R) \qquad lfacts(l) \subseteq^\sharp S \qquad pfacts(l) \subseteq S}{S \xrightarrow{set(a)}_R ((S \setminus^\sharp lfacts(l)) \cup^\sharp r)}$$

where S *is the current state.*

As we perform multiset rewriting modulo equations E, we use $\in_E$ which denotes set inclusion modulo E.

Linear facts are consumed according to the multiplicity of their appearance, so we use multiset inclusion to check that all facts in $lfacts(l)$ occur sufficiently often in S. For persistent facts, we only need to check that each fact in $pfacts(l)$ occurs in S. The successor state is computed by removing all consumed linear facts and adding the generated facts. The action associated to the transition is the set of actions of the rule instance.

Executions With the above notions, we define an *execution* and its associated *trace*:

Definition 4.1.6 *An* ***execution*** *e of a protocol, specified by a set of multiset rewriting rules P, is the alternating sequence of states and rule instances:*

$$S_0, (l_1 \text{-}[\, a_1 \,]\!\rightarrow r_1), S_1, \ldots, S_{n-1}, (l_n \text{-}[\, a_n \,]\!\rightarrow r_n), S_n$$

such that:

- ***E1*** $S_0 = \emptyset$,
- ***E2*** *for all* $i \in \{1, \ldots, n\}$, $(S_{i-1}, (l_i \text{-}[\, a_i \,]\!\rightarrow r_i), S_i)$ *is a valid step according to Definition 4.1.5,*
- ***E3*** *for all* $i, j \in \{1, \ldots, n\}$, *if* $(l_i \text{-}[\, a_i \,]\!\rightarrow r_i) = (l_j \text{-}[\, a_j \,]\!\rightarrow r_j) = ([] \text{-}[]\!\rightarrow Fr(\sim n))$, *then* $i = j$.

Conditions **E1** and **E2** ensure that an execution begins with $\emptyset$ and each step is valid. Condition **E3** ensures that an instance of the FRESH rule is used at most once for any given fresh name in the execution, ensuring uniqueness.

The associated trace is the sequence of the actions that happened during the execution:

$$trace(e) = [set(a_1), \ldots, set(a_n)]$$

We write $execs(P)$ for the set of all executions of P, and $traces(P)$ for the corresponding set of traces. We define the *observable trace* $\overline{tr}$ of a trace tr as the subsequence of all non-silent actions in tr, i.e., those not equal to $\emptyset$.

Example 4.1.6 We illustrate the notions of executions and trace with the example of $CRXOR$ protocol, recall in Listing 4.5. An execution of this protocol is:
$\emptyset$, $[]\text{—}[]\rightarrowtail \mathsf{Fr}(\sim k)$,
$\{\mathsf{Fr}(\sim k)\}$, `Setup`,
$\{!\mathsf{InitLTK}(\sim k), !\mathsf{RespLTK}(\sim k)\}$, $[]\text{—}[]\rightarrowtail \mathsf{Fr}(\sim n_a)$,
$\{!\mathsf{InitLTK}(\sim k), !\mathsf{RespLTK}(\sim k), \mathsf{Fr}(\sim n_a)\}$, `initiator`
S_4, $\mathsf{Out}(\sim n_a)\text{—}[]\rightarrowtail !\mathsf{K}(\sim n_a)$,
S_5, $!\mathsf{K}(\sim n_a)\text{—}[\ \mathsf{K}(\sim n_a)\ ]\rightarrowtail \mathsf{In}(\sim n_a)$,
S_6, $[]\text{—}[]\rightarrowtail \mathsf{Fr}(\sim n_b)$,
S_7, `responder`,
S_8, $\mathsf{Out}(\langle h_n, \sim n_b\rangle)\text{—}[]\rightarrowtail !\mathsf{K}(\langle h_n, \sim n_b\rangle)$,
S_9, $!\mathsf{K}(\langle h_n, \sim n_b\rangle)\text{—}[\ \mathsf{K}(\langle h_n, \sim n_b\rangle)\ ]\rightarrowtail \mathsf{In}(\langle h_n, \sim n_b\rangle)$,
S_{10}, `initiator2`,
$\{!\mathsf{InitLTK}(\sim k), !\mathsf{RespLTK}(\sim k), !\mathsf{K}(\sim n_a), !\mathsf{K}(\langle h_n, \sim n_b\rangle)\}$

where:
$S_4 = \{!\mathsf{InitLTK}(\sim k), !\mathsf{RespLTK}(\sim k), \mathsf{Initiated}(\sim n_a), \mathsf{Out}(\sim n_a)\}$
$S_5 = \{!\mathsf{InitLTK}(\sim k), !\mathsf{RespLTK}(\sim k), \mathsf{Initiated}(\sim n_a), !\mathsf{K}(\sim n_a)\}$
$S_6 = S_5 \cup^\sharp \{\mathsf{In}(\sim n_a)\}$
$S_7 = S_6 \cup^\sharp \{\mathsf{Fr}(\sim n_b)\}$
$S_8 = \{!\mathsf{InitLTK}(\sim k), !\mathsf{RespLTK}(\sim k), \mathsf{Initiated}(\sim n_a), !\mathsf{K}(\sim n_a), \mathsf{Out}(\langle h_n, \sim n_b\rangle)\}$
$S_9 = \{!\mathsf{InitLTK}(\sim k), !\mathsf{RespLTK}(\sim k), \mathsf{Initiated}(\sim n_a), !\mathsf{K}(\sim n_a), \mathsf{K}(\langle h_n, \sim n_b\rangle)\}$
$S_{10} = S_9 \cup^\sharp \{\mathsf{In}(\langle h_n, \sim n_b\rangle)\}$ and $h_n = \mathsf{h}(n_a \oplus \sim k \oplus \sim n_b)$

The corresponding observable trace is, to be read left to right and up to bottom:

$$\overline{\mathsf{tr}} = \begin{bmatrix} \mathsf{OnlyOnce}(), & \mathsf{Challenge}(\sim n_a), & \mathsf{K}(\sim n_a), \\ \mathsf{Response}(\sim k), & \mathsf{Neq}(n_a, \mathtt{zero}), & \mathsf{NeqForExe}(n_a, \mathtt{zero}), \\ \mathsf{K}(\langle h_n, \sim n_b\rangle), & \mathsf{Alive}(\sim n_a, \sim k), & \mathsf{Eq}(h_n, m) \end{bmatrix}$$

```
rule Setup:
  [ Fr(~k) ]
--[ OnlyOnce() ]->
  [ !InitLTK(~k), !RespLTK(~k) ]

rule initiator:
  [ Fr(~na) ]
--[ Challenge(~na) ]->
  [ Out(~na), Initiated(~na) ]

rule responder:
  [ In(na),Fr(~nb),!RespLTK(~k) ]
--[ Response(~k)
  , Neq(na,zero)
  , NeqForExe(na,zero) ]->
  [ Out(<h(na⊕~k⊕~nb),~nb>) ]

rule initiator2:
  [ Initiated(~na), !InitLTK(~k),
    In(<m,nb>) ]
--[ Alive(~na,~k)
  , Eq(h(~na ⊕ ~k ⊕ nb),m) ]->
  []
```

Listing 4.5: $CRXOR$ protocol rules

Trace Restrictions In TAMARIN, we can restrict the set of traces considered valid for a protocol P, using `restriction`. These are written as trace formulas, described in detail in Section 2.1.3. A restriction allows the user to constrain the space of possible executions. If a trace violates a restriction, TAMARIN ignores the corresponding execution and explores alternative ones.

Formally, with $\rho_1, \ldots, \rho_n$ restrictions and ϕ the property we want to prove, we verify

$$(\bigwedge_{i=1}^{n} \rho_i) \Rightarrow \phi$$

Example 4.1.7 Some restrictions have been written for the running example $CRXOR$. The first ensures the uniqueness of the `Setup` rule. Since this rule is the only one whose actions contain the `OnlyOnce()` fact, a trace containing this fact twice would indicate that the rule was applied twice, meaning two long-term key were generated. We prevent this with `restriction unique`.

The second restriction ensures that when applying the `initiator2` rule, the received message is of the expected form. The third restriction is the opposite: it prevents traces where the received value is null, in this context.

```
restriction unique:
  "All #i #j. OnlyOnce()@#i & OnlyOnce()@#j ==> #i = #j"
```

```
restriction equality: "All x y #i. Eq(x,y)@#i ==> x=y"

restriction inequality: "not (Ex x #i. Neq(x,x)@#i)"
```

4.1.3. Properties as first-order formulas

The first-order logic used to express reachability properties in Tamarin is the one detailed in Section 2.1.3. Example 2.1.6 is a valid example in Tamarin.

In addition to reachability, Tamarin supports the verification of indistinguishability properties by proving *observational equivalence* [48] defined over sets of multiset rewriting rules.

[48]: Basin et al. (2015), *Automated Symbolic Proofs of Observational Equivalence*

Intuitively, if two protocols are observationally equivalent, then their executions appear identical to the adversary. In other words, whenever the attacker can apply a rule in one instance using a certain recipe, they must also be able to apply the same rule with the same recipe in the other instance. A `diff` operator is used to model where both protocols differ.

Example 4.1.8 Consider two protocols $P_{1.1}, P_{1.2}$ with a single rule `Out_Hash` that generates a nonce and outputs either the hash of the nonce in $P_{1.1}$, or the nonce in $P_{1.2}$:

```
rule Out_Hash:
[ Fr(~ltk) ]-->[ Out(diff(h(~ltk),~ltk)) ]
```

Since the adversary cannot distinguish the hash of a nonce from the nonce itself, the two protocols $P_{1.1}, P_{1.2}$ are observationally equivalent.

Now consider two other protocols $P_{2.1}, P_{2.2}$:

```
rule Initialize:
[ Fr(~x) ]-->[ Test(~x), Out(~x) ]

rule Register_pk:
[ Fr(~ltk), In(x), Test(diff(x,y)) ]-->[ Out(~ltk) ]
```

The `Initialize` rule generates a nonce, outputs it and marks this state with the fact `Test`.
In $P_{2.1}$, the `Register_pk` rule is applicable if `Test(x)` can be consumed, requiring that the rule `Initialize` has previously sent this fact along with the same `x`. The rule `Register_pk` receives the same value from the network.
In $P_{2.2}$, the logic is similar with the name `y`, but the `Register_pk` rule expects a different variable `x` potentially distinct from the value generated in `Initialize`.
If the adversary can successfully trigger `Register_pk` by sending arbitrary values, it learns that it is interacting with $P_{2.2}$. Therefore, the two protocols $P_{2.1}, P_{2.2}$ are *not* observationally equivalent.

A complete formalization of observational equivalence is beyond the scope of this work. For full theoretical details, we refer the reader to [48].

4.2. Intruder deduction and automated reasoning

The multiset rewriting semantics has certain limitations when it comes to proving security properties. In particular, the absence of history and causal dependencies between the states of an execution makes it difficult to reconstruct an attack with traces alone. Moreover, the message deduction rules as presented may allow redundant steps to occur during an execution. To address these issues, Tamarin internally represents executions using *dependency graphs* and models the ability of an active Dolev-Yao attacker with *normal message deduction*.

4.2.1. Internal representation: Dependency graphs

Tamarin internally uses a different representation of the protocol execution and adversary deduction: *dependency graphs*. In these graphs, each node corresponds to a rule instance, and edges connect conclusions that produce facts to premises that consume them, thus expressing the dependencies between nodes.
Let P be a protocol, consider a sequence $I \in (ginsts(P \cup MD))^*$ [6] and $D \subseteq \mathbb{N}^2 \times \mathbb{N}^2$. A dependency graph is a couple (I, D), where:

- $idx(I)$ denotes the set of nodes in the graph:
  If $I = [\mathsf{l}_1 -[\ \mathsf{a}_1\ ]\rightarrow \mathsf{r}_1, \ldots, \mathsf{l}_n -[\ \mathsf{a}_n\ ]\rightarrow \mathsf{r}_n]$, then $I_i = \mathsf{l}_i -[\ \mathsf{a}_i\ ]\rightarrow \mathsf{r}_i$ for each $i \in idx(I)$.
- D defines the edges in the graph:
  An edge between two facts $(i, u), (j, v)$ is written $(i, u) \rightarrowtail (j, v)$ and it links the uth conclusion of rule I_i to the vth premise of rule I_j, where both facts are by definition equal modulo $\mathcal{E}$.

6: P denotes in this context the protocol rules of P

A conclusion of a dependency graph dg is a pair (i, u), where i is a node of dg and $u \in idx(\mathsf{r}_i)$: it refers to the uth conclusion fact of I_i, denoted $(\mathsf{r}_i)_u$. A premise of dg is similarly defined as a pair (i, u) with i a node of dg and $u \in idx(\mathsf{l}_i)$: it is the uth premise fact of I_i, denoted $(\mathsf{l}_i)_u$. A conclusion or premise is said to be *linear* if the associated fact is linear. As $I = [\mathsf{l}_1 -[\ \mathsf{a}_1\ ]\rightarrow \mathsf{r}_1, \ldots, \mathsf{l}_n -[\ \mathsf{a}_n\ ]\rightarrow \mathsf{r}_n]$, the trace of dg is $trace(dg) = [set(\mathsf{a}_1), \ldots, set(\mathsf{a}_n)]$.

Definition 4.2.1 (Dependency Graph [18]) *Let R be a protocol. We say that the pair $dg = (I, D)$ is a dependency graph modulo $\mathcal{E}$ for R if I is a sequence of $ginsts_{\mathcal{E}}(R \cup \text{Fresh})$, $D \subseteq \mathbb{N}^2 \times \mathbb{N}^2$, and dg satisfies the conditions:*

DG1 *For every edge $(i, u) \rightarrowtail (j, v) \in D$ it holds that $i \leq j$ and the conclusion fact of (i, u) is equal to the premise fact of (j, v).*
DG2 *Every premise of dg has exactly one incoming edge.*
DG3 *Every linear conclusion of dg has at most one outgoing edge.*
DG4 *The Fresh instances are unique.*

We denote the set of all dependency graphs modulo $\mathcal{E}$ for R by $dgraphs_{\mathcal{E}}(R)$.

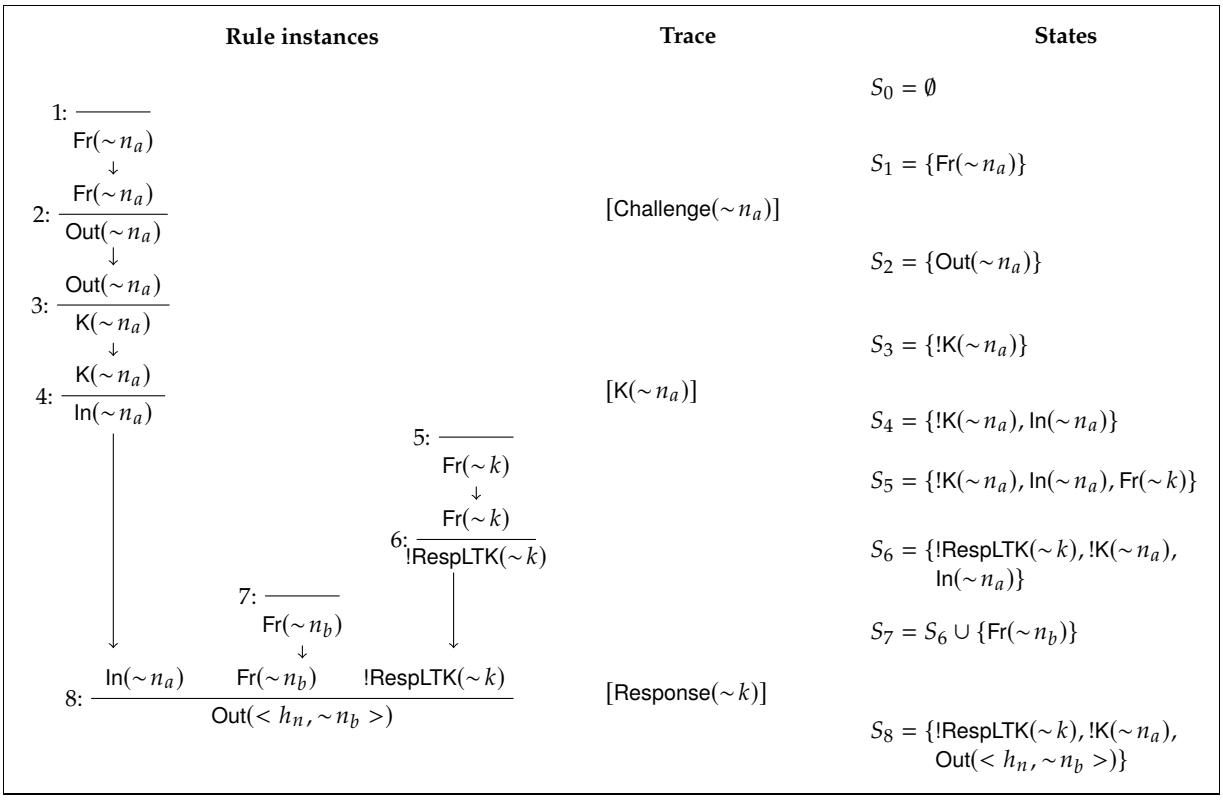


Figure 4.3.: Example execution of $(P_{CRabr} \cup MD)$. $h_n = \mathsf{h}(\sim n_a \oplus \sim k \oplus \sim n_b)$

Example 4.2.1 Consider an abridged version of the $CRXOR$ protocol with only the three rules below, forming the set P_{CRabr}:

```
rule Setup:
  [ Fr(~k) ]--[ ]->[ !RespLTK(~k) ]

rule initiator:
  [ Fr(~na) ]--[ Challenge(~na) ]->[ Out(~na) ]

rule responder:
  [ In(na),Fr(~nb),!RespLTK(~k) ]
--[ Response(~k) ]->
  [ Out(<h(na⊕~k⊕~nb),~nb>) ]
```

Figure 4.3 shows a possible execution as a dependency graph, along with its corresponding trace and states.

To illustrate the concept of edges, consider the following two examples:

- $(1,1) \rightarrowtail (2,1)$ links the fresh fact for n_a (the only fact produced by FRESH) to the only fresh fact for n_a consumed by the `initiator` protocol rule.
- $(6,1) \rightarrowtail (8,3)$ connects the persistent fact `!RespLTK(~k)` produced by the protocol rule `Setup` to the third premise consumed by the `responder` protocol rule.

We write $traces(dg)$ for the traces of a dependency graph dg. It is straightforward to observe that the dependency graphs have the same traces as the labeled transition system executions.

Lemma 4.2.1 ([18], Lemma 4) *Let E be an equational theory. Then for all protocols P:*

$$\{trace(e) \mid e \in execs(P \cup MD)\} =_E \{t \mid t \in traces(dg), dg \in dgraphs_E(P \cup MD)\}$$

Since reasoning modulo the full equational theory can be computationally expensive, Tamarin internally uses rule variants and normal forms for terms, reasoning only modulo AC. We therefore split the equational theory $\mathscr{E}$ into two parts: the convergent part $\mathscr{E}'$, and the associativity and commutativity axioms for all AC symbols $\mathscr{AC}$.

We extend the notion of normal form $\downarrow_E$ from terms to facts, rules and dependency graphs, i.e., a fact, a rule or a dependency graph is in normal form, written $\downarrow_E - normal$, if all terms inside the fact, rule or dependency graph are in normal form. We also extend the notion of variants to multiset rewriting rules by simply considering rules as terms and the required new function symbols as free.

For a protocol P and an equational theory $\mathscr{E}'$, we denote the variants of all protocol rules induced by the equational theory $\lceil P \rceil^{\mathscr{E}'}$.

The following lemma states that this is sufficient to use rule variants and terms in normal form, and only reasons modulo AC:

Lemma 4.2.2 ([18], Lemma 5) *For all protocols P,*

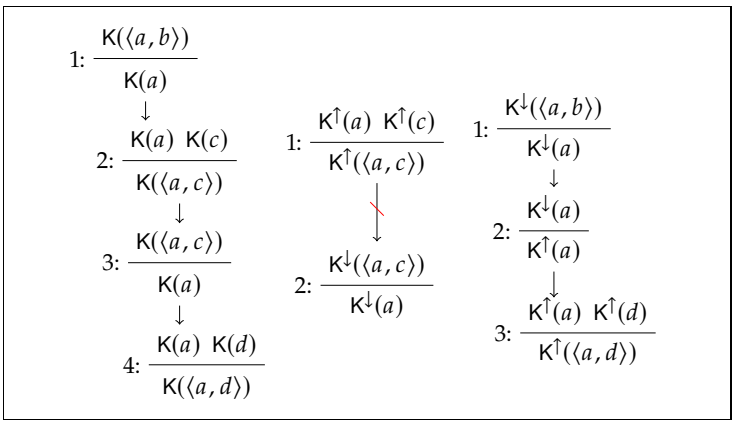
$$dgraphs_{\mathscr{E}}(P \cup MD)\downarrow_{\mathscr{E}'} =_{\mathscr{AC}} \{dg \mid dg \in dgraphs_{\mathscr{AC}}(\lceil P \cup MD \rceil^{\mathscr{E}'}) \wedge dg \downarrow_{\mathscr{E}'} - normal\}$$

Proof. By extension of Lemma 5 in [18]. The proof is analogous as the equational theory has the same properties. □

4.2.2. Intruder Rules

Normal deduction rules

Unfortunately, normalized graphs are still insufficient to enable full automation, as intruder deduction can easily loop even for simple equational theories, as illustrated in Figure 4.4.



Consider the pairing operator $\langle \cdot, \cdot \rangle$ and the two functions fst() and snd(). On the left, we show such a looping deduction: the adversary starts with a pair $\langle a, b \rangle$, extracts a, then adds c only to extract a again, and so on.

Figure 4.4.: Example message deduction graphs for pairing: the left column represents a redundant deduction, the middle column an impossible deduction with ordered K-facts, and the right column shows a normalized and shorter deduction with the same final conclusion as on the left column.

To avoid such loops, Tamarin internally uses **normal deduction rules** to model the capabilities of an active adversary. It splits K facts into $\mathsf{K}^{\downarrow}$ (for decomposition) and $\mathsf{K}^{\uparrow}$ (for composition) facts. Figure 4.5 gives the normal deduction rules for an equational theory $\mathcal{X}\oplus\mathcal{R}$ consisting of pairing and an exclusive-or operator.

These rules are organized into four categories:

- **Communication rules**: Irecv represents the reception by the attacker of a message from the network (i.e., the adversary gains knowledge) and Isend models the transmission of a message to the network (possibly constructed by the adversary).
- **Construction rules**: These rules are used to compose larger terms, e.g., by applying the pairing function as in Figure 4.4 using pair. Construction rules involve only $\mathsf{K}^{\uparrow}$ facts: for a function symbol $\mathsf{f}/k \in \Sigma_{Op}$, the premises of the corresponding construction rule consist of k distinct $\mathsf{K}^{\uparrow}$ facts over variables $x_1, \ldots, x_k$, and the conclusion is $\mathsf{K}^{\uparrow}(\mathsf{f}(x_1 \ldots, x_k))$, representing the application of the function symbol to those variables.
- **Deconstruction rules**: These rules are used to decompose terms, e.g., by applying fst() or snd() as in Figure 4.4 via fst and snd. Both the first premise and the conclusion of a deconstruction rule are $\mathsf{K}^{\downarrow}$ facts, the other premises are $\mathsf{K}^{\uparrow}$ facts to allow the adversary to build the terms for the other arguments of the function. Tamarin includes an algorithm that computes such rules for user-defined convergent equational theories respecting the finite variant property [49].
- **Coerce rule**: A new Coerce rule allows to convert a $\mathsf{K}^{\downarrow}$ fact into a $\mathsf{K}^{\uparrow}$ fact, but not the reverse. The IFresh and Pub rules are versions with ordered K-facts of their counterpart in *MD*.

$$\textsc{Irecv}\frac{\mathsf{Out}(x)}{\mathsf{K}^{\downarrow}(x)} \quad \textsc{Isend}\frac{\mathsf{K}^{\uparrow}(x)}{\mathsf{In}(x)}[\mathsf{K}(x)] \quad \textsc{Coerce}\frac{\mathsf{K}^{\downarrow}(x)}{\mathsf{K}^{\uparrow}(x)}$$

$$\textsc{IFresh}\frac{\mathsf{Fr}(x:fr)}{\mathsf{K}^{\uparrow}(x:fr)} \quad \textsc{Pub}\frac{}{\mathsf{K}^{\uparrow}(x:pub)}$$

Construction rules:

$$\textsc{pair}\frac{\mathsf{K}^{\uparrow}(x_1)\ \mathsf{K}^{\uparrow}(x_2)}{\mathsf{K}^{\uparrow}(\langle x_1, x_2\rangle)} \qquad 0\frac{}{\mathsf{K}^{\uparrow}(0)} \qquad \oplus\frac{\mathsf{K}^{\uparrow}(x_1)\ \mathsf{K}^{\uparrow}(x_2)}{\mathsf{K}^{\uparrow}(x_1 \oplus x_2)}$$

Deconstruction rules:

$$\textsc{fst}\frac{\mathsf{K}^{\downarrow}(\langle x, y\rangle)}{\mathsf{K}^{\downarrow}(x)} \qquad R_{\oplus.1}\frac{\mathsf{K}^{\downarrow}(x_1 \oplus x_2)\ \mathsf{K}^{\uparrow}(x_2)}{\mathsf{K}^{\downarrow}(x_1)}$$

$$\textsc{snd}\frac{\mathsf{K}^{\downarrow}(\langle x, y\rangle)}{\mathsf{K}^{\downarrow}(y)} \qquad R_{\oplus.2}\frac{\mathsf{K}^{\downarrow}(x_1 \oplus x_2)\ \mathsf{K}^{\uparrow}(x_2 \oplus x_3)}{\mathsf{K}^{\downarrow}(x_1 \oplus x_3)}$$

Figure 4.5.: Set of normal deduction rules $ND_{\mathcal{X}\oplus\mathcal{R}}$

[49]: Dreier et al. (2017), *Beyond Subterm-Convergent Equational Theories in Automated Verification of Stateful Protocols (extended version)*

Among these, Irecv is the only rule that produces a $\mathsf{K}^{\downarrow}$ fact without consuming one. This ensures that the adversary always begins by applying deconstruction rules. Only later can construction rules be applied to build larger terms, as seen on the right of Figure 4.4. Since construction rules produce only $\mathsf{K}^{\uparrow}$ facts, the adversary cannot return to deconstruction rules, thus avoiding loops like the one shown in the center of Figure 4.4.

Internally, Tamarin knows the normal deduction rules for all built-in equational theories.

Deconstruction rules computation

As mentioned, Tamarin relies on an algorithm to compute the deconstruction rules for user-defined function symbols.

Let us consider a function symbol $\mathsf{f}/k \in \Sigma_{Op}$ and the deconstruction rule

$$\mathsf{f}_0\frac{\mathsf{K}^{\downarrow}(x_1) \quad \mathsf{K}^{\uparrow}(x_2) \quad \ldots \quad \mathsf{K}^{\uparrow}(x_k)}{\mathsf{K}^{\downarrow}(\mathsf{f}(x_1, \ldots, x_k))}$$

where $x_1, \ldots, x_k \in \mathcal{V}$. We denote by $\mathcal{RR}_{\mathsf{f}}$ the set of rewriting rules obtained from $\lceil \mathsf{f}_0 \rceil^{\mathscr{C}}$: A rewriting rule $l \to r$, with $l, r \in T_{\Sigma_{Op}}(\mathcal{V}, \mathcal{N})$[7], is constructed from the variant of $\mathsf{f}_0\ \dfrac{\mathsf{K}^{\downarrow}(tp_1) \quad \mathsf{K}^{\uparrow}(tp_2) \quad \ldots \quad \mathsf{K}^{\uparrow}(tp_n)}{\mathsf{K}^{\downarrow}(tc)}$ by setting $l = \mathsf{f}(tp_1, \ldots, tp_n)$ and $r = tc$.

7: As the right-hand side of a rewriting rule introduces no new variables, there exist index i and positions $p_1, \ldots, p_i$ such that $r \in T_{\Sigma_{Op}}(l|_{p_1}, \ldots, l|_{p_i})$.

$$\mathcal{RV}_{\mathtt{f}} \triangleq \bigcup_{l\to r\in\mathcal{R}\mathcal{R}_{\mathtt{f}}} \bigcup_{p\in P(l,r)} ctxtdrules(l,p,r)$$

where

- $P(l,r) = \{p \mid \exists q.\ l|_p = r|_q$ and $l|_p = r|_q$ is a maximal common subterm$\}$
- $ctxtdrules(l,p,r) = \{[\mathsf{K}^{\downarrow}(l|_{p'})] \cdot cprems(l,p') \text{–}[\,]\text{→}\mathsf{K}^{\downarrow}(r) \mid p'$ strictly above p and $p' \neq []\}$
- $cprems(l,p') = seq(\{\mathsf{K}^{\uparrow}(l|_{\tilde{p}}) \mid \tilde{p} \neq [] \wedge \tilde{p}$ has a sibling above or equal to p'$\})$

Figure 4.6.: Algorithm for computing $\mathcal{RV}_{\mathtt{f}}$, the set of deconstruction rules for $\mathtt{f}$, w.r.t. the equational theory $\mathcal{E}$ [49]. $seq(\cdot)$ converts sets into sequences.

We introduce also the notion of *common maximal subterm*:

Definition 4.2.2 ([49] Definition 2) *A common subterm t of a rewriting rule $l \longrightarrow r$ is a term such that there are p and q such that $t = l|_p = r|_q$.*
*A **common maximal subterm** t of a rewriting rule $l \longrightarrow r$ is a common subterm of $l \longrightarrow r$ such that there is no common subterm $t' \neq t$ such that t is a subterm of t'.*

For each rewriting rule in $\mathcal{RR}_{\mathtt{f}}$ and for each position $p' \neq []$ strictly above a position p (corresponding to a *common maximal subterm*), we compute a deconstruction rule for which the term $l|_{p'}$ is in a $\mathsf{K}^{\downarrow}$ fact and the terms $l|_{p''}$, where p'' has a sibling equal or above p' , are required in a $\mathsf{K}^{\uparrow}$ fact. Figure 4.6 formalizes this algorithm.

Example 4.2.2 To illustrate the deconstruction rules generation algorithm, consider $\mathcal{E}_{BS}$ the equational theory of blind signatures, used for instance in Okamoto's e-voting protocol mentioned in Chapter 5:

```
functions: sign/2, blind/2, unblind/2
equations: unblind(blind(m,b),b) = m,
           unblind(sign(blind(m,b),sk),b) = sign(m,sk)
```

We apply the algorithm from Figure 4.6. First, the rewriting rules derived from $\lceil_{\mathtt{unblind}_0} \dfrac{\mathsf{K}^{\downarrow}(x_1) \quad \mathsf{K}^{\uparrow}(x_2)}{\mathsf{K}^{\downarrow}(\mathtt{unblind}(x_1,x_2))} \rceil^{\mathcal{E}_{BS}}$ are

$$\mathcal{RR}_{\mathtt{unblind}} = \left\{ \begin{array}{c} \mathtt{unblind}(\mathtt{blind}(m,b),b) \longrightarrow m, \\ \mathtt{unblind}(\mathtt{sign}(\mathtt{blind}(m,b),sk),b) \longrightarrow m \end{array} \right\}$$

Consider the rewriting rule

$$\mathtt{unblind}(\mathtt{sign}(\mathtt{blind}(m,r),k),r) \longrightarrow \mathtt{sign}(m,k)$$

The common maximal subterms here are m and k, located at position $p_1 = [1,1,1]$ and $p_2 = [1,2]$ respectively.
The positions strictly above p_1 are $P'_1 = \{[1,1],[1]\}$ and those strictly above p_2 are $P'_2 = \{[1]\}$.
This gives us three positions p'_i, resulting in three deconstruction rules for this rewriting rule.

$p'_{1.1} = [1,1]$ Here, $l|_{p'_{1.1}} = \mathtt{blind}(m,r)$ and its sibling positions are $P''_{1.1} = \{[1,2],[2]\}$, corresponding to the terms k and r, respectively. The resulting deconstruction rule is shown opposite.

$$\frac{\mathsf{K}^{\downarrow}(\mathtt{blind}(m,r)) \quad \mathsf{K}^{\uparrow}(k) \quad \mathsf{K}^{\uparrow}(r)}{\mathsf{K}^{\downarrow}(\mathtt{sign}(m,k))}$$

N1. The dependency graph dg is $\downarrow_{\mathcal{R}_{\mathcal{BP}_e}}$-normal.
N2. There is no multiplication rule that has a premise fact of the form $\mathsf{K}^{\uparrow}(s * t)$ and all conclusion facts of the form $K^{x}(s * t)$ are conclusions of a multiplication rule.
N3. If there are two conclusions c and c' with conclusion facts $K^{x}(m)$ and $K^{x'}(m')$ such that $m =_{AC} m'$ and either $x = x' = \uparrow$ or $x = \downarrow y$ and $x' = \downarrow y'$ for $y,\ y' \in \{d, e\}$, then $c = c'$.
N4. All conclusion facts $\mathsf{K}^{\uparrow}(\mathtt{f}(t_1, \ldots, t_n))$ where $\mathtt{f}$ is an invertible function symbol are conclusions of the construction rule for $\mathtt{f}$.
N5. If a node i has a conclusion $K^{\downarrow y}(m)$ for $y \in \{d, e\}$ and a node j has a conclusion $\mathsf{K}^{\uparrow}(m')$ with $m =_{AC} m'$, then $i < j$ and either $root(m)$ is invertible or the node j is an instance of coerce.
N6. There is no node $[K^{\downarrow d}(a),\ K^{\uparrow}(b)] -[]\rightarrowtail K^{\downarrow e}(c\hat{}d)$ where c does not contain any fresh names and $nifactors(d) \subseteq_{AC} nifactors(b)$.
N7. There is no construction rule for $\sharp$ that has a premise of the form $\mathsf{K}^{\uparrow}(s \sharp t)$ and all conclusion facts of the form $\mathsf{K}^{\uparrow}(s \sharp t)$ are conclusions of a construction rule for $\sharp$.
N8. The conclusion of a deconstruction rule for $\sharp$ is never of the form $K^{\downarrow d}(s \sharp t)$.
N9. There is no node $[K^{\downarrow d}(a), \mathsf{K}^{\uparrow}(b)] -[]\rightarrow K^{\downarrow e}([d]c)$ such that c does not contain any fresh names and $nifactors(d) \subseteq_{ACC} nifactors(b)$.
N10. There is no node i labeled with $[K^{\downarrow d}([t_1]p), K^{\downarrow d}([t_2]q)] -[]\rightarrowtail K^{\downarrow d}(\hat{e}(p, q)\hat{}c)$ such that there is a node j labeled with $[K^{\downarrow d}(\hat{e}(p, q)\hat{}c), \mathsf{K}^{\uparrow}(d)] -[]\rightarrowtail K^{\downarrow d}(\hat{e}(p, q)\hat{}e)$, an edge $(i, 1) \rightarrowtail (j, 1)$, $nifactors(ti) \subseteq_{ACC} nifactors(d)$ for $i = 1$ or $i = 2$, and $\hat{e}(p, q)$ does not contain any fresh names.
N11. There is no node $[K^{\downarrow d}([a]p), K^{\downarrow d}([b]q)] -[]\rightarrow K^{\downarrow d}(\hat{e}(p, q)\hat{}(a * b))$ such that the send-nodes of the first and second premises are labeled with ru_1 and ru_2 and $fsyms(ru_2) <_{fs} fsyms(ru_1)$ where $<_{fs}$ is a total order on sequences of fact symbols.
N12. There is no chain of nodes repeatedly instantiating a rule of the form $\mathsf{K}^{\downarrow}(l|_p), \mathsf{K}^{\uparrow}(t_1), \ldots, \mathsf{K}^{\uparrow}(t_i) -[]\rightarrowtail \mathsf{K}^{\downarrow}(r)$ of length at least equal to the number of subterms of $l|_p$, if $l|_p$ and r are unifiable.
N13. There is no chain of repeated instantiations of the deconstruction rules for XOR.

Figure 4.7.: Full list of pre-existing normal form conditions (see, e. g., [50])

$p'_{1.2} = [1]$ Here, $l|_{p'_{1.2}} = \mathtt{sign}(\mathtt{blind}(m, r), k)$ and the only sibling positions is $P''_{1.2} = \{[2]\}$, corresponding to r.

$p'_2 = [1]$ $l|_{p'_2} = \mathtt{sign}(\mathtt{blind}(m, r), k)$. Identical to the previous case, thus producing the same deconstruction rule.

$$\frac{\mathsf{K}^{\downarrow}(\mathtt{sign}(\mathtt{blind}(m, r), k)) \quad \mathsf{K}^{\uparrow}(r)}{\mathsf{K}^{\downarrow}(\mathtt{sign}(m, k))}$$

Similarly, for the rewriting rule $\mathtt{unblind}(\mathtt{blind}(m, r), r) \rightarrow m$, we generate $\dfrac{\mathsf{K}^{\downarrow}(\mathtt{blind}(m, r)) \quad \mathsf{K}^{\uparrow}(r)}{\mathsf{K}^{\downarrow}(m)}$. Altogether, the set of deconstruction rules for $\mathtt{unblind}$ is:

$$\mathscr{RV}_{\mathtt{unblind}} = \left\{ \begin{array}{c} \dfrac{\mathsf{K}^{\downarrow}(\mathtt{blind}(m, r)) \quad \mathsf{K}^{\uparrow}(r)}{\mathsf{K}^{\downarrow}(m)} \\[2ex] \dfrac{\mathsf{K}^{\downarrow}(\mathtt{blind}(m, r)) \quad \mathsf{K}^{\uparrow}(k) \quad \mathsf{K}^{\uparrow}(r)}{\mathsf{K}^{\downarrow}(\mathtt{sign}(m, k))}, \\[2ex] \dfrac{\mathsf{K}^{\downarrow}(\mathtt{sign}(\mathtt{blind}(m, r), k)) \quad \mathsf{K}^{\uparrow}(r)}{\mathsf{K}^{\downarrow}(\mathtt{sign}(m, k))} \end{array} \right\}$$

4.2.3. Internal representation: Normal Dependency Graphs

In addition to the normal deduction rules $ND_{\mathscr{E}}$, normal dependency graphs must satisfy 13 additional **normal form conditions**, labeled **N1** to

N13, as shown in Figure 4.7. These conditions impose further restrictions to prevent loops and to improve efficiency by avoiding redundancies. We briefly explain a few of them:

- **N3.** Prevents the adversary from deducing the same term multiple times.
- **N5.** Ensures that function applications such as pairing must always be completely deconstructed before they can be coerced.
- **N2. N6.** Defines specialized conditions targeting the built-in equational theory of Diffie-Hellman exponentiation.
- **N13.** Specifies specialized conditions targeting the built-in equational theory of XOR, an associative and commutative symbol.
- **N12.** Bounds the length of chained deconstruction rules for non-AC symbols.

Conditions **N12** and **N13** will be of particular relevance to the work presented in Chapter 5.

Formally, a *normal dependency graph* for a set of protocol rules P is a dependency graph dg such that $dg \in dgraphs_{\mathcal{AC}}(\lceil P \rceil^{\mathcal{C}} \cup ND_{\mathcal{C}})$ and respects all normal form conditions. Normal dependency graphs allow the same executions as multiset rewriting semantics (see Theorem 5.2.1).

Definition 4.2.3 *The known messages of a dependency graph dg are defined as $\mathsf{known}(dg) = \{m \mid \text{exists conclusion } \mathsf{K}(m) \in dg\}$. The construction-, deconstruction- and combined-known messages for normal dependency graphs ndg are defined as:*

$$\mathsf{known}^{\uparrow}(ndg) = \{m \mid \text{exists conclusion } \mathsf{K}^{\uparrow}(m) \in ndg\}$$
$$\mathsf{known}^{\downarrow}(ndg) = \{m \mid \text{exists conclusion } \mathsf{K}^{\downarrow}(m) \in ndg\}$$
$$\mathsf{known}^{\updownarrow}(ndg) = \mathsf{known}^{\uparrow}(ndg) \cup \mathsf{known}^{\downarrow}(ndg)$$

The notation $\mathtt{stfacts}(dg)$ defines all state conclusion facts of a normal dependency graph except for adversary knowledge facts $\mathsf{K}, \mathsf{K}^{\uparrow}, \mathsf{K}^{\downarrow}$:

$$\mathtt{stfacts}(dg) = \{\mathsf{F}(m) \mid \text{ exists conclusion } \mathsf{F}(m) \in dg, \mathsf{F} \notin \{\mathsf{K}, \mathsf{K}^{\uparrow}, \mathsf{K}^{\downarrow}\}\}$$

The created messages $\mathtt{created}(dg)$ are all the fresh values that appear under Fr facts in a normal dependency graph.

A normal dependency graph $ndg' = (I', D')$ is a *deduction extension* of $ndg = (I, D)$ if I is a prefix of I', $D \subseteq D'$, $trace(ndg) = trace(ndg')$, $\mathtt{stfacts}(ndg) = \mathtt{stfacts}(ndg')$, and $\mathtt{created}(ndg) = \mathtt{created}(ndg')$.

4.2.4. Automated reasoning

To verify that a protocol P satisfies a security property ϕ, formally $P \models \phi$, TAMARIN follows a proof-by-contradiction approach, i.e., it searches the dependency graph for a trace satisfying $\neg\phi$. Since this problem is undecidable in general, the search may not terminate, and the tool might run out of time or memory before reaching a conclusion. When it does reach a conclusion, TAMARIN produces one of two possible outcomes:

- It discovers a trace satisfying $\neg\phi$, and returns the corresponding path in the normal dependency graph, enabling reconstruction of the attack.
- It determines that all constraints required to construct a counter-example are contradictory. This implies that no trace satisfies $\neg\phi$, and hence $P \models \phi$.

Guarded trace properties

To enable proof by contradiction, TAMARIN employs *guarded trace property*, which are derived from $\neg\phi$.

Definition 4.2.4 ([18]) *A trace formula φ is in* negation normal form *if negation is only applied to trace atoms and all other logical connectives are $\wedge, \vee, \forall$ and $\exists$.*
A trace formula φ is a guarded trace formula *if it is in negation normal form and all its quantifiers are of the form $\exists\vec{x}.(F@i) \wedge \psi$ or $\forall\vec{x}.\neg(F@i) \vee \psi$, for an action $F@i$, a guarded trace formula ψ and $x \in vars(F@i) \cap (\mathcal{V}_{msg} \cup \mathcal{V}_{temp})$ for every $x \in \vec{x}$.*
A guarded trace formula φ is a ***guarded trace property*** *if it is closed and $t \in \mathcal{V} \cup \mathcal{N}_{pub}$ holds for all terms t occurring in φ.*

Example 4.2.3 Consider the lemma `alive` presented in Example 2.1.6.

$$\phi = \forall x, y, \#i.\ \mathsf{Alive}(x, y)@\#i \Rightarrow \exists \#j.\ \mathsf{Response}(y)@\#j$$

To construct a guarded trace property, we rewrite the negation $\neg\phi$ as follows:

$$\neg\phi = \exists x, y, \#i.\ \mathsf{Alive}(x, y)@\#i \wedge \forall \#j.\ \neg(\mathsf{Response}(y)@\#j)$$

By using this definition and Theorem 5.2.1, stating that normal dependency graphs allow the same executions as multiset rewriting semantics, we can verify properties in a dependency graph semantics modulo $\mathcal{AC}$ rather than it in the multiset rewriting semantics modulo the full equational theory.

Lemma 4.2.3 ([18], Theorem 1) *Let $\mathcal{E}$ be an equational theory. For every protocol P and every guarded trace property φ,*

$$P \models_{\mathcal{E}} \varphi \text{ iff } \{trace(dg) | dg \in ndgraphs(P)\} \models_{\mathcal{AC}} \varphi$$

Constraint-solving algorithm

Based on the above notions, TAMARIN attempts to prove $P \models \varphi$, with φ a guarded trace property negating the property we want to prove. It does so by searching for a solution to a corresponding *constraint system*.
In this section, we use ri to range over (multiset rewriting) rule instances, u, v over natural numbers, and φ over guarded trace formulas.

Definition 4.2.5 ([18]) *Let $dg = (I, D)$ be a normal dependency graph for a protocol P. Its **deconstruction chain relation** $\dashrightarrow_{dg}$ dg is the smallest relation such that $c \dashrightarrow_{dg} dgp$ if c is a $K^{\downarrow}$-conclusion in dg and*

(a) $c \rightarrowtail p \in D$ or
(b) there is a premise (j, u) such that $c \rightarrowtail (j, u) \in D$ and $(j, 1) \dashrightarrow_{dg} dgp$.

Definition 4.2.6 ([18], simplified) *A **graph constraint** is either a node $i : ri$, an edge $(i, u) \rightarrowtail (j, v)$ or a deconstruction chain $(i, u) \dashrightarrow (j, v)$.*
*A **constraint** is a graph constraint or a guarded trace formula.*

*A **structure** is a tuple (dg, θ) of a dependency graph $dg = (I, D)$ and a valuation θ. We define that a structure (dg, θ) satisfies a constraint γ, written $(dg, \theta) \Vdash \gamma$, if it satisfies:*

$$
\begin{array}{ll}
(dg, \theta) \Vdash i : ri & \text{iff } \theta(i) \in idx(I) \text{ and } ri\theta =_{\mathcal{AC}} I_{\theta(i)} \\
(dg, \theta) \Vdash (i, u) \rightarrowtail (j, v) & \text{iff } (\theta(i), u) \rightarrowtail (\theta(j), v) \in D \\
(dg, \theta) \Vdash (i, u) \dashrightarrow (j, v) & \text{iff } (\theta(i), u) \dashrightarrow_{dg} dg(\theta(j), v) \\
(dg, \theta) \Vdash \varphi & \text{iff } (trace(dg), \theta) \models_{\mathcal{AC}} \varphi
\end{array}
$$

*A **constraint system** Γ is a finite set of constraints. The structure (dg, θ) satisfies Γ, written $(dg, \theta) \Vdash \Gamma$, if (dg, θ) satisfies all constraints in Γ.*

A structure (dg, θ) is a *P-model* of Γ, if dg is a normal dependency graph for P and $(dg, \theta) \Vdash \Gamma$. A *P-solution* of Γ is a normal dependency graph dg for P such that there is a valuation θ with $(dg, \theta) \Vdash \Gamma$.

Tamarin attempts to find a trace verifying $\neg\phi$ by constructing a P-solution for the constraint system $\{\varphi\}$, where φ is $\neg\phi$ rewritten in negation normal form. To refine this constraint system, Tamarin applies a *constraint-reduction relation* $\rightsquigarrow_P$ between constraint systems and sets of constraint systems[8].

8: Used to represent case distinctions

This relation $\rightsquigarrow_P$ is defined using two types of constraint-reduction rules:

- **Simplification rules**, which remove contradictory constraint systems or refine constraint system by simplifying constraints.
- **Case distinction rules**, which add additional constraints to refine the constraint systems, branching into one constraint system per case. They are written $\Gamma \rightsquigarrow_P \Gamma_1 \parallel \cdots \parallel \Gamma_n$ for $\Gamma \rightsquigarrow_P \{\Gamma_1, \ldots, \Gamma_n\}$, which denotes an n-fold case distinction.

These rules are sound and complete, as stated in Lemma 4.2.4, allowing Tamarin to extract a P-solution from all solved constraint system, i.e., those irreducible w.r.t. $\rightsquigarrow_P$.

To define constraint-relation rules, we extend the equality $\approx$ over terms to facts and rule instances by interpreting the constructors for facts and rule instances as free function symbols. A premise of the dependency graph is called *open* if it has no incoming edges or deconstruction chains in Γ, i.e., no rule has been applied to refine it. A term m is *trivial* if $m \in \mathcal{V}_{msg} \cup \mathcal{V}_{pub} \cup \mathcal{N}_{pub}$. We define the input component of a term t as $inp(t)$, i.e., the maximal subterms of t. We refer to $K^{\uparrow}$- and $K^{\downarrow}$-premises

Remark 4.2.1 The full set of constraint-reduction rules is provided in [18, Figure 10]. We will only present the most relevant examples in this thesis.

as *message deduction constraints*. We define the *temporal order* of Γ as:

$$(\lessdot_\Gamma) = \{(i,j) \mid (i \lessdot j) \in \Gamma \vee \exists u, v.\ ((i,u) \rightarrowtail (j,v)) \in \Gamma \vee ((i,u) \dashrightarrow (j,v)) \in \Gamma\}^+$$

Moreover, we define the actions $as(\Gamma)$, premises $ps(\Gamma)$, and conclusions $cs(\Gamma)$ of a constraint system Γ as:

$$\begin{aligned} as(\Gamma) &= \{\mathsf{f}@i \mid \exists\, \mathsf{r}\ \mathsf{a}.(i : \mathsf{l} -\!\![\ \mathsf{a}\ ]\!\!\rightarrow \mathsf{r}) \in \Gamma \wedge \mathsf{f} \in \mathsf{a}\} \\ ps(\Gamma) &= \{((i,u), \mathsf{l}_u) \mid \exists\, \mathsf{r}\ \mathsf{a}.(i : \mathsf{l} -\!\![\ \mathsf{a}\ ]\!\!\rightarrow \mathsf{r}) \in \Gamma \wedge u \in idx(\mathsf{l})\} \\ cs(\Gamma) &= \{((i,v), \mathsf{r}_v) \mid \exists\, \mathsf{l}\ \mathsf{a}.(i : \mathsf{l} -\!\![\ \mathsf{a}\ ]\!\!\rightarrow \mathsf{r}) \in \Gamma \wedge v \in idx(\mathsf{r})\} \end{aligned}$$

$\mathbf{s}_\approx$:	$\Gamma \rightsquigarrow_P \|_{\sigma \in unif_{\mathcal{AC}}(t_1 =^? t_2)}\ (\Gamma\sigma)$	if $t_1 \approx t_2 \in \Gamma \wedge t_1 \neq_{\mathcal{AC}} t_2$
$\mathbf{DG1_2}$:	$\Gamma \rightsquigarrow_P (\mathsf{f} \approx \mathsf{f}', \Gamma)$	if $c \rightarrowtail p \in \Gamma, (c, \mathsf{f}) \in cs(\Gamma), (p, \mathsf{f}') \in ps(\Gamma)$, and $\mathsf{f} \neq_{\mathcal{AC}} \mathsf{f}'$
$\mathbf{N_1}$:	$\Gamma \rightsquigarrow_P \bot$	if $(i : ri) \in \Gamma \wedge ri$ not in normal form
$\mathbf{N_{13}}$:	$\Gamma \rightsquigarrow_P \bot$	if $(i : ri), (j : ri'), (i,1) \dashrightarrow (j,1)$, ri and ri' are instances of XOR deconstruction rules
$\mathbf{DG2_{2,\uparrow e}}$:	$\Gamma \rightsquigarrow_P \|_{ri \in ND_{\mathcal{C}}}\ (i : ri; (i,1) \rightarrowtail p, \Gamma)$	if p is an open $\mathsf{K}^\uparrow(m)$-premise in Γ, $\{m\} = inp(m)$, m non-trivial and i fresh

Figure 4.8.: Selected rules for the constraint-reduction relation $\rightsquigarrow_P$ ([18], extract). Rule $\mathbf{N_{13}}$ is introduced in [50].

Figure 4.8 shows some examples of constraint-reduction rules:

- $\mathbf{s}_\approx$ Trace formula reduction rule. For each unifier σ of t_1 and t_2, this rule produces a refined constraint system where σ is applied homomorphically.
- $\mathbf{DG1_2}$ Graph constraint reduction rule. This rule adds an equality constraint ($\approx$) between two facts connected by an edge, ensuring, along with $\mathbf{s}_\approx$ rule, the equality of all such connected facts throughout the system.
- $\mathbf{N_1}$ Graph constraint reduction rule. This rule corresponds to the first normal form condition: it ensures all rule instances are in normal form.
- $\mathbf{N_{13}}$ Graph constraint reduction rule. This rule corresponds to the normal form conditions for XOR operator: it prevents a deconstruction chain between two XOR deconstruction rule instances.
- $\mathbf{DG2_{2,\uparrow e}}$ Message deduction constraint reduction rule. This rule solves open $\mathsf{K}^\uparrow$-premises that involve no term of sort *msg* or *pub*, by applying all applicable normal deduction rules.

TAMARIN's constraint-solving procedure is summarized in Figure 4.9. Intuitively, the algorithm applies constraint-reducing rules to the initial set $\{\{\varphi\}\}$ until it either finds a solved constraint system (i.e., an attack is found) or all constraints systems become irreducible (i.e., ϕ holds).

The correctness of this algorithm is ensured by the following two theorems, whose proofs are given in [18, Appendix E].

Lemma 4.2.4 ([18], Theorem 2) *The constraint-reduction relation $\rightsquigarrow_P$ is sound and complete: for every $\Gamma \rightsquigarrow_P \{\Gamma_1, \ldots, \Gamma_n\}$, the set of P-solutions of*

```
1  Function Solve(P ⊨_ℰ φ):
2      φ ← ¬φ rewritten in negation normal form;
3      Ω ← {{φ}};
4      while Ω ≠ ∅ and solved(Ω) = ∅ do
5          choose Γ ⇝_P {Γ_1, ..., Γ_n} such that Γ ∈ Ω ;
6          Ω ← (Ω\{Γ} ∪ {Γ_1, ..., Γ_n};
7      if solved(Ω) ≠ ∅ then
8          return "attack(s) found:", solved(Ω);
9      else
10         return "verification successful";
```

Figure 4.9.: Pseudocode of the constraint-solving algorithm [18].

Γ *is equal to the union of the sets of P-solutions of all* Γ_i*, with* $1 \leqslant i \leqslant n$.

Lemma 4.2.5 ([18], Theorem 3) *We can construct a P-solution from every solved system in the state* Ω *of the constraint-solving algorithm.*

Handling User-Defined AC Operators in Tamarin

5.

Summary

This chapter presents joint work with Jannik Dreier and Steve Kremer on a new feature for Tamarin: user-defined Associative and Commutative (AC) symbols. To address the issue of infinite chains of deconstructors often related to such operators, we introduce the *No Deconstruction Chain* (NDC) property, along with a corresponding normal form condition and constraint-reduction rule. Moreover, to enable Tamarin to handle these new operators, we design an algorithm that checks whether a function symbol satisfies the NDC property, and thus whether deconstruction chains involving it can be bounded. We introduce also another constraint-reduction rule $\mathbf{DG}_{+,\perp}$ to address the issue of the constructor chains, also related to AC operators. We illustrate the impact of our algorithm on existing case-studies and introduce new ones that can now be modeled thanks to this extension.

5.1 Context: AC symbols in Tamarin 69
5.1.1 Related work 70
5.1.2 Contributions 70
5.2 The No Deconstruction Chain property 72
5.3 Verifying the No Deconstruction Chain property 73
5.3.1 Algorithm to check the NDC property 74
5.3.2 Verifying deduction 74
5.4 Pruning constructor chains 79
5.4.1 Constructor chains issue . 79
5.4.2 Extending $\rightsquigarrow_p$ with $\mathbf{DG}_{+,\perp}$ 80
5.5 Evaluation and Case studies 83
5.5.1 Impact of $\mathbf{DG}_{+,\perp}$ 84
5.5.2 Existing AC symbols . . . 85
5.5.3 Improvement of existing case studies 86
5.5.4 Toy voting system based on distributed decryption 87
5.5.5 Exponentiation Mixnet . . 88
5.5.6 Summary 89
5.6 Future Work 90

5.1. Context: AC symbols in Tamarin

In this work we focus on the Tamarin Prover. Tamarin is one of the most feature-rich and mature tools: it supports both verification and falsification, allows the specification of complex state machines, and offers support for many cryptographic primitives that can be specified through equational theories as well as built-in support for several theories which require an AC operator (such as XOR, Diffie-Hellman (DH) exponentiation, or multisets). Moreover, for the most complex case studies, that may lead to non-termination issues, it offers various methods to guide proofs using lemmas, so-called oracles and an interactive user interface. One limitation, however, is that all AC operators are built-in and must not be used in user-defined equational theories. Yet, it would be tempting to use the built-in theory for exponentiation to provide a user-defined theory that accounts for public keys of the form g^s for a private key s as, e.g., in El-Gamal encryption, by the equation

$$\mathsf{dec}(\mathsf{enc}(m, g^s, r), s) = m$$

Similarly, one may want to model re-randomization of encryption using an AC operator +, such as the built-in operator for constructing multisets, by the equation

$$\mathsf{reenc}(\mathsf{enc}(m, pk, r_1), r_2, pk) = \mathsf{enc}(m, pk, r_1 + r_2)$$

However, as stated above, using symbols from built-in theories in user-defined theories is forbidden as this might break the correctness of hard-coded optimizations.

5.1.1. Related work

The support of AC symbols in automated verification tools is a challenging problem that has been studied across a wide range of tools. Most previous work on this topic has focused on specific, dedicated equational theories that include AC symbols, rather than on providing general support for user-defined theories involving AC symbols. In this context, Maude-NPA [51] was the only tool allowing user-defined theories including AC symbols, as it directly inherits Maude's handling of equational theories. Tamarin relies on Maude for variant computation and unification modulo $\mathscr{AC}$, and therefore has a generic infrastructure capable of handling arbitrary AC symbols. However, this level of generality often comes at the cost of performance and may lead to termination issues.

[51]: Escobar et al. (2007), *Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties*

In the case of a bounded number of sessions, several results have been obtained, notably for:

- XOR: In [52], Comon-Lundh *et al.* show that constraint systems for an intruder theory extended with the possibility to exploit XOR's properties is decidable. In [53], Chevalier *et al.* present an NP decision procedure for protocol security against a similar intruder theory, based on Oracle rules that must be proved manually for each equational theory.
- Diffie-Hellman exponentiation: Chevalier *et al.* [54] prove that the insecurity problem for protocols using Diffie-Hellman exponentiation is NP-complete and consequently that the corresponding derivation problem is decidable.

[52]: Comon-Lundh et al. (2003), *Intruder Deductions, Constraint Solving and Insecurity Decision in Presence of Exclusive or*

[53]: Chevalier et al. (2005), *An NP decision procedure for protocol insecurity with XOR*

[54]: Chevalier et al. (2008), *Complexity results for security protocols with Diffie-Hellman exponentiation and commuting public key encryption*

In the case of an unbounded number of sessions, there has been work on XOR and Diffie-Hellman exponentiation in ProVerif [55, 56]. These approaches, however, operate directly at the level of Horn clauses and are limited to a certain subclass of protocols.

[55]: Küsters et al. (2009), *Using ProVerif to Analyze Protocols with Diffie-Hellman Exponentiation*

[56]: Küsters et al. (2011), *Reducing Protocol Analysis with XOR to the XOR-Free Case in the Horn Theory Based Approach*

Tamarin allows user-defined theories with a number of built-ins, including XOR, Diffie-Hellman exponentiation, bilinear pairing and multisets operator. Similarly, AKiSs verifies equivalence properties for subterm convergent theories combined with a built-in XOR [57]. In Tamarin, the built-in equational theory for XOR is implemented using dedicated optimizations, designed and proved to avoid termination issues [50].

[57]: Baelde et al. (2017), *Symbolic Verification of Privacy-Type Properties for Security Protocols with XOR*

[50]: Dreier et al. (2020), *Verification of Stateful Cryptographic Protocols with Exclusive OR*

5.1.2. Contributions

In this work we enrich Tamarin with user-defined AC symbols that can be used in any equational theory, as long as it can be oriented in an AC-convergent rewrite system with the finite variant property (Definition 4.1.2). Integrating these theories in a naive way is straightforward, as they are directly supported by Maude [58] already used for handling the built-in theories. Moreover, the algorithm presented in the previous chapter for generating the corresponding deconstruction rules (Figure 4.6) is directly applicable to these kinds of operators. However, a naive integration also leads to immediate non-termination when reasoning about attacker deductions.

[58]: Clavel et al. (2007), *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*

To overcome these non-termination issues we introduce a novel property, the *No Deconstruction Chain* (NDC) property. Informally, the property

states that if a function symbol `f` satisfies NDC then it is not necessary to chain two consecutive deconstruction rules of `f`. The corresponding normal form condition then states that any normal dependency graph[1] does not contain such chains of deconstruction rules, avoiding these non-termination issues in message deduction. We also define and implement a sound test to check whether a symbol satisfies NDC. This test generalizes ideas from previous work on the XOR operator [50] and exploits TAMARIN's internal deduction algorithm to find alternative normal deductions for all non-normal cases. In addition to enabling user-defined AC symbols, we also observed improved performance for some other existing theories, such as trap-door commitments, that happen to satisfy the NDC property.

1: TAMARIN's internal representation of protocol executions, Section 4.2.3

While developing support for user-defined AC theories, we identified an optimization that restricted the instantiations of construction rules for AC operators such as the built-in XOR operator and that could lead to incorrect results[2]. Simply removing this optimization recovers soundness but significantly increases the number of non-terminating cases. We thus introduce a new constraint-reduction rule that removes redundant constraint systems to recover termination for most of the examples, and provide its soundness proof. This proof requires a novel notion of *flattened* normal dependency graphs as well as more global reasoning. Specifically, while each previous constraint-reduction rule preserved the solutions of the constraint system it applied to, we now reason about the set of constraint systems, show that all solutions of removed constraint systems are also solutions of other preserved constraint systems.

2: A counter-example to the correctness of the initial version of our novel support for pure AC operators was discovered by *Stéphanie Delaune*. When analyzing the issue, we discovered the optimization and were able to adapt the example to demonstrate the existence of this bug even with the built-in XOR.

To validate our approach, we conducted a first sanity check by comparing user-defined theories for XOR and multisets with their built-in counterparts. We confirmed that both generate the same internal rules and exhibit the same behavior on most existing examples, available on TAMARIN repository (more details in Section 5.5). Moreover, in all cases where termination was achieved, we confirmed that the previous results using XOR were indeed correct, and not altered due to the bug. The built-in theories however benefit from some tailored, hard-coded optimizations resulting in better performance on some examples. Similarly, the built-in theory for DH exponentiation has been optimized by restricting the set of allowed protocols; the user-defined version does not have this limitation but in exchange produces too many variants and becomes too inefficient for practical use. This is expected and initially motivated the protocol restriction for the built-in DH theory.

More interestingly, we are able to model several other examples of equational theories that require an AC symbol and were previously out of scope. We notably model theories for re-encryption, a theory for exponentiation that is simpler than the built-in one but sufficient to model a re-encryption mixnet, as well as a theory for distributed decryption which we use to model a toy-voting protocol.

Outline We introduce the *No Deconstruction Chain* property and a corresponding new normal form condition in Section 5.2. We present an algorithm to verify the NDC property, its optimizations and the correctness proof in Section 5.3. Then we discuss the issue of the constructor chains and our fix in Section 5.4. Finally, we detail our case studies to evaluate

our contributions in Section 5.5, before presenting our future work in Section 5.6.

5.2. The No Deconstruction Chain property

The existing normal deduction rules and normal form conditions are unfortunately often insufficient when handling equational theories with AC symbols.

We illustrate this on the example of the $\oplus$ operator. When TAMARIN requires a fact $\mathsf{K}^{\downarrow}(a \oplus b)$ this fact may have been produced by $R_{\oplus.2}$ (Figure 4.5), requiring premises $\mathsf{K}^{\downarrow}(a \oplus c)$ and $\mathsf{K}^{\uparrow}(c \oplus b)$. Hence, TAMARIN requires the fact $\mathsf{K}^{\downarrow}(a \oplus c)$. This fact, in turn, can again be produced by $R_{\oplus.2}$, leading to an infinite chain, as illustrated opposite.

$$\dfrac{\dfrac{\dfrac{\dots \quad \dots}{\mathsf{K}^{\downarrow}(a \oplus d)}}{\mathsf{K}^{\downarrow}(a \oplus d) \quad \mathsf{K}^{\uparrow}(d \oplus c)}}{\mathsf{K}^{\downarrow}(a \oplus c)}}{\mathsf{K}^{\downarrow}(a \oplus c) \quad \mathsf{K}^{\uparrow}(c \oplus b)}}{\mathsf{K}^{\downarrow}(a \oplus b)}$$

The normal form condition **N12** (see Figure 4.7) forbids this kind of behavior for non-AC symbols. **N12** is a syntactical criterion that bounds the number of consecutive instantiation of deconstruction rules of the same symbol based on the number of subterms. For built-in $\oplus$ operator this is avoided using the dedicated normal form condition **N13** which forbids two consecutive applications of $\oplus$ deconstruction rules.

Our goal is to define a more general normal form condition covering also user-defined *AC* symbols, similar to **N13**. Disallowing consecutive applications of deconstruction rules is however not sound in general. We therefore define a property which, when satisfied, ensures its soundness for a given function symbol.

Definition 5.2.1 (No Deconstruction Chain property (NDC)) *Let $\mathcal{E}$ be an equational theory and $\mathtt{f}/k \in \Sigma_{Op}$. $\mathtt{f}$ has the NDC property w.r.t. to $\mathcal{E}$ if, for any dependency graph ndg that*

1. *is normal w.r.t. **N1-N13** and*
2. *ends with two rule instances r_a, r_b of rules in $\mathcal{RV}_{\mathtt{f}}$*[3]

$$r_b\dfrac{r_a\dfrac{K^{\downarrow}(t) \quad K^{\uparrow}(t_1) \quad \dots \quad K^{\uparrow}(t_m)}{K^{\downarrow}(u)} \qquad K^{\uparrow}(u_1) \quad \cdots \quad K^{\uparrow}(u_l)}{K^{\downarrow}(v)}$$

3. *and does not otherwise contain two chained deconstruction rules in $\mathcal{RV}_{\mathtt{f}}$,*

then there exists a deduction extension de of ndg without the rules r_a and r_b, such that $v \in \mathsf{known}^{\updownarrow}(de)$ and de contains no chain of repeated instantiations of deconstruction rules for $\mathtt{f}$.

The NDC property basically states that whenever we can chain two deconstructor rules of the same symbol, then we can remove the chain, and find an alternative derivation that only uses single instances instead.

Example 5.2.1 As we will see the $\oplus$ operator has the NDC property.

3: Here and in the following we use a more compact representation of deduction steps where we do not show the edges but instead merge connected conclusions and premises.

Indeed, the chain

$$\dfrac{\dfrac{\mathsf{K}^{\downarrow}(a \oplus d) \quad \mathsf{K}^{\uparrow}(d \oplus c)}{\mathsf{K}^{\downarrow}(a \oplus c)} \qquad \mathsf{K}^{\uparrow}(c \oplus b)}{\mathsf{K}^{\downarrow}(a \oplus b)}$$

can be replaced by

$$\dfrac{\mathsf{K}^{\downarrow}(a \oplus d) \quad \mathsf{K}^{\uparrow}(d \oplus b)}{\mathsf{K}^{\downarrow}(a \oplus b)}$$

plus some additional deduction steps to obtain $\mathsf{K}^{\uparrow}(d \oplus b)$, which however do not violate NDC.

Hence, we can define the following normal form condition that applies on all function symbols that satisfy NDC.

Definition 5.2.2 ***N14.*** *There is no chain of repeated instantiations of deconstruction rules for a function symbol with the NDC property.*

The correctness of our approach is given by the following theorem, stating that the observable traces of the protocol given the semantics are equal to the observable traces of all normal dependency graphs, i.e., graphs respecting all normal form conditions including **N14**.

Theorem 5.2.1 (From [50], Theorem 1, but now including **N14.**) *For all sets P of protocol rules,*

$$\overline{trace(execs(P \cup MD))} \downarrow_{\mathscr{E}'=_{\mathscr{AC}}} \overline{trace(ndgraphs(P))}.$$

The proof is given in Appendix C.1.

5.3. Verifying the No Deconstruction Chain property

$$\mathsf{Check\_NDC}_{\mathscr{E}}(\mathsf{f}) \triangleq \forall\, r_a, r_b \in \mathscr{RV}_{\mathsf{f}}, \sigma \in unif_{\mathscr{AC}}(u^a =^? t^b).\textsc{Ded}_{\mathsf{f},\mathscr{E}}(\{u^b\sigma\}, U, D)$$

where

- $r_{\mathsf{x}} = \dfrac{\mathsf{K}^{\downarrow}(t^{\mathsf{x}}) \quad \mathsf{K}^{\uparrow}(t_1^{\mathsf{x}}) \quad \ldots \quad \mathsf{K}^{\uparrow}(t_{n_{\mathsf{x}}}^{\mathsf{x}})}{\mathsf{K}^{\downarrow}(u^{\mathsf{x}})}$ for $\mathsf{x} \in \{a, b\}$
- $U = \{t_1^a\sigma, \ldots, t_{n_a}^a\sigma, t_1^b\sigma, \ldots, t_{n_b}^b\sigma\}$
- $D = \{t^a\sigma\}$

and $\textsc{Ded}_{\mathsf{f},\mathscr{E}}(T, U, D)$ verifies whether all facts $\mathsf{K}^{\uparrow}(t)$ for $t \in T$ can be derived from facts $\mathsf{K}^{\uparrow}(u)$ for $u \in U$ and facts $\mathsf{K}^{\downarrow}(d)$ for $d \in D$ w.r.t. the equational theory $\mathscr{E}$ (see Section 5.3.2) without chaining deconstruction rules for f.

Figure 5.1.: Algorithm to check whether a function symbol f as the NDC property w.r.t. the equational theory $\mathscr{E}$.

5.3.1. Algorithm to check the NDC property

To check whether a function symbol f satisfies the NDC property we apply the test $\mathsf{Check\_NDC}_{\mathscr{C}}(\mathtt{f})$ defined in Figure 5.1. In summary, we check that for any two deconstruction rules for f (denoted r_a, r_b) that can be chained, there exists another way of deriving the same value while respecting the normal form condition **N14**. We now explain the steps of the algorithm in more details and illustrate them on the example of the $\oplus$ operator.

Chaining rules

To check whether two deconstructor variants of f can be chained we check unification of the conclusion of r_a and the first premise of r_b, i.e., $\sigma \in unif_{\mathscr{AC}}(u^a =^? t^b)$:

$$\dfrac{\dfrac{\mathsf{K}^{\downarrow}(t^a\sigma)\ \mathsf{K}^{\uparrow}(t^a_1\sigma)\ \cdots\ \mathsf{K}^{\uparrow}(t^a_{n_a}\sigma)}{\mathsf{K}^{\downarrow}(u^a\sigma =_{\mathscr{AC}} t^b\sigma)} \qquad \mathsf{K}^{\uparrow}(t^b_1\sigma)\ \cdots\ \mathsf{K}^{\uparrow}(t^b_{n_b}\sigma)}{\mathsf{K}^{\downarrow}(u^b\sigma)}$$

Remark 5.3.1 When choosing $r_a, r_b \in \mathscr{RV}_{\mathtt{f}}$ we tacitly assume that variables in r_a and r_b are fresh and distinct, to avoid any name clashes during unification. This holds in particular, when r_a and r_b are (fresh renamings of) the same rule.

Example 5.3.1 We illustrate our algorithm on the $\oplus$ operator, for which $\mathscr{RV}_{\oplus}$ contains

$$r_{\oplus.1}\dfrac{\mathsf{K}^{\downarrow}(a \oplus b) \quad \mathsf{K}^{\uparrow}(a \oplus c)}{\mathsf{K}^{\downarrow}(b \oplus c)}, r_{\oplus.2}\dfrac{\mathsf{K}^{\downarrow}(d \oplus e) \quad \mathsf{K}^{\uparrow}(d)}{\mathsf{K}^{\downarrow}(e)}$$

To check whether $r_{\oplus.1}$ and $r_{\oplus.2}$ can be chained we compute $unif_{\mathscr{AC}}(b \oplus c =^? d \oplus e)$ which contains 7 unifiers among which

$$\sigma_1 = \{a \mapsto x_1, b \mapsto x_2, c \mapsto x_3, d \mapsto x_3, e \mapsto x_2\}$$
$$\sigma_2 = \{a \mapsto x_1, b \mapsto x_3, c \mapsto x_2, d \mapsto x_3, e \mapsto x_2\}$$

(We do not detail the cases for the other unifiers.) Applying σ_1 we can chain $r_{\oplus.1}$ and $r_{\oplus.2}$ as follows:

$$\dfrac{\dfrac{\mathsf{K}^{\downarrow}(x_1 \oplus x_2) \quad \mathsf{K}^{\uparrow}(x_1 \oplus x_3)}{\mathsf{K}^{\downarrow}(x_3 \oplus x_2)} \qquad \mathsf{K}^{\uparrow}(x_3)}{\mathsf{K}^{\downarrow}(x_2)}$$

5.3.2. Verifying deduction

Our goal is to replace the two chained rules in the previous section by a different deduction that also leads to the same term $u^b\sigma$, but that satisfies the normal form condition **N14**. The goal is to encode this problem directly in Tamarin and leverage Tamarin's deduction capabilities.

Given a set of facts $D_F = \{\mathsf{K}^{\downarrow}(d) | d \in D = \{d_1, \ldots d_n\}\}$ and a set of facts $T_F = \{\mathsf{K}^{\uparrow}(t) | t \in T = \{t_1, \ldots t_n\}\}$ it is indeed straightforward to write a

$$\textsc{Ded}_{\mathtt{f},\mathscr{E}}(T,U,D) = \begin{cases} \textsc{Ded}^{\mathtt{f},\mathscr{E}}_{tam}(T,D) & \text{if } U = \emptyset \\ \textsc{Ded}_{\mathtt{f},\mathscr{E}}(T, U \setminus \{t\}, D \cup \{t\}) \wedge & \\ \bigwedge_{t \in U} \left(t = \mathtt{f}(t'_1, \ldots t'_k) \wedge \mathtt{f}/k \in \Sigma_{pub}\right) \rightarrow \textsc{Ded}_{\mathtt{f},\mathscr{E}}(T, U \setminus \{t\} \cup \{t'_1, \ldots t'_k\}, D) & \text{otherwise} \end{cases}$$

Figure 5.2.: Algorithm for checking deduction.

one-rule Tamarin encoding and a lemma to check whether facts in T_F can be deduced from facts in D_F as follows. Consider the rule Deduction:

```
[Fr(x_1),...,Fr(x_k)] --[Gen(x_1,...,x_k)]-> [Out(d_1),...,Out(d_n)]
```

where $\{x_1, \ldots, x_k\} = vars(D)$ and the lemma

```
Ex x_1 ... x_k #i #j_1 ... #j_n. Gen(x_1,...,x_k)@#i & K(t_1)@#j_1 & ... & K(t_n)@#j_n
```

This encoding simply outputs the terms in D and checks whether the attacker can know terms in T. As rules cannot contain free variables we simply replace variables by fresh names; the `Gen` action is used to bind these names between the rule and the lemma. Given that `Out`(d) facts are translated to $\mathsf{K}^{\downarrow}(d)$ facts by the Irecv rule and `K`(t) requires a fact $\mathsf{K}^{\uparrow}(t)$ to hold this encoding is correct. Moreover, for this check, we forbid chaining of deconstruction rules for $\mathtt{f}$ as we want to show that a deduction respecting **N14** exists. We denote this check by $\textsc{Ded}^{\mathtt{f},\mathscr{E}}_{tam}(T,D)$ [4].

4: The check is always related to a function symbol $\mathtt{f}$ and an equational theory $\mathscr{E}$

However, the above check does not directly fit our needs, because we need to check whether terms in T can be deduced from not only $\mathsf{K}^{\downarrow}$ facts, but also from $\mathsf{K}^{\uparrow}$ facts. We therefore perform a case analysis on how the $\mathsf{K}^{\uparrow}$ facts could have been obtained and use Tamarin to perform the check in each of the possible cases. We define the deduction check in Figure 5.2.

Intuitively, Ded holds whenever facts $\mathsf{K}^{\uparrow}(t)$ for $t \in T$ can be obtained from facts $\mathsf{K}^{\uparrow}(u)$ for $u \in U$ and $\mathsf{K}^{\downarrow}(d)$ for $d \in D$. When all hypothesis are of the form $\mathsf{K}^{\downarrow}(\cdot)$ this problem can be straightforwardly checked by Tamarin using $\textsc{Ded}^{\mathtt{f},\mathscr{E}}_{tam}(T,D)$. Otherwise we perform a case analysis on each of the $\mathsf{K}^{\uparrow}(\cdot)$ facts and consider all possible cases to produce such a fact using a message deduction rule (see Figure 4.5):

- $\mathsf{K}^{\uparrow}(t)$ was obtained using the Coerce rule $\dfrac{\mathsf{K}^{\downarrow}(t)}{\mathsf{K}^{\uparrow}(t)}$;
- $t = \mathtt{f}(t_1, \ldots, t_k)$ was obtained by applying $\mathtt{f}/k \in \Sigma_{Op}$ using the construction rule $\dfrac{\mathsf{K}^{\uparrow}(t_1) \quad \cdots \quad \mathsf{K}^{\uparrow}(t_k)}{\mathsf{K}^{\uparrow}(t)}$.

One may note that we omit two cases: t is fresh or public and has been obtained by $\dfrac{}{\mathsf{K}^{\uparrow}(t : pub)}$ or $\dfrac{\mathsf{Fr}(t : fr)}{\mathsf{K}^{\uparrow}(t : fr)}$. However, these two cases are similar to the coerce case, as t cannot be decomposed and $\mathsf{K}^{\uparrow}(t)$ is available.

Lemma 5.3.1 *Let $dg \in ndgraphs(P)$. If $U \subseteq \mathsf{known}^{\uparrow}(dg)$, $\forall d \in D.\ (d \in \mathsf{known}^{\downarrow}(dg) \vee (d : fr \wedge d \in \mathsf{known}^{\uparrow}(dg)) \vee d : pub)$ and $\textsc{Ded}_{\mathtt{f},\mathscr{E}}(T,U,D)$*

holds then there exists a deduction extension de of dg such that $T \subseteq \mathsf{known}^{\uparrow}(de)$ and de respects ***N14****.*

Proof. We proceed by induction on the sum of the sizes of the terms in U.

Base case: $U = \emptyset$. As $\text{Ded}_{\mathsf{f},\mathscr{E}}(T, \emptyset, D)$ holds we have that $\text{Ded}^{\mathsf{f},\mathscr{E}}_{tam}(T, D)$ holds.

This means that Tamarin has constructed a dependency graph de' which contains the rule Deduction, the preceding Fresh rules and some deduction rules such that $T \subseteq \mathsf{known}^{\uparrow}(de')$, respecting the normal form condition **N14** for rules in $\mathscr{RV}_{\mathsf{f}}$. We can remove the Deduction rule, its preceding Fresh rules, the Irecv rules turning $\mathsf{Out}(d)$ facts into $\mathsf{K}^{\downarrow}(d)$ facts for some $d \in D$, and obtain a graph de'' with certain open premises, namely $\mathsf{K}^{\downarrow}(d)$ for some $d \in D$.
We can add de'' to dg to obtain de as $\forall d \in D.\ (d \in \mathsf{known}^{\downarrow}(dg) \vee (d : fr \wedge d \in \mathsf{known}^{\uparrow}(dg)) \vee d : pub)$. In other words, either $\mathsf{K}^{\downarrow}(d)$ exists as a conclusion in dg, or $d : fr \wedge d \in \mathsf{known}^{\uparrow}(dg)$, or $d : pub$:

- In the first case, we can simply connect the open premise to the existing conclusion.
- In the other cases, as $d \notin \mathsf{known}^{\downarrow}(dg)$ and d is atomic, if there is a premise $\mathsf{K}^{\downarrow}(d)$ in de'', it is necessarily the premise of a Coerce rule.
  - If $d : fr$, we know that $d \in \mathsf{known}^{\uparrow}(dg)$, hence we can remove the Coerce rule and directly connect to the $\mathsf{K}^{\uparrow}(d)$ conclusion.
  - If $d : pub$, we can replace the Coerce rule by an $\dfrac{}{\mathsf{K}^{\uparrow}(d : pub)}$ rule.

Inductive case: $U \neq \emptyset$. Take $u \in U$. We proceed by case analysis on the rule producing the conclusion $\mathsf{K}^{\uparrow}(d)$, i.e., showing that $u \in \mathsf{known}^{\uparrow}(dg)$.

- Coerce $\dfrac{\mathsf{K}^{\downarrow}(u)}{\mathsf{K}^{\uparrow}(u)}$. In this case we have that $u \in \mathsf{known}^{\downarrow}(dg)$. As $\text{Ded}_{\mathsf{f},\mathscr{E}}(T, U \setminus \{u\}, D \cup \{u\})$ holds we conclude by induction hypothesis.
- IFresh $\dfrac{u : fr}{\mathsf{K}^{\uparrow}(u : fr)}$. We have that $\text{Ded}_{\mathsf{f},\mathscr{E}}(T, U \setminus \{u\}, D \cup \{u\})$ holds. Moreover $u : fr$ and $u \in \mathsf{known}^{\uparrow}(dg)$. Hence by induction hypothesis we conclude.
- Pub $\dfrac{}{\mathsf{K}^{\uparrow}(u : pub)}$. Identical to the previous case.
- f $\dfrac{\mathsf{K}^{\uparrow}(u_1) \quad \cdots \quad \mathsf{K}^{\uparrow}(u_k)}{\mathsf{K}^{\uparrow}(\mathsf{f}(u_1, \ldots, u_k))}$. If this rule was applied, $u = \mathsf{f}(u_1, \ldots, u_k)$. As $u_1, \ldots, u_k \in \mathsf{known}^{\uparrow}(dg)$, and $\text{Ded}_{\mathsf{f},\mathscr{E}}(T, U', D)$ holds for $U' = U \setminus \{u\} \cup \{u_1, \ldots u_k\}$ and the size of U' is smaller than the size of U, we conclude by induction hypothesis.

□

Correctness proof

We prove the correctness of $\mathsf{Check\_NDC}_{\mathscr{C}}(\mathtt{f})$ by using the result of Lemma 5.3.1:

Lemma 5.3.2 *If $\mathsf{Check\_NDC}_{\mathscr{C}}(\mathtt{f})$ holds, then $\mathtt{f}$ has the NDC property.*

Proof. Let $\mathtt{f}/k \in \Sigma_{Op}$ such that $\mathsf{Check\_NDC}_{\mathscr{C}}(\mathtt{f})$ holds. Suppose we have a dependency graph ndg that is normal except that it ends with two chained instances $r_a\sigma_a$ and $r_b\sigma_b$ of rules $r_a, r_b \in \mathscr{RV}_{\mathtt{f}}$. Using notations from Figure 5.1, this means that $\mathsf{K}^{\downarrow}(u^a\sigma) =_{\mathscr{AC}} \mathsf{K}^{\downarrow}(t^b\sigma)$. Hence there exists $\sigma \in unif_{\mathscr{AC}}(u^a =^? t^b)$ such that σ_a and σ_b are instances of σ. Moreover, we have that $\textsc{Ded}_{\mathtt{f},\mathscr{C}}(\{u^b\sigma\}, U, D)$ holds, hence, by Lemma 5.3.1 there exists a deduction extension de of ndg without r_b such that $u^b\sigma_b \in \mathsf{known}^{\uparrow}(de)$ and which does not contain chained instances of the deconstruction rules of $\mathtt{f}$, which concludes the proof. □

Implementation optimizations

As we have seen $\textsc{Ded}_{\mathtt{f},\mathscr{C}}(T, U, D)$ requires multiple calls to Tamarin. To improve the efficiency of the deduction phase, we apply some optimizations at the implementation level.

Naive approximation of the deduction problem In our implementation, each time we call $\textsc{Ded}_{\mathtt{f},\mathscr{C}}(T, U, D)$, described in Figure 5.2, we perform a naive, but very efficient under-approximation of the deducibility check on each term $t \in T$, using $\textsc{DedNaive}_{\mathtt{f},\mathscr{C}}(t, S) =$

$$\begin{cases} \mathsf{true} & \text{if } t \in S \\ \bigwedge_{t' \in \{t_1,\ldots,t_k\}} \textsc{DedNaive}_{\mathtt{f},\mathscr{C}}(t', S) & \text{if } \begin{pmatrix} t \notin S \wedge \\ t = \mathtt{f}(t'_1, \ldots, t'_k) \wedge \\ \mathtt{f}/k \in \Sigma_{pub} \end{pmatrix} \\ \mathsf{false} & \text{otherwise} \end{cases}$$

This check holds when t can be obtained by simply applying public function symbols on known terms in S. We observe that, in practice, this check is often sufficient and significantly improves performance (with only a negligible overhead when it fails). At each (possibly recursive) call on $\textsc{Ded}_{\mathtt{f},\mathscr{C}}(T, U, D)$ we only call Ded on terms that cannot be deduced with the naive check, i.e., we call $\textsc{Ded}_{\mathtt{f},\mathscr{C}}(T', U, D)$ where $T' = \{t \mid t \in T \wedge \neg\textsc{DedNaive}_{\mathtt{f},\mathscr{C}}(t, U \cup D)\}$. Note that this naive check is trivially sound as it simply corresponds to applying the constructor rule for a function symbol $\mathtt{f}$, after possibly coercing a $\mathsf{K}^{\downarrow}$ fact.

Example 5.3.2 Continuing Example 5.3.1, we need to check whether the attacker can deduce x_2 from the facts $\{\mathsf{K}^{\uparrow}(x_1 \oplus x_3), \mathsf{K}^{\uparrow}(x_3), \mathsf{K}^{\downarrow}(x_1 \oplus x_2)\}$. Therefore, we check whether $\textsc{Ded}_{\oplus,\mathscr{C}}(\{x_2\}, \{x_1 \oplus x_3, x_3\}, \{x_1 \oplus x_2\})$ holds. By following the definition in Figure 5.2, as $U \neq \emptyset$, we decompose the terms in $\{x_1 \oplus x_3, x_3\}$. This recursively induces checks:

- $\text{Ded}_{\oplus,\mathscr{C}}(\{x_2\},\{x_3\},\{x_1 \oplus x_2, x_1 \oplus x_3\})$
- $\text{Ded}_{\oplus,\mathscr{C}}(\{x_2\},\{x_1,x_3\},\{x_1 \oplus x_2\})$
- $\text{Ded}_{\oplus,\mathscr{C}}(\{x_2\},\{x_1 \oplus x_3\},\{x_1 \oplus x_2, x_3\})$

For each of these three recursive calls, $x_2 \notin U \cup D$ and x_2 is not a function application and thus cannot be decomposed further. Hence the calls to DedNaive fail. Further recursive calls result in the following two final decompositions (they appear multiple times, but we merge duplicates):

- $\text{Ded}_{\oplus,\mathscr{C}}(\{x_2\},\{\emptyset\},\{x_1 \oplus x_2, x_1 \oplus x_3, x_3\})$
- $\text{Ded}_{\oplus,\mathscr{C}}(\{x_2\},\{\emptyset\},\{x_1 \oplus x_2, x_1, x_3\})$

The calls to DedNaive fail for the same reason as above and we finally evaluate:

$$thy_1 = \text{Ded}^{\oplus,\mathscr{C}}_{tam}(\{x_2\},\{x_1 \oplus x_3, x_3, x_1 \oplus x_2\}) \text{ and}$$
$$thy_2 = \text{Ded}^{\oplus,\mathscr{C}}_{tam}(\{x_2\},\{x_1, x_3, x_1 \oplus x_2\})$$

The call to thy_1 generates a Tamarin theory with the rule

```
[Fr(x1),Fr(x2),Fr(x3)]
--[Gen(x1,x2,x3)]->
[Out(x1 ⊕ x3),Out(x1 ⊕ x2),Out(x3)]
```

that outputs $x_1 \oplus x_3$, $x_1 \oplus x_2$ and x_3 to the adversary and a lemma:

```
Ex x x1 x2 #i #j. Gen(x,x1,x2)@#i & K(x1)@#j
```

stating that the adversary can know x_2. Tamarin finds the following message deduction to show that x_2 can be deduced respecting the normal form condition **N14**:

$$\dfrac{\dfrac{\dfrac{\mathsf{K}^{\downarrow}(x_1 \oplus x_2) \quad \dfrac{\dfrac{\mathsf{K}^{\downarrow}(x_1 \oplus x_3) \quad \dfrac{\mathsf{K}^{\downarrow}(x_3)}{\mathsf{K}^{\uparrow}(x_3)}\,coerce}{}R_{\oplus.1} \quad \dfrac{\mathsf{K}^{\downarrow}(x_1)}{\mathsf{K}^{\uparrow}(x_1)}\,coerce}{}R_{\oplus.1}}{\dfrac{\mathsf{K}^{\downarrow}(x_2)}{\mathsf{K}^{\uparrow}(x_2)}\,coerce}$$

The call for thy_2 is similar. As both calls prove the lemma true, we know that Tamarin can deduce x_2 from the facts $\{\mathsf{K}^{\uparrow}(x_1 \oplus x_3), \mathsf{K}^{\uparrow}(x_3), \mathsf{K}^{\downarrow}(x_1 \oplus x_2)\}$ without applying the chained instance from Example 5.3.1.

Restrictions during Tamarin calls To check $\text{Ded}^{\mathsf{f},\mathscr{C}}_{tam}(T,D)$, we add optimizations to the generated Tamarin theory to improve efficiency.

- We restrict the rule that outputs terms in D in $\text{Ded}^{\mathsf{f},\mathscr{C}}_{tam}(T,D)$ to be used once only, using the restriction:

  ```
  restriction OnlyOnce: All #j #i.
  OnlyOnce()@#i & OnlyOnce()@#j => #i = #j
  ```

  and annotating the rule with the action `OnlyOnce`.

We observed that otherwise Tamarin explores proofs with multiple, unneeded outputs.

- Similarly, when verifying the NDC property for a function `f`, we restrict the attacker to apply at most two instances of the deconstruction rules for `f`. In all our examples Tamarin can prove the deduction with this restriction, but in case it fails, Tamarin is re-run without this restriction.

5.4. Pruning constructor chains

5.4.1. Constructor chains issue

While developing the support for user defined AC symbols we discovered a bug that may cause Tamarin to report incorrect results on examples using the built-in XOR operator ⊕. Consider the following slightly contrived protocol where `h` is a free symbol modeling, e.g., a hash function.

```
rule r0:
  [ Fr(~n0),Fr(~n1),Fr(~n2),Fr(~n3) ]
  --[OnlyOnce()]->
  [ Name(~n0,~n1,~n2,~n3),Out(~n0⊕~n2),Out(~n1⊕~n3),Out(h(~n2⊕~n3)) ]
rule r1:
  [ Name(~n0,~n1,~n2,~n3),In(~n0⊕~n1⊕x),In(h(x)) ]--[Alive()]-> [ ]
```

and the following property stating that the second rule cannot be executed.

```
All #i. (Alive() @ #i) ==> F
```

This property should not hold as the rule can be executed as follows: for the first input of rule `r1` the attacker applies ⊕ to the two first outputs of `r0`, obtaining `~n0⊕~n1⊕~n2⊕~n3`; for the second input of rule `r1` the attacker simply provides the third output of `r0`.

However, Tamarin proves this property to hold. Investigating the cause of this behavior we found an optimization that forbids to unify `x` with the conclusion of the constructor rule of an AC symbol when trying to solve the open premise $K^{\downarrow}(\texttt{~n0⊕~n1⊕x})$. This indeed avoids a costly AC unification. In interactive mode, when guiding the tool to first solve the open premise $K^{\downarrow}(\texttt{h(x)})$, Tamarin successfully finds the trace witnessing the property violation, as `x` becomes instantiated by `~n2⊕~n3`. Hence, depending on the order of solving open goals, the property can be shown to either hold or be violated.

Remark 5.4.1 We reported this bug to the Tamarin developer team and a corresponding issue has been opened on github. Having investigated existing case studies using exclusive or, we did not find any examples where a wrong result had been produced.[5].

An obvious fix to restore soundness is to deactivate the optimization. However, without this optimization, Tamarin does not terminate on many examples: testing 128 lemmas using the built-in ⊕ from the Tamarin example folder, 19 (15%) do not terminate anymore (see Section 5.5.1

5: Looking ahead, our fixed version does not terminate on 5 lemmas that previously did terminate with already very large proofs, as the number of cases to consider unavoidably increases and appears to go beyond what our machine can handle. We thus cannot assert the correctness of these lemmas which could possibly be achieved with an interactive proof, or using a different configuration.

for the complete analysis). Moreover, cases of non-termination do include security proofs as well as falsification and reachability as well as equivalence properties, showing no clear pattern of non-termination.

Non-termination can be illustrated by the following simple example. Suppose we have an open premise $\mathsf{K}^{\uparrow}(n + x)$ where $+$ is an AC operator, n a fresh name and x a variable. One way to solve this premise is to apply the construction rule for $+$, which requires AC unification of the open premise $n + x$ and the conclusion of the constructor $x^1 + x^2$. Among the most general unifiers we have $\{x \mapsto x_1 + x_2;\ x^1 \mapsto n + x_1;\ x^2 \mapsto x_2\}$. We depict in Figure 5.3 the initial open premise (on the left) and the resulting constraint system after applying the constructor rule (on the right). Applying the constructor creates a new open premise $\mathsf{K}^{\uparrow}(n + x_1)$. Hence, we are exactly in the same case and we can create a new open premise $\mathsf{K}^{\uparrow}(n + x_{11})$, and so on.

$$i:\ \frac{\mathsf{K}^{\uparrow}(n+x)\quad \cdots}{\cdots} \qquad\qquad j:\ \frac{\mathsf{K}^{\uparrow}(n+x_1)\quad \mathsf{K}^{\uparrow}(x_2)}{\mathsf{K}^{\uparrow}(n+x_1+x_2)}\ \downarrow\ i:\ \frac{\mathsf{K}^{\uparrow}(n+x_1+x_2)\quad \cdots}{\cdots}$$

Figure 5.3.: Illustration of AC unification issue

5.4.2. Extending $\rightsquigarrow_P$ with $\mathrm{DG}_{+,\bot}$

To recover termination wherever possible we design a new constraint solving rule to prune unnecessary chains of successive applications of a construction rule for AC operators. This rule is shown in Figure 5.5.

Intuitively, the graph constraints of the constraint system form a partial, symbolic dependency graph. Our new constraint solving rule states that we can remove a constraint system if it contains

- node constraints $i_1 : ri_1 \ldots, i_n : ri_n$ that each apply the constructor rules of the same AC symbol $+$,
- edge contraints that chain these nodes to form a subgraph, that is the conclusion of each node i_j (except the final node i_n) is connected to the premise of another node i_k such that $i_j < i_k$ (ensuring the graph is acyclic), and
- two node constraints with a premise that is trivial for $+$. A premise is trivial[6] for $+$ if it is of the form $\mathsf{K}^{\uparrow}(x_1 + \ldots + x_n)$ where x_i are message variables (and n may be 1, i.e., $\mathsf{K}^{\uparrow}(x_1)$ is trivial).

Continuing the previous example, applying the construction rule for $+$ twice we obtain the constraint system in Figure 5.4. We discard this constraint system as we have two rules with a trivial premise: $\mathsf{K}^{\uparrow}(x_{12})$ for node k and $\mathsf{K}^{\uparrow}(x_2)$ for j. Intuitively, such a constraint system can indeed be discarded as any solution of the above constraint system that instantiates open premises $\mathsf{K}^{\uparrow}(n + x_{11})$ to $\mathsf{K}^{\uparrow}(n + t_{11})$, $\mathsf{K}^{\uparrow}(x_{12})$ to $\mathsf{K}^{\uparrow}(t_{12})$ and $\mathsf{K}^{\uparrow}(x_2)$ to $\mathsf{K}^{\uparrow}(t_2)$ can be used to construct a solution of the previous, simpler one by instantiating $\mathsf{K}^{\uparrow}(n + x_1)$ to $\mathsf{K}^{\uparrow}(n + t_{11})$ and $\mathsf{K}^{\uparrow}(x_2)$ to $\mathsf{K}^{\uparrow}(t_{12} + t_2)$.

We also implemented another optimization: when two constraint systems are identical up to a permutation of the two premises (and a renaming of variables) of an AC construction rule, we only keep one of them.

A technical problem is that the correctness proof of TAMARIN's constraint solving algorithm shows that constraint solving rules preserve the set of P-solutions[7] (cf. Theorem 4.2.4). This is not the case for our new rule as we indeed discard (locally) some solutions. We therefore need to show that there exists (globally) another constraint system that admits the discarded solutions. Moreover, the solutions are not exactly the same as we do not necessary apply the same number of constructor rules for $+$.

6: We extend the definition of a trivial term given in Section 4.2.4 to premises in a constraint system.

$$k:\ \frac{\mathsf{K}^{\uparrow}(n+x_{11})\quad \mathsf{K}^{\uparrow}(x_{12})}{\mathsf{K}^{\uparrow}(n+x_{11}+x_{12})}\ \downarrow\ j:\ \frac{\mathsf{K}^{\uparrow}(n+x_{11}+x_{12})\quad \mathsf{K}^{\uparrow}(x_2)}{\mathsf{K}^{\uparrow}(n+x_{11}+x_{12}+x_2)}\ \downarrow\ i:\ \frac{\mathsf{K}^{\uparrow}(n+x_{11}+x_{12}+x_2)\quad \cdots}{\cdots}$$

Figure 5.4.: Constructor chain

7: As defined in Section 4.2.4, a P-solution is a normal dependency graph that satisfies all constraints.

$\mathbf{DG}_{+,\bot}$: $\Gamma \rightsquigarrow_P \bot$
if + is an AC symbol, $i_1 \ldots, i_n \in \Gamma$ are instances of $\mathsf{K}(x_1), \mathsf{K}(x_2) \dashv\!\vdash \mathsf{K}(x_1 + x_2)$, and $\forall 1 \leq j < n$:
1. $i_j < i_{j+1}$ and
2. $\exists v, k, \mathsf{f}_j, \mathsf{f}_k.\ j < k \leq n, (i_j, 1) \rightarrowtail (i_k, v) \in \Gamma, ((i_j, 1), \mathsf{f}_j) \in cs(\Gamma), ((i_k, v), \mathsf{f}_k) \in ps(\Gamma)$, and $\mathsf{f}_j =_{\mathscr{AC}} \mathsf{f}_k$, and
3. $\exists \ell, \ell', u, u', \mathsf{f}, \mathsf{f}'.\ 1 \leq \ell \neq \ell' \leq n, ((i_\ell, u), \mathsf{f}) \in ps(\Gamma), ((i_{\ell'}, u'), \mathsf{f}') \in ps(\Gamma)$, f and f' are trivial for +

Figure 5.5.: New constraint solving rule to prune constructor chains for AC symbols.

To prove soundness of our new constraint reduction rule, for each (binary) AC symbol +, we introduce a new n-ary symbol $+_n$[8] for each $n \geq 2$. This allows us to introduce the notion of **flat** normal dependency graph.[9]

Definition 5.4.1 *A normal dependency graph (I, D) is **flat** if there are no nodes i and j such that $(i, 1) \rightarrowtail (j, u) \in D$ and both i and j apply a constructor rule for the same AC symbol +, i.e.,*
$I_i = [K^{\uparrow}(t_i^1), \ldots, K^{\uparrow}(t_i^{n_i})] \dashv\!\vdash [K^{\uparrow}(t_i^1 + \ldots + t_i^{n_i})]$ *and*
$I_j = [K^{\uparrow}(t_j^1), \ldots, K^{\uparrow}(t_j^{n_j})] \dashv\!\vdash [K^{\uparrow}(t_j^1 + \ldots + t_j^{n_j})]$.

Any normal dependency graph can be easily transformed into a flat one using a flattening relation $\rightarrow_{\mathsf{f},+}$ (see Appendix C.2.2 for the formal definition). The flattening of a normal dependency graph dg is the graph obtained by replacing each node applying a constructor rule for + with a node applying the corresponding n-ary constructor rule, and merging consecutive nodes applying the same AC constructor rule into a single node. In Figure 5.6 we depict the relevant parts of two graphs related by $\rightarrow_{\mathsf{f},+}$.

We write $\rightarrow_{\mathsf{f},+}^*$ for the reflexive transitive closure of $\rightarrow_{\mathsf{f},+}$. As each time $\rightarrow_{\mathsf{f},+}$ applies we remove a node, $\rightarrow_{\mathsf{f},+}^*$ terminates. Moreover, $\rightarrow_{\mathsf{f},+}$ is confluent and yields a unique normal form (up to $\mathscr{AC}$) for ndg. We denote this normal form by $ndg \downarrow_{\mathsf{f},+}$. We write $ndg \downarrow_{\mathsf{f}}$ to denote the normal form w.r.t. all AC operators, and $ndgraphs(P) \downarrow_{\mathsf{f}}$ the set of flattened normal dependency graphs for the protocol P.

We show that visible traces of normal dependency graphs and their flattened counterparts are identical.

Lemma 5.4.1 *Let ndg be a normal dependency graph. Then*

$$\overline{trace(ndg)} =_{\mathscr{AC}} \overline{trace(ndg \downarrow_{\mathsf{f}})}$$

Proof. The proof is a simple induction on the number of applications of $\rightarrow_{\mathsf{f},+}$ (for all AC symbols +) needed to reach the normal form. We observe that when $ndg \rightarrow_{\mathsf{f},+} ndg'$, then the visible traces of ndg' are the same (we remove one silent edge). □

8: By abuse of notation we will omit the subscript n as it will be clear from the context.

9: Note that the notion of flat normal dependency graph is purely a proof technique, even though internally Tamarin has an n-ary representation for AC symbols.

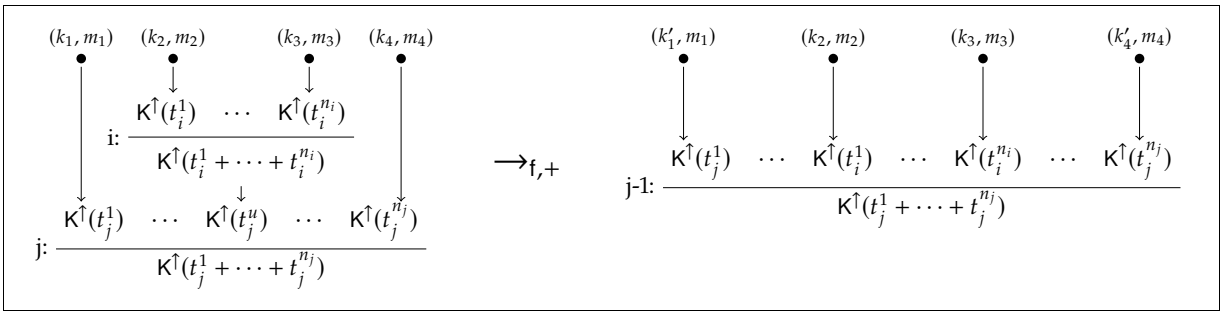


Figure 5.6.: Depicting one flattening step where $t_j^u =_{\mathcal{AC}} t_i^1 + \ldots + t_i^{n_i}$.

Lemma 5.4.2 *Given a protocol P we have*

$$\{\overline{trace(ndg)} \mid ndg \in ndgraphs(P)\} =_{\mathcal{AC}} \{\overline{trace(ndg)} \mid ndg \in ndgraphs(P)\downarrow_{\mathsf{f}}\}$$

Proof. The first inclusion is a consequence of the previous lemma. We can show the other inclusion by an induction on the number of premises of the flattened AC constructor rules: for each additional premise one needs to add a new binary constructor. This can be done at any premise or after the conclusion, allowing to build all possible combinations. Note that again this does not change the visible traces. □

In the following we assume, without loss of generality, that when a term t is written as $\sum_i^{n_t} t_i + \sum_i^{n_x} x_i$ then t_i are non-variable terms that do not have + as their root symbol and x_i are variables. Indeed, any term t can always be written in this form modulo AC.

When dealing with constructor chains of an AC symbol +, we will require to unify modulo AC $x^1 + x^2$ with another term. Unification with $x^1 + x^2$ has particular properties.

Definition 5.4.2 *Given an AC symbol +, we define $\overset{+x}{\leq}$ as the smallest reflexive and transitive relation on terms such that*

- $\overset{+x}{\leq} \subset =_{\mathcal{AC}}$,
- $x \overset{+x}{\leq} x_1 + x_2$ *for any variables x, x_1, x_2 and*
- $\overset{+x}{\leq}$ *is closed under application of function symbols.*

We also write $t_1 \overset{+x}{=} t_2$ when $t_1 \overset{+x}{\leq} t_2$ or $t_2 \overset{+x}{\leq} t_1$.

Intuitively, $t_1 \overset{+x}{\leq} t_2$ if t_2 has been obtained from t_1 by replacing a variable by a sum of variables.

In the following, we suppose that $unif_{\mathcal{AC}}(t_1 =^? t_2)$ returns a *minimal* complete set of AC unifiers, i.e., there do not exist $\sigma_1, \sigma_2 \in unif_{\mathcal{AC}}(t_1 =^? t_2)$ such that $\sigma_1\theta = \sigma_2$ for some unifier θ.

Lemma 5.4.3 *If* $\sigma \in unif_{\mathscr{AC}}(x^1 + x^2 =^? t)$ *then* $t \overset{+x}{\leq} t\sigma$.

Moreover, if

- $x^1\sigma = \sum_i^{n_t^1} t_i^1 + \sum_i^{n_x^1} x_i^1$ *and*
- $x^2\sigma = \sum_i^{n_t^2} t_i^2 + \sum_i^{n_x^2} x_i^2$

and $n_x^1 > 0$ *and* $n_x^1 + n_t^1 > 1$ *then there exists* $\sigma' \in unif_{\mathscr{AC}}(x^1 + x^2 =^? t)$ *such that*

- $x^1\sigma' = \sum_i^{n_t^1} u_i^1 + \sum_i^{m_x^1} x_i^1$,
- $x^2\sigma' = \sum_i^{n_t^2} u_i^2 + \sum_i^{m_x^2} x_i^2$ *and*
- $m_x^1 < n_x^1$ *and* $m_x^1 + m_x^2 \leq n_x^1 + n_x^2$, $\sum_i^{n_t^j} u_i^j \overset{+x}{\leq} \sum_i^{n_t^j} t_i^j$ $(j \in \{1, 2\})$.

and symmetrically if $n_x^2 > 0$.

In other words, non variable terms are preserved (up to $\overset{+x}{\leq}$), but the number of variables may increase. Moreover, variables may be "shifted around" by choosing another unifier, possibly adding less variables.

Example 5.4.1 We have $\sigma \in unif_{\mathscr{AC}}(x^1 + x^2 =^? n + x)$ where $x^1\sigma = n + x_1^1$ and $x^2\sigma = x_1^2$. We indeed have preserved the non-variable term n and added an additional variable: the original term only contains one variable x, while the instantiated term contains two variables x_1^1 and x_1^2.
We also have $\sigma' \in unif_{\mathscr{AC}}(x^1 + x^2 =^? n + x)$ where $x^1\sigma' = n$ and $x^2\sigma' = x$.

We will now adapt the correctness lemma from [18] to prove that the set of flattened (rather than all) P-solutions is preserved by all existing rules of Tamarin's constraint solving algorithm and by our new rule $\mathbf{DG}_{+,\perp}$.

[18]: Schmidt et al. (2012), *Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties (extended version)*

Lemma 5.4.4 *Let* $\Omega_0, \ldots, \Omega_n$ *be a sequence of sets of constraint systems such that*

- $\Omega_0 = \{\{\varphi\}\}$ *where* φ *is the negation of some guarded trace property rewritten into negation normal form,*
- *for all* $0 \leq i < n$, *there exists* $\Gamma \in \Omega_i$ *such that* $\Gamma \rightsquigarrow_P \{\Gamma_1, \ldots \Gamma_k\}$, *and* $\Omega_{i+1} = (\Omega_i \setminus \{\Gamma\}) \cup \{\Gamma_1, \ldots, \Gamma_k\}$

The union of sets of flattened *P-solutions of constraint systems in* Ω_i *and* Ω_{i+1} *are equal.*

The proof is given in Appendix C.2.2.

5.5. Evaluation and Case studies

We implemented our approach on a fork of Tamarin, based on the `develop` branch of the main Tamarin repository. To define an AC symbol, a user simply needs to add `[AC]` to a function of arity two, e.g., `functions: plus/2 [AC]`. The code is available online [59].

[59]: Klein et al. (2025), *Source code of the presented version of Tamarin and case studies models*

To evaluate the performance and effectiveness of our approach, we assembled a large set of case studies based on both existing theories and new protocol theories. The evaluation was performed on a server with 64 cores and 504 GB of RAM. For each theory, we bounded memory to 110 GB and time to 4 hours, using 3 cores. We restricted our evaluation

to TAMARIN's automated mode, and did not attempt interactive proofs, unless stated otherwise.

5.5.1. Impact of $\mathbf{DG}_{+,\perp}$

First, we evaluate the impact of the new constraint-reduction rule $\mathbf{DG}_{+,\perp}$ on existing models using built-in XOR (the other built-in AC symbols are not affected by the bug described in Section 5.4.1). We extracted all case-studies available on the official TAMARIN github repository that use the built-in theory for XOR. Note that we did not consider files that import a built-in but never use the concerned functions. Moreover, we removed duplicates, SAPIC$^+$ files, tests, and files marked as "experimental" or "not working". The resulting set of examples includes both trace and equivalence properties.

We call `develop` the TAMARIN version available in develop branch of the main repository, that include the unsound optimization. We call `baseline` the TAMARIN version without the unsound optimization and without $\mathbf{DG}_{+,\perp}$.
In total, there are 176 lemmas, `develop` terminates in 128 cases (73%), `baseline` in 109 cases (62%) and our version in 123 cases (70%). The detailed results are given in Table 5.1.

Number of lemmas with	`develop`	`baseline`
same number of steps	116 (66%)	102 (58%)
less steps than our version	7 (4%)	0 (0%)
more steps than our version	0 (0%)	7 (4%)
both fail to terminate	48 (27%)	53 (30%)
only our version fails to terminate	5 (3%)	0 (0%)
only the other version fails to terminate	0 (0%)	14 (8%)

Table 5.1.: Results of our version with $\mathbf{DG}_{+,\perp}$ compared to `develop` and `baseline` versions on all examples.

For instance, to read the table, the first row should be interpreted as follows: among the 176 lemmas (representing 100% of the cases), 116 lemmas (66% of 176) terminate in both our version and the `develop` version, and 102 (58%) terminate in both our version and the `baseline` version.

$\mathbf{DG}_{+,\perp}$ is a significantly better solution than the `baseline`: more lemmas (14) are able to terminate, sometimes in less steps. Moreover, on all examples with the same number of steps, our version is on average 34s faster or 4% (median 0.01s/0.1%). Compared to `develop`, the correction of the optimization has a small impact on practical case studies: in 93% of the examples, we obtain the same result (termination and number of steps) as in `develop`, and our version is on average 48s or 100% (median 0.73s/8%) slower. The large difference between average and median is caused by a few examples (attacks and equivalences) with a large difference, although in most cases the difference is very small. In 4% of the cases, our version terminates with more steps.

5.5.2. Existing AC symbols

Second, we compare all built-in AC symbols with user defined ones as the built-in theories provide a baseline we can compare to.

XOR Our extension of Tamarin allows the user to define an AC function symbol $\mathtt{xor}^{\mathtt{UD}}$ equipped with the same equational theory as the built-in $\oplus$:

```
functions: xor^UD/2 [AC], z^UD/0
equations: xor^UD(x,xor^UD(y,x))= y, xor^UD(x,x)= z^UD, xor^UD(x,z^UD)= x
```

Tamarin generates exactly the same deconstruction rules for $\mathtt{xor}^{\mathtt{UD}}$ as for the built-in $\oplus$.

Multisets We also explored the built-in theory of multisets. We encoded the theory using the AC symbol $\mathtt{union}^{\mathtt{UD}}$ and the $\mathtt{elem}^{\mathtt{UD}}$ symbol which allows to extract elements from a multiset:

```
functions: union^UD/2 [AC], elem^UD/1 [destructor]
equations: elem^UD(union^UD(x, y)) = x
```

Again, this encoding generates the same rules as the built-in multiset operator.

DH exponentiation We also wrote a user defined version of Diffie-Hellman exponentiation using the same theory as the built-in one. However, the built-in version considers only the class of $*$-restricted protocols, i.e., protocols that do not use multiplications. This restriction allows for hard-coded optimizations, which we cannot reconstruct easily. In particular, the built-in version of Diffie-Hellman exponentiation does not contain deconstruction rules for the multiplication symbol as all products are necessarily generated by the adversary, as well as numerous tailored normal form conditions. This leads to a user defined version which is largely less efficient, but also incomparable to the built-in one. Hence we did not perform any detailed experiments.

Evaluation We use the set of lemmas presented in Section 5.5.1 and all case studies available on the official Tamarin github repository that use the built-in theory for multisets or XOR. For XOR we obtain 156 lemmas, and 295 lemmas for multisets. Note that some lemmas concern both XOR and multiset, and appear thus in both batches. This set of examples includes both trace and equivalence properties for multiset, but only trace properties for XOR. We excluded equivalence properties for XOR as we obtain incomparable results: the user-defined version often finds spurious attacks, where the built-in once concludes correctly. In fact, the built-in XOR uses a hard-coded optimization to avoid these cases (see [50] for details), which our approach does not.

We ran the examples from these two sets both with the built-in and with the user-defined version of the operators: the results are detailed in Table 5.2. It turns out that even the built-in version does not terminate on all lemmas. We did not find a reason for all of those, but in most

[50]: Dreier et al. (2020), *Verification of Stateful Cryptographic Protocols with Exclusive OR*

cases a manual inspection of the concerned files showed that they were annotated as being "manual". To focus on the interesting cases, we removed lemmas that fail to terminate in both built-in and user-defined versions (this concerns 46 lemmas for XOR and 47 lemmas for multiset). We end up with 110 lemmas for XOR and 248 for the multiset operator.

	XOR	multiset
Number of lemmas	110 (100%)	248 (100%)
Termination built-in	110 (100%)	248 (100%)
Termination user-defined	74 (67%)	190 (77%)
Number of lemmas with		
same number of steps	71 (65%)	122 (50%)
less steps than built-in	2 (2%)	34 (14%)
more steps than built-in	1 (1%)	34 (14%)

Table 5.2.: Results of the user-defined version of XOR and multiset operators compared to the built-in version

The difference in termination and number of proof steps can be explained as follows. In automated mode, Tamarin ranks possible steps following a heuristic (built-in or user-defined) and chooses the first one. The names of symbols, nonces and variables can influence this ranking, leading Tamarin to sometimes follow a different path in the proof tree when replacing the built-in with a user-defined version where the names of the functions change (we cannot reuse the same names as they are reserved). We checked multiple examples where the built-in version terminates but not the user-defined version: in all cases, the proof strategy diverged at some point. We were not able to transpose the proof strategy used in the built-in case to the user-defined case; the proofs are typically far too long to do this manually. However, we checked on multiple smaller examples where both approaches terminate that a shorter proof can be reproduced in interactive mode by manually following the proof strategy of the built-in version.

Overall, in 65% of the examples for XOR and 50% of the examples for the multiset operator we obtain the exactly same behavior: both terminate with the same number of steps. In these cases the only difference is a small additional amount of time due to the computation of Check_NDC: in multiset case, this amount of time is negligible (a few seconds at most), and in XOR case this time difference is on average 207s.

5.5.3. Improvement of existing case studies

In [49] Dreier *et al.* defines a bound on the number of successive destructor applications of the same symbol, formalized in the normal form condition **N12**. This bound is a function of the number of subterms. For symbols that have the NDC property this bound is always one. Therefore, our bound can sometimes be smaller than the previous bound defined in [49]. This is indeed the case on the following equational theory for trapdoor commitments:

[49]: Dreier et al. (2017), *Beyond Subterm-Convergent Equational Theories in Automated Verification of Stateful Protocols (extended version)*

```
open(tdcommit(m,r,td),r) = m
tdcommit(m2,f(m1,r,td,m2),td) = tdcommit(m1,r,td)
open(tdcommit(m1,r,td),f(m1,r,td,m2)) = m2
f(m2,f(m1,r,td,m2),td,m3) = f(m1,r,td,m3)
```

This theory was used in [49] to model a receipt-free e-voting protocol by Okamoto [60].

[60]: Okamoto (1996), *An electronic voting scheme*

Among the deconstruction rules computed for this equational theory, the following rule is bounded to three consecutive repetitions:

$$d_{\mathsf{f}}\ \frac{\mathsf{K}^{\downarrow}(\mathsf{f}(m_1, r, td, m_2)) \quad \mathsf{K}^{\uparrow}(m_2) \quad \mathsf{K}^{\uparrow}(td) \quad \mathsf{K}^{\uparrow}(m_3)}{\mathsf{K}^{\downarrow}(\mathsf{f}(m_1, r, td, m_3))}$$

As f has the NDC property, we now bound the number of repetitions to one, leading to fewer proof steps, as demonstrated by the proof of receipt-freeness in Table 5.3.

Table 5.3.: Improvement of the number of proof steps for receipt-freeness of Okamoto's e-voting protocol

	w/o Check_NDC			**with** Check_NDC		
Property	Result	Time	Steps	Result	Time	Steps
eligibility	✓	1s	5	✓	3s	5
privacy	✓	41s	3701	✓	48s	3701
receipt-freeness	✓	630s	41172	✓	552s	29248

5.5.4. Toy voting system based on distributed decryption

We now show how we can model an equational theory for distributed decryption and re-encryption. Such theories previously exceeded the modeling capabilities of TAMARIN. We model a toy voting system where the trust among the talliers is distributed by not providing a single authority with the decryption key. Otherwise, a single tallier that knows the private election key could easily break vote privacy, and would create a single point of failure.

Consider an election with n voters and m talliers. The protocol is depicted in Figure 5.7 and proceeds as follows.

- Each voter i encrypts her vote v_i with the election public key $\mathsf{pk}(k)$ using randomness r_i. To authenticate the ballot she signs it with her private signature key sk_i and sends the ballot and signature to a mixnet.
- The mixnet verifies eligibility of each ballot by checking the signature. Then, the mixnet performs a re-encryption mix, i.e., re-encrypts and shuffles the ballots. It then sends the resulting multiset of ballots to each of the m talliers through an authenticated channel. (A more realistic protocol would include zero knowledge proofs of correct mixing that we omit here for simplicity.)
  We model re-encryption with an AC symbol $*$ that combines the initial randomness with the randomness used for re-encryption:

```
reenc(enc(m,pk(k),r1),r2,pk(k)) = enc(m,pk(k),r1*r2)
```

- To allow for distributed decryption, the secret key of the election is actually chosen to be a combination, denoted by +, of partial keys: $k = \sum_{j=1}^{m} k_j$. Here + is again an AC symbol. Each tallier j holds a partial key k_j and computes a partial decryption of the ballots sent by the mix server. The partially decrypted ballots are then sent to the election authority through an authenticated channel.

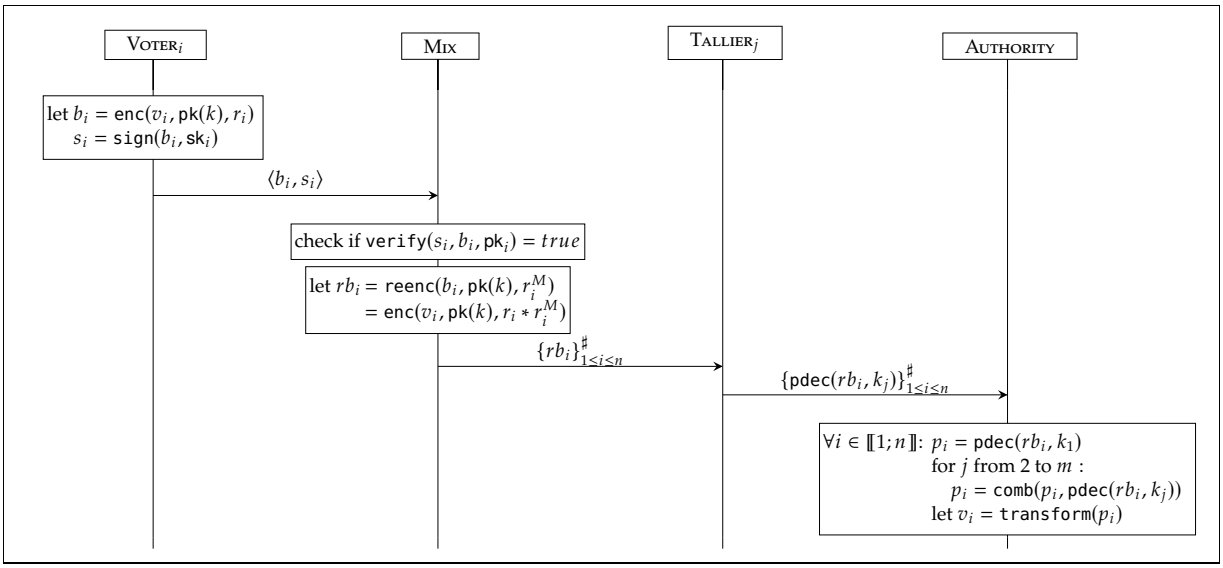


Figure 5.7.: Toy voting protocol using distributed decryption

- The election authority combines the partial decryptions to retrieve the plaintext votes v_i and publishes the result. Such a distributed decryption can be modeled by the following theory that allows to combine partial decryptions:

  ```
  comb(pdec(c,sk1),pdec(c,sk2)) = pdec(c,sk1+sk2)
  ```

  We add an additional equation that transforms a partial encryption into a regular one:

  ```
  transform(pdec(c,k)) = dec(c,k)
  ```

For simplicity we only consider two voters and two authenticators in our TAMARIN model.

We prove an eligibility property of the protocol: if a vote is published by the election authority, it has been validated and registered by the mixnet before.

```
All v #j. VotePublished(v)@#j ==>
  Ex pub r k #i. Register(<enc(v,pub,r),sign(enc(v,pub,r),k)>)@#i & #i<#j
```

We were not able to prove privacy modeled as an observational equivalence property; even when using a server with 64 cores and 504G of RAM we ran out of time or memory. All results are summarized in Table 5.4.

5.5.5. Exponentiation Mixnet

Another case study we are now able to model in TAMARIN is the exponentiation mixnet analyzed in [61] using PROVERIF. The mixnet consists of m mix servers M_j and is based on El Gamal encryption. It is used to shuffle n private keys sk_i whose corresponding public keys are $pk_i = g^{sk_i}$.

[61]: Dreier et al. (2024), *Shaken, not Stirred -Automated Discovery of Subtle Attacks on Protocols using Mix-Nets*

Each mix server M_j receives as input a list of public keys and an element h (which is simply g for M_1). It generates a fresh random r_j and computes ${pk_i}^{r_j}$ for each public key pk_i, as well as h^{r_j}. It then sends a permutation

of the masked public keys as well as h^{r_j} to the next mix server. The last server displays the list of all ${pk_i}^r$ and g^r where $r = \prod_{j=1}^{m} r_j$. Each user i can retrieve their public key by computing $(g^r)^{sk_i} = {pk_i}^r = (g^{sk_i})^r$.

To model this exponentiation mixnet, we need to model a commutative exponentiation. In ProVerif, this was modeled using equations for two levels of exponentiation [61].

```
exp(exp(g,x),y) = exp(exp(g,y),x)
exp(exp(exp(g,x),y),z) = exp(exp(exp(g,x),z),y)
```

Note that the modeling in ProVerif requires `g` to be a constant to effectively bound the level of exponentiations. In Tamarin, we use a user-defined AC symbol `mult` to model multiplication of the exponents. Our equational theory is more general, as the number of levels of exponentiation is not bounded (and g is a variable).

```
functions: expo/2,mult/2[AC],zkp/3,verify/3,true/0
equations: expo(expo(g,x),y) = expo(g,mult(x,y))
```

We are able to find an attack on anonymity reported in [61] on this exponentiation mixnet. We model the fix to avoid this attack, suggested by Dreier *et al.*, which integrates a zero-knowledge proof as follows:

```
verify(zkp(expo(g,x),x,g),expo(g,x),g) = true
```

However, as for privacy in the previous section, we were not able to prove this fix (modeled in ProVerif in [61]) in Tamarin. The results are summarized in Table 5.4.

In Tamarin, it is also possible to model this exponentiation mixnet with the built-in Diffie-Hellman equational theory. However, using the built-in symbol precludes modeling protocols that use the obtained public keys for encryption as in [61] using, e.g., the following equation:

```
dec(enc(m,g,expo(g,s),r),g,s) = m
```

However, this equation is possible using a user-defined AC symbol as above.

5.5.6. Summary

The results of the case studies presented in Sections 5.5.4 and 5.5.5 are summarized in Table 5.4: For each proof, we give the result of the proof (if the lemma is proven to be true or false), the processing time on a server with 504GB of RAM and the number of steps. It is important to note that all proofs are run with Tamarin's automated mode, with the exception of the toy voting system's source lemma [62]. In this instance, a semi-manual approach was employed, whereby the Tamarin's interactive mode was used to manually choose the first proof steps. Thereafter, the tool's automated mode was able to complete the proof.

[62]: Cortier et al. (2022), *Automatic generation of sources lemmas in TAMARIN: towards automatic proofs of security protocols*

The experiments are available in the repository [59].

Table 5.4.: Summary of results for toy voting system and exponentiation mixnet. We mark lemma proven true with ✓and false with ×. *SM = Semi-Manual

Protocol	Property	Result	Time	Steps
Toy Voting System	eligibility	✓	390s	16
	executability	✓	422s	30
	source lemma	✓	1192s (SM*)	20371
Exponentiation Mixnet	privacy (as observ. equiv.)	×	83s (BFS)	3008

5.6. Future Work

In most of our examples, the computation of $\mathsf{Check\_NDC}_{\mathscr{E}}(\mathtt{f})$ is negligible or introduces only a small overhead during the precomputation phase. However, in some cases, such as for user-defined XOR, this precomputation can take up to 200 seconds. In such cases, it would be beneficial to introduce a caching mechanism to store the result and perform this check only once for a given equational theory.

Since we compare built-in XOR and multiset theories with our user-defined counterparts, it would also be interesting to extend this comparison to the built-in theory for Diffie-Hellman exponentiation with the simpler version introduced in Section 5.5.5. While fewer attacks are expected with this less sophisticated version, it may provide performance benefits in terms of runtime and/or the number of steps required for simpler models and lemmas.

In some examples, we observe that a function `f` has the NDC property because Tamarin is able to insert an additional deconstruction rule between two successive instances of deconstruction rules for `f`. This means the tool resolves the case in three steps instead of two. It would be interesting to investigate whether these are isolated occurrences, whether such behavior could negatively impact future proofs, and whether our algorithm should be refined to determine if the deconstruction rules for a function should be chained two, three, or a finite number of times, instead of exactly once.

6. Conclusion & Future work

In this thesis, we presented two main contributions.

Comprehensive formal analysis of the EDHOC protocol We illustrate how recent state-of-the-art developments from the formal method community can be leveraged in the standardization processes of security protocols. Concretely, we performed an automated analysis of the EDHOC protocol (draft 12). Building on automated tools and a systematized workflow allowed us to consider combinations of strong threat models and identify multiple ways to strengthen the protocol. Those were reported to the working group and lead to lively interactions over GitHub issues and pull-requests. Several of our findings lead to changes integrated in draft 14 which we subsequently verified to confirm that the proposed changes mitigate previously identified weaknesses.

Handling user-defined associative and commutative (AC) operators in Tamarin AC symbols are often part of equational theories. However, automated tool support of such symbols has been limited due to non-termination and performance issues. We introduced AC symbols in user-defined equational theories in the Tamarin Prover. To address the issue of non-termination caused by infinite deduction chains, we designed a novel property, the *No Deconstruction Chain* (NDC) property, along with an algorithm that verifies whether a function symbol satisfies this property given the equational theory. We then introduced a new normal form condition that forbids repeating chains of deconstruction rules for such functions. We also uncovered an unsound pre-existing optimization that restricted the instantiations of construction rules for AC operators and could lead to incorrect results. Removing the optimization lead to non-termination in many cases, so we designed a new constraint reduction rule $\mathbf{DG}_{+,\perp}$ pruning redundant constructor chains, improving the termination of proofs using AC operators. Using our approach we were able to model a user-defined XOR and a user-defined multiset that behave similarly to the existing built-in versions in Tamarin, except for a small overhead at startup when verifying the NDC property. We also modeled a toy voting system using re-encryption and distributed decryption, which was previously out of scope in Tamarin, and we generalized an attack on exponentiation mixnets. All these new models terminate within an acceptable amount of time. Moreover, allowing users to define AC symbol enables more flexible equational theories, such as the use of exponentiation in decryption.

6.1. Future Work

We summarize here the directions for future work discussed throughout this thesis.

Maintenance and Updating of the EDHOC Model As the EDHOC protocol is now standardized as an RFC, our goal is to maintain the formal models presented in Chapter 3 in alignment with the final version of the specification. This includes incorporating elements not yet modeled: the optional fourth message, the integrated session key update mechanism, and the upcoming authentication method based on pre-shared keys. This ongoing maintenance will ensure that our model remains a reference model for the EDHOC protocol.

Dealing with Computation issue for Check_NDC$_{\mathscr{E}}$(f) We presented an algorithm for determining whether a function symbol f has the NDC property (Check_NDC$_{\mathscr{E}}$(f)). While the computation is negligible or introduces only a small overhead in most cases, it can be significantly more expensive in others (for example, in the case of user-defined XOR, where it can take up to 200 seconds). Introducing in Tamarin a caching mechanism to store the results during precomputations would allow this check to be performed only once per equational theory, significantly improving efficiency.
In general, equivalence proofs tend to be very large because of the numerous cases to check. This problem is exacerbated when using AC symbols, which also cause numerous cases to check, leading to situations where in case of complex examples even dedicated servers run out of time or memory. Finding further techniques to deal with this state space explosion is future work.

Improving the algorithm of Check_NDC$_{\mathscr{E}}$(f) In some examples, we observed that a function f satisfies the NDC property because Tamarin can insert an additional deconstruction rule between two successive instances of f's deconstruction rules. In such cases, the tool resolves the case in three steps instead of two. It would be interesting to investigate whether these are isolated cases, whether such behavior could negatively impact future proofs, and whether our algorithm should be refined to allow chaining of deconstruction rules two, three, or a finite number of times, rather than strictly once.

Studying the Impact of Check_NDC$_{\mathscr{E}}$(f) While we have compared built-in XOR and multiset theories with their user-defined counterparts, it would also be interesting to extend this comparison to the built-in Diffie-Hellman exponentiation theory, using the simplified version introduced in Section 5.5.5. Although fewer attacks are expected with the simpler theory, it may offer advantages in terms of runtime and/or the number of proof steps required for simpler models and lemmas.

New Possibilities for Sapic$^+$ and Tamarin Since Sapic$^+$ is integrated into Tamarin, using user-defined AC symbols are straightforward when translated into Tamarin. This opens up broader modeling possibilities, especially if other backend tools adopt support for such features in the future.

Currently, Tamarin only supports convergent equational theories that satisfy the Finite Variant Property (FVP). However, the tool does not

verify whether a given equational theory actually satisfies FVP, it assumes this property holds throughout all stages of precomputation and proof (including in procedures such as $\mathsf{Check\_NDC}_{\mathscr{E}}(\mathtt{f})$). If this assumption is incorrect, the computation of deconstruction rules may not terminate, or the proof results may be wrong. Cheval *et al.* [63] proposed a semi-decision procedure that transforms an equational theory E into a *rewrite theory mimicking* E, thus ensuring the FVP. Integrating such a procedure into Tamarin 's precomputation phase could detect and warn against potential non-termination issues or block proofs when the FVP is not satisfied.

[63]: Cheval et al. (2025), *Automatic verification of Finite Variant Property beyond convergent equational theories*

Furthermore, certain equational theories that do not satisfy the FVP still admit dedicated unification algorithms (e. g., homomorphic encryption with cap unification [64]). Supporting such theories as built-in modules with dedicated algorithms could expand the modeling capabilities of Tamarin.

[64]: Anantharaman et al. (2012), *Unification modulo Homomorphic Encryption*

In addition, the new features introduced in this thesis enable the symbolic analysis of previously unsupported classes of real-world protocols, such as shuffling re-encryption mixnets part of electronic voting systems (e. g., the Estonian e-Voting System IVXV[65], Prêt à Voter [66], JCJ/Civitas [67, 68]), or protocols relying on homomorphic encryption (e. g., Multi-Party Computation with a Semi-Trusted Third Party, TMN [69]), using two AC symbols $h(x +_{in} y) = h(x) +_{out} h(y)$ [63] as common for El-Gamal secret computation in e-voting.

[65]: Arafat (2025), *On the Estonian Internet Voting System, IVXV, SoK and Suggestions*

[66]: Ryan et al. (2006), *Prêt à Voter with Re-encryption Mixes*

[67]: Juels et al. (2005), *Coercion-resistant electronic elections*

[68]: Clarkson et al. (2008), *Civitas: Toward a Secure Voting System*

[69]: Cortier et al. (2006), *A Survey of Algebraic Properties Used in Cryptographic Protocols*

Bibliography

[1] Joeri de Ruiter and Erik Poll. "Formal Analysis of the EMV Protocol Suite". In: *Theory of Security and Applications*. Ed. by Sebastian Mödersheim and Catuscia Palamidessi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 113–129 (cited on pages 2, 112).

[2] David A. Basin, Ralf Sasse, and Jorge Toro-Pozo. "The EMV Standard: Break, Fix, Verify". In: *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 2021, pp. 1766–1781. DOI: 10.1109/SP40001.2021.00037 (cited on pages 2, 3, 8, 112, 113, 118).

[3] Cohn-Gordon Katriel, Cremers Cas, Dowling Benjamin, Garratt Luke, and Stebila Douglas. "A Formal Security Analysis of the Signal Messaging Protocol". In: *Journal of Cryptology* 33.4 (2020), pp. 1914–1983. DOI: 10.1007/s00145-020-09360-1 (cited on pages 2, 112).

[4] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. "A Comprehensive Symbolic Analysis of TLS 1.3". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. Ed. by Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM, 2017, pp. 1773–1788. DOI: 10.1145/3133956.3134063 (cited on pages 2, 8, 27, 112, 118).

[5] Adrien Koutsos. "The 5G-AKA Authentication Protocol Privacy". In: *EuroS&P 2019 - IEEE European Symposium on Security and Privacy*. Stockholm, Sweden: IEEE, June 2019, pp. 464–479. DOI: 10.1109/EuroSP.2019.00041 (cited on pages 2, 112).

[6] David A. Basin, Jannik Dreier, Lucca Hirschi, Sasa Radomirovic, Ralf Sasse, and Vincent Stettler. "A Formal Analysis of 5G Authentication". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. Ed. by David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang. ACM, 2018, pp. 1383–1396. DOI: 10.1145/3243734.3243846 (cited on pages 2, 8, 27, 112, 118).

[7] Véronique Cortier, Pierrick Gaudry, and Stephane Glondu. "Belenios: a simple private and verifiable electronic voting system". In: *Foundations of Security, Protocols, and Equational Reasoning - Essays Dedicated to Catherine A. Meadows*. Ed. by Joshua D. Guttman, Carl E. Landwehr, José Meseguer, and Dusko Pavlovic. Vol. 11565. LNCS. Springer, 2019, pp. 214–238. DOI: 10.1007/978-3-030-19052-1_14 (cited on pages 2, 112).

[8] Véronique Cortier, Alexandre Debant, and Pierrick Gaudry. "A privacy attack on the Swiss Post e-voting system". In: *Real World Crypto Symposium (RWC'22)*. IACR. Amsterdam, Netherlands, 2022 (cited on pages 2, 112).

[9] Houda Ferradi, Rémi Géraud, David Naccache, and Assia Tria. "When organized crime applies academic results: a forensic analysis of an in-card listening device". In: *Journal of Cryptographic Engineering* 6.1 (2016), pp. 49–59. DOI: 10.1007/s13389-015-0112-3 (cited on pages 3, 113).

[10] David Basin, Patrick Schaller, and Jorge Toro-Pozo. "Inducing Authentication Failures to Bypass Credit Card PINs". In: *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 3065–3079 (cited on pages 3, 113).

[11] Alexandre Debant and Lucca Hirschi. "Reversing, Breaking, and Fixing the French Legislative Election E-Voting Protocol". In: *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 6737–6752 (cited on pages 3, 113).

[12] Jan Camenisch et al. "Specification of the identity mixer cryptographic library". In: *IBM Research–Zurich* (2010), pp. 1–48 (cited on pages 4, 114).

[13] Martín Abadi, Bruno Blanchet, and Cédric Fournet. "The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication". In: *J. ACM* 65.1 (2018). DOI: 10.1145/3127586 (cited on pages 6, 12, 16, 116).

[14] I. Cervesato, N.A. Durgin, P.D. Lincoln, J.C. Mitchell, and A. Scedrov. "A meta-notation for protocol analysis". In: *Proceedings of the 12th IEEE Computer Security Foundations Workshop*. 1999, pp. 55–69. DOI: 10.1109/CSFW.1999.779762 (cited on pages 6, 116).

[15] Danny Dolev and Andrew Chi-Chih Yao. "On the security of public key protocols". In: *IEEE Trans. Inf. Theory* 29.2 (1983), pp. 198–207. DOI: 10.1109/TIT.1983.1056650 (cited on pages 7, 117).

[16] Bruno Blanchet. "An Efficient Cryptographic Protocol Verifier Based on Prolog Rules". In: *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), 11-13 June 2001, Cape Breton, Nova Scotia, Canada*. IEEE Computer Society, 2001, pp. 82–96. DOI: 10.1109/CSFW.2001.930138 (cited on pages 8, 117).

[17] Bruno Blanchet, Vincent Cheval, and Véronique Cortier. "ProVerif with Lemmas, Induction, Fast Subsumption, and Much More". In: *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 2022, pp. 69–86. DOI: 10.1109/SP46214.2022.9833653 (cited on pages 8, 23, 29, 117).

[18] Benedikt Schmidt, Simon Meier, Cas Cremers, and David A. Basin. *Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties (extended version)*. Available at http://www.infsec.ethz.ch/content/dam/ethz/special-interest/infk/inst-infsec/information-security-group-dam/research/software/dh_tamarin_extended_v1.pdf. 2012 (cited on pages 8, 21, 23, 29, 50, 54, 55, 58, 60, 65–68, 83, 107, 108, 117).

[19] David A. Basin, Cas Cremers, Jannik Dreier, and Ralf Sasse. "Tamarin: Verification of Large-Scale, Real-World, Cryptographic Protocols". In: *IEEE Secur. Priv.* 20.3 (2022), pp. 24–32. DOI: 10.1109/MSEC.2022.3154689 (cited on pages 8, 117).

[20] Karthikeyan Bhargavan, Vincent Cheval, and Christopher A. Wood. "A Symbolic Analysis of Privacy for TLS 1.3 with Encrypted Client Hello". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*. Ed. by Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi. ACM, 2022, pp. 365–379. DOI: 10.1145/3548606.3559360 (cited on pages 8, 27, 118).

[21] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. "Automated Verification for Secure Messaging Protocols and Their Implementations: A Symbolic and Computational Approach". In: *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*. IEEE, 2017, pp. 435–450. DOI: 10.1109/EUROSP.2017.38 (cited on pages 8, 27, 118).

[22] Andreea-Ina Radu, Tom Chothia, Christopher J. P. Newton, Ioana Boureanu, and Liqun Chen. "Practical EMV Relay Protection". In: *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 2022, pp. 1737–1756. DOI: 10.1109/SP46214.2022.9833642 (cited on pages 8, 118).

[23] Vincent Diemunsch, Lucca Hirschi, and Steve Kremer. "A Comprehensive Formal Security Analysis of OPC UA". In: *34th USENIX Security Symposium (USENIX Security'25)*. Seattle, WA, USA: USENIX Association, Aug. 2025 (cited on pages 8, 118).

[24] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. "DEEPSEC: Deciding Equivalence Properties in Security Protocols Theory and Practice". In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2018. DOI: 10.1109/SP.2018.00033 (cited on pages 8, 23, 29, 118).

[25] Vincent Cheval, Charlie Jacomme, Steve Kremer, and Robert Künnemann. "SAPIC$^+$: protocol verifiers of the world, unite!" In: *31st USENIX Security Symposium*. USENIX Association, 2022 (cited on pages 8, 12, 16–19, 21, 23, 28, 29, 41, 118).

[26] Steve Kremer and Robert Künnemann. "Automated analysis of security protocols with global state". In: *Journal of Computer Security* 24.5 (2016), pp. 583–616. DOI: 10.3233/JCS-160556 (cited on page 12).

[27] Hubert Comon-Lundh and Stéphanie Delaune. "The finite variant property: How to get rid of some algebraic properties". In: *Proceedings of the 16th International Conference on Rewriting Techniques and Applications (RTA'05)*. Ed. by Jürgen Giesl. Vol. 3467. Lecture Notes in Computer Science. Nara, Japan: Springer, Apr. 2005, pp. 294–307 (cited on page 24).

[28] Vincent Cheval, Véronique Cortier, and Mathieu Turuani. "A little more conversation, a little less action, a lot more satisfaction: Global states in ProVerif". In: *Proceedings of the 31st IEEE Computer Security Foundations Symposium (CSF'18)*. IEEE Computer Society Press, July 2018, pp. 344–358. DOI: 10.1109/CSF.2018.00032 (cited on page 26).

[29] Charlie Jacomme, Elise Klein, Steve Kremer, and Maïwenn Racouchot. "A comprehensive, formal and automated analysis of the EDHOC protocol". In: *USENIX Security '23 - 32nd USENIX Security Symposium*. Anaheim, CA, United States, Aug. 2023 (cited on page 27).

[30] Göran Selander, John Preuß Mattsson, and Francesca Palombini. *Ephemeral Diffie-Hellman Over COSE (EDHOC)*. RFC 9528. Mar. 2024. DOI: 10.17487/RFC9528. URL: https://www.rfc-editor.org/info/rfc9528 (cited on page 27).

[31] Mališa Vučinić, Göran Selander, John Preuss Mattsson, and Thomas Watteyne. "Lightweight Authenticated Key Exchange With EDHOC". In: *Computer* 55.4 (2022). DOI: 10.1109/MC.2022.3144764 (cited on page 27).

[32] Göran Selander, John Preuß Mattsson, and Francesca Palombini. *Ephemeral Diffie-Hellman Over COSE (EDHOC)*. Internet-Draft draft-ietf-lake-edhoc-12. Work in Progress. Internet Engineering Task Force, Oct. 2021. 80 pp. (cited on pages 27, 30–34, 38, 39, 43, 44, 46).

[33] Karl Norrman, Vaishnavi Sundararajan, and Alessandro Bruni. "Formal Analysis of EDHOC Key Establishment for Constrained IoT Devices". In: *CoRR* abs/2007.11427 (2020) (cited on page 28).

[34] Cas Cremers and Dennis Jackson. "Prime, Order Please! Revisiting Small Subgroup and Invalid Curve Attacks on Protocols using Diffie-Hellman". In: *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*. IEEE, 2019. DOI: 10.1109/CSF.2019.00013 (cited on pages 28, 37–39).

[35] Dennis Jackson, Cas Cremers, Katriel Cohn-Gordon, and Ralf Sasse. "Seems Legit: Automated Analysis of Subtle Attacks on Protocols that Use Signatures". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. ACM, 2019. DOI: 10.1145/3319535.3339813 (cited on pages 28, 37–39).

[36] Vincent Cheval, Cas Cremers, Alexander Dax, Lucca Hirschi, Charlie Jacomme, and Steve Kremer. "Hash Gone Bad: Automated discovery of protocol attacks that exploit hash function weaknesses". In: *32nd USENIX Security Symposium*. Anaheim, United States, Aug. 2023 (cited on pages 28, 37, 38, 40).

[37] David A. Basin and Cas Cremers. "Know Your Enemy: Compromising Adversaries in Protocol Analysis". In: *ACM Trans. Inf. Syst. Secur.* 17.2 (2014). DOI: 10.1145/2658996 (cited on page 28).

[38] Guillaume Girol, Lucca Hirschi, Ralf Sasse, Dennis Jackson, Cas Cremers, and David Basin. "A Spectral Analysis of Noise: A Comprehensive, Automated, Formal Analysis of Diffie-Hellman Protocols". In: *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020 (cited on page 28).

[39] Charlie Jacomme and Steve Kremer. "An Extensive Formal Analysis of Multi-Factor Authentication Protocols". In: *ACM Trans. Priv. Secur.* 24.2 (2021). DOI: 10.1145/3440712 (cited on pages 28, 40).

[40] Charlie Jacomme, Elise Klein, Steve Kremer, and Maïwenn Racouchot. *LAKE EDHOC models*. https://github.com/charlie-j/edhoc-formal-analysis. 2022 (cited on pages 29, 36, 42, 103).

[41] Simon Josefsson and Ilari Liusvaara. *Edwards-Curve Digital Signature Algorithm (EdDSA)*. RFC 8032. Jan. 2017. DOI: 10.17487/RFC8032. URL: https://www.rfc-editor.org/info/rfc8032 (cited on page 38).

[42] Marc Stevens. "A Survey of Chosen-Prefix Collision Attacks". In: *Computational Cryptography: Algorithmic Aspects of Cryptology*. London Mathematical Society Lecture Note Series. Cambridge University Press, 2021. DOI: 10.1017/9781108854207.009 (cited on page 38).

[43] Karthikeyan Bhargavan and Gaëtan Leurent. "Transcript Collision Attacks: Breaking Authentication in TLS, IKE and SSH". In: *Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2016 (cited on pages 38, 46).

[44] David McGrew. *An Interface and Algorithms for Authenticated Encryption*. RFC 5116. Jan. 2008. DOI: 10.17487/RFC5116. URL: https://www.rfc-editor.org/info/rfc5116 (cited on page 46).

[45] Loïc Ferreira. "Computational Security Analysis of the Full EDHOC Protocol". In: *Topics in Cryptology – CT-RSA 2024*. Ed. by Elisabeth Oswald. Cham: Springer Nature Switzerland, 2024, pp. 25–48 (cited on page 48).

[46] Elsa López Pérez, Göran Selander, John Preuß Mattsson, Thomas Watteyne, and Mališa Vučinić. "EDHOC is a New Security Handshake Standard: An Overview of Security Analysis". In: *Computer* (Sept. 2024) (cited on page 48).

[47] Hubert Comon-Lundh and Stéphanie Delaune. "The Finite Variant Property: How to Get Rid of Some Algebraic Properties". In: *Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings*. Ed. by Jürgen Giesl. Vol. 3467. Lecture Notes in Computer Science. Springer, 2005, pp. 294–307. DOI: 10.1007/978-3-540-32033-3_22 (cited on page 51).

[48] David A. Basin, Jannik Dreier, and Ralf Sasse. "Automated Symbolic Proofs of Observational Equivalence". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM, 2015, pp. 1144–1155. DOI: 10.1145/2810103.2813662 (cited on pages 57, 107).

[49] Jannik Dreier, Charles Duménil, Steve Kremer, and Ralf Sasse. "Beyond Subterm-Convergent Equational Theories in Automated Verification of Stateful Protocols (extended version)". In: *POST 2017 - 6th International Conference on Principles of Security and Trust*. Vol. 10204. Proceedings of the 6th International Conference on Principles of Security and Trust. Uppsala, Sweden: Springer, Apr. 2017, pp. 117–140. DOI: 10.1007/978-3-662-54455-6_6 (cited on pages 61, 62, 86, 87, 106, 107).

[50] Jannik Dreier, Lucca Hirschi, Saša Radomirović, and Ralf Sasse. "Verification of Stateful Cryptographic Protocols with Exclusive OR". In: *Journal of Computer Security* 28.1 (Feb. 2020), pp. 1–34. DOI: 10.3233/JCS-191358 (cited on pages 63, 67, 70, 71, 73, 85, 106, 107).

[51] Santiago Escobar, Catherine Meadows, and José Meseguer. "Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties". In: *Foundations of Security Analysis and Design V, FOSAD 2007/2008/2009 Tutorial Lectures*. Ed. by Alessandro Aldini, Gilles Barthe, and Roberto Gorrieri. Vol. 5705. Lecture Notes in Computer Science. Springer, 2007, pp. 1–50. DOI: 10.1007/978-3-642-03829-7_1 (cited on page 70).

[52] Hubert Comon-Lundh and Vitaly Shmatikov. "Intruder Deductions, Constraint Solving and Insecurity Decision in Presence of Exclusive or". In: *18th IEEE Symposium on Logic in Computer Science (LICS 2003), 22-25 June 2003, Ottawa, Canada, Proceedings*. IEEE Computer Society, 2003, p. 271. DOI: 10.1109/LICS.2003.1210067 (cited on page 70).

[53] Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch, and Mathieu Turuani. "An NP decision procedure for protocol insecurity with XOR". In: *Theor. Comput. Sci.* 338.1-3 (2005), pp. 247–274. DOI: 10.1016/J.TCS.2005.01.015 (cited on page 70).

[54] Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch, and Mathieu Turuani. "Complexity results for security protocols with Diffie-Hellman exponentiation and commuting public key encryption". In: *ACM Trans. Comput. Log.* 9.4 (2008), 24:1–24:52. DOI: 10.1145/1380572.1380573 (cited on page 70).

[55] Ralf Küsters and Tomasz Truderung. "Using ProVerif to Analyze Protocols with Diffie-Hellman Exponentiation". In: *Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF 2009, Port Jefferson, New York, USA, July 8-10, 2009*. IEEE Computer Society, 2009, pp. 157–171. DOI: 10.1109/CSF.2009.17 (cited on page 70).

[56] Ralf Küsters and Tomasz Truderung. "Reducing Protocol Analysis with XOR to the XOR-Free Case in the Horn Theory Based Approach". In: *J. Autom. Reason.* 46.3-4 (2011), pp. 325–352. DOI: 10.1007/S10817-010-9188-8 (cited on page 70).

[57] David Baelde, Stéphanie Delaune, Ivan Gazeau, and Steve Kremer. "Symbolic Verification of Privacy-Type Properties for Security Protocols with XOR". In: *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*. IEEE Computer Society, 2017, pp. 234–248. DOI: 10.1109/CSF.2017.22 (cited on page 70).

[58] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott, eds. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*. Vol. 4350. Lecture Notes in Computer Science. Springer, 2007 (cited on page 70).

[59] Elise Klein, Jannik Dreier, and Steve Kremer. *Source code of the presented version of Tamarin and case studies models*. https://github.com/EliseKlein/ac-tamarin-experiments. 2025 (cited on pages 83, 89).

[60] Tatsuaki Okamoto. "An electronic voting scheme". In: *Advanced IT Tools: IFIP World Conference on IT Tools 2–6 September 1996, Canberra, Australia*. Ed. by Nobuyoshi Terashima and Edward Altman. Boston, MA: Springer US, 1996, pp. 21–30. DOI: 10.1007/978-0-387-34979-4_3 (cited on page 87).

[61] Jannik Dreier, Pascal Lafourcade, and Dhekra Mahmoud. "Shaken, not Stirred -Automated Discovery of Subtle Attacks on Protocols using Mix-Nets". In: *Proceedings of the 33rd USENIX Conference on Security Symposium*. Philadelphia, United States, Aug. 2024 (cited on pages 88, 89).

[62] Véronique Cortier, Stéphanie Delaune, Jannik Dreier, and Elise Klein. "Automatic generation of sources lemmas in TAMARIN: towards automatic proofs of security protocols". In: *Journal of Computer Security* 30.4 (Aug. 2022), pp. 573–598. DOI: 10.3233/JCS-210053 (cited on page 89).

[63] Vincent Cheval and Caroline Fontaine. "Automatic verification of Finite Variant Property beyond convergent equational theories". In: *2025 IEEE 38th Computer Security Foundations Symposium (CSF 2025)*. IEEE. Santa Cruz, United States: IEEE, July 2025 (cited on page 93).

[64] Siva Anantharaman, Hai Lin, Christopher Lynch, Paliath Narendran, and Michael Rusinowitch. "Unification modulo Homomorphic Encryption". In: *Journal of Automated Reasoning* 48.2 (Feb. 2012), pp. 135–158. DOI: 10.1007/s10817-010-9205-y (cited on page 93).

[65] Shymaa M. Arafat. *On the Estonian Internet Voting System, IVXV, SoK and Suggestions*. Cryptology ePrint Archive, Paper 2025/506. 2025. URL: https://eprint.iacr.org/2025/506 (cited on page 93).

[66] P. Y. A. Ryan and S. A. Schneider. "Prêt à Voter with Re-encryption Mixes". In: *Computer Security – ESORICS 2006*. Ed. by Dieter Gollmann, Jan Meier, and Andrei Sabelfeld. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 313–326 (cited on page 93).

[67] Ari Juels, Dario Catalano, and Markus Jakobsson. "Coercion-resistant electronic elections". In: *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*. WPES '05. Alexandria, VA, USA: Association for Computing Machinery, 2005, 61–70. DOI: 10.1145/1102199.1102213 (cited on page 93).

[68] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. "Civitas: Toward a Secure Voting System". In: *2008 IEEE Symposium on Security and Privacy (sp 2008)*. 2008, pp. 354–368. DOI: 10.1109/SP.2008.32 (cited on page 93).

[69] Véronique Cortier, Stéphanie Delaune, and Pascal Lafourcade. "A Survey of Algebraic Properties Used in Cryptographic Protocols". In: *Journal of Computer Security* 14.1 (2006), pp. 1–43 (cited on page 93).

Appendix

Appendix for formal analysis background

A.

A.1. Model in SAPIC$^+$ of $CRXOR$ protocol

```
theory CRXOR

builtins: xor, hashing

let Initiator(~k) =
  new ~na;
  event Challenge(~na);
  out(c,~na);
  in(c,<m,nb>);
  let n = ~na ⊕ ~k ⊕ nb in
  if h(n) = m then
    event Alive(~na, ~k)

let Responder(~k) =
  in(c,na);
  new ~nb;
  let n = na ⊕ ~k ⊕ nb in
  if na = zero then 0
  else (
    event NeqForExe(na,zero);
    event Response(~k);
    out(c,<h(n), ~nb>)
  )

process:
  new ~k; ! ( Initiator(~k) || Responder(~k) )

lemma alive:
  All x y #i. Alive(x,y)@#i ==> Ex #j. Response(y)@#j

lemma recentalive_tag:
  exists-trace
    not (All x y #i. Alive(x,y)@#i ==>
         Ex #j #k. Challenge(x)@#k & Response(y)@#j
         & #k < #j & #j < #i)

lemma executable:
  exists-trace
    Ex x y #i #j.
      Alive(x,y)@#i & Response(y)@#j
      & (not Ex #k. Response(y)@#k & (not #j = #k))
      & (not Ex #l. NeqForExe(x,x)@#l)

end
```

A.2. Model in ProVerif of $CRXOR$ protocol

```
free c:channel.
fun fst(bitstring):bitstring.
fun h(bitstring):bitstring.
fun pair(bitstring,bitstring):bitstring.
fun snd(bitstring):bitstring.
fun xor(bitstring,bitstring):bitstring.
fun zero():bitstring.
event eAlive(bitstring,bitstring).
event eChallenge(bitstring).
event eNeqForExe(bitstring,bitstring).
event eResponse(bitstring).
equation forall x_1:bitstring, x_2:bitstring;   fst((x_1, x_2)) = x_1.
equation forall x_1:bitstring, x_2:bitstring;   snd((x_1, x_2)) = x_2.

(*alive*)
query x:bitstring, y:bitstring, i:time, j:time;
 (event(eAlive(x,y))@i) ==> (event(eResponse(y))@j)
.

(*executable*)
query x:bitstring, y:bitstring, i:time, j:time, k:time, l:time;
     (event(eAlive(x,y))@i) && (event(eResponse(y))@j)
  && not((event(eResponse(y))@k) && (j <> k))
  && not(event(eNeqForExe(x,x))@l)
.

let Initiator(k:bitstring)=
    new na:bitstring;
    event eChallenge(na);
    out(c,na);
    in(c,(m:bitstring, nb:bitstring));
    let (=h(xor(k,xor(na,nb))))=m in
        (event eAlive(na,k)).

let Responder(k:bitstring)=
    in(c,na:bitstring);
    new nb:bitstring;
    let (=na)=zero in
        (0)
    else(event eNeqForExe(na,zero);
         event eResponse(k);
         out(c,(h(xor(k,xor(nb,na))),nb))).

process
    new k:bitstring;
    ! (Initiator(k) | Responder(k))
```

A.3. Model in Tamarin of $CRXOR$ protocol

```
theory CRXOR
begin

builtins: xor,hashing

rule Setup:
  [ Fr(~k) ]
--[ OnlyOnce() ]->
  [ !InitLTK(~k), !RespLTK(~k) ]

rule initiator:
  [ Fr(~na) ]
--[ Challenge(~na) ]->
  [ Out(~na), Initiated(~na) ]

rule responder:
  [ In(na), Fr(~nb), !RespLTK(~k) ]
--[ Response(~k)
  , Neq(na,zero)
  , NeqForExe(na,zero) ]->
  [ Out(<h(na ⊕ ~k ⊕ ~nb), ~nb>) ]

rule initiator2:
  [ Initiated(~na), !InitLTK(~k), In(<m,nb>) ]
--[ Alive(~na,~k), Eq(h(~na ⊕ ~k ⊕ nb),m) ]->
  []



restriction equality:
  "All x y #i. Eq(x,y)@i ==> x=y"

restriction unique:
  "All #i #j. OnlyOnce() @i & OnlyOnce() @j ==> #i = #j"

restriction inequality:
  "not (Ex x #i. Neq(x,x)@i)"


lemma alive:
  All x y #i. Alive(x,y)@#i ==> Ex #j. Response(y)@#j

lemma recentalive_tag:
  exists-trace
    not (All x y #i. Alive(x,y)@#i ==>
         Ex #j #k. Challenge(x)@#k & Response(y)@#j
         & #k < #j & #j < #i)

lemma executable:
  exists-trace
    Ex x y #i #j.
      Alive(x,y)@#i & Response(y)@#j
      & (not Ex #k. Response(y)@#k & (not #j = #k))
      & (not Ex #l. NeqForExe(x,x)@#l)

end
```

Appendix for EDHOC analysis B.

B.1. Detailed analysis results

The following tables (Tables B.2 and B.3) present the detailed results of our analysis, along with the required verification times. The results are presented per lemma (expressing the security properties, as explained in Section 3.3.1). The four others columns correspond to the protocol version that is analysed. The results in Draft 12 and Draft 12 KEM correspond to the version of the draft we originally analysed while the results in the Draft 14 and Draft 14 KEM correspond to the new versions of the protocol including mitigations we suggested.

For each lemma, the results are presented as follow:

- The scenario: a summary of the attacker capabilities and optional checks that are considered in this specific run. The explanation for the abreviations used can be found in Table 3.3 and at the bottom of each table;
- The result of the verification: it can be an attack ($\times$) or a proof (using ProVerif: $\checkmark^P$, or using Tamarin: $\checkmark^T$);
- The verification time.

Due to lack of space, we omit the results for secretR which are similar to secretI, for auth-RI-unique which are similar to auth-IR-unique, as well as no-reflection-attacks-RI which instantly holds when we add the Cred-Check. The full and latest results over the draft can be found in [40].

[40]: Jacomme et al. (2022), *LAKE EDHOC models*

The results on anonymity are presented in a separated table (Table B.1) indicating the precise protocol, whether the Cred-Check test is performed and whether we test the property for a bounded or unbounded number of sessions. $\checkmark^D$ and $\times^D$ indicate that the DeepSec tool was used.

Table B.1.: Summary of our results for anonymity

	Draft 12	**Draft 14**
KEM w/ Cred-Check , bounded sessions	$\times^D$ (3s)	$\times^D$ (3s)
KEM w/o Cred-Check , bounded sessions	$\checkmark^D$ (3s)	$\checkmark^D$ (3s)
KEM w/o Cred-Check, unbounded sessions	$\checkmark^P$ (10m)	$\checkmark^P$ (6m)
DH w/o Cred-Check, unbounded sessions, only method 0	$\checkmark^P$ (267m)	$\checkmark^P$ (148m)

Table B.2.: Detailed results
Attackers capabilities: Sig$^{\not\zeta}$: precise signatures (for attack finding), Sig$^{\not\zeta}$-proof: precise signature (stronger guarantees), DH$^{\not\zeta}$: precise DH with small subrougps, AEAD$^{\not\zeta}$: precise AEAD model, $\oplus^{\not\zeta}$: malleable xor encryption, SessKey$^{\not\zeta}$: compromise of session keys, DHShare$^{\not\zeta}$: compromise ephemeral shares
Protocol optional checks: DH-Check: neutral DH element check, Cred-Check: own identity check

Lemma	Draft 12	Draft 12 KEM	Draft 14	Draft 14 KEM
secretI	DHShare$^{\not\zeta}$, Sig$^{\not\zeta}$, SessKey$^{\not\zeta}$ ✓^{P} (1076s)	DHShare$^{\not\zeta}$, SessKey$^{\not\zeta}$, $\oplus^{\not\zeta}$, AEAD$^{\not\zeta}$, Sig$^{\not\zeta}$-proof ✓^{P} (316s)	DHShare$^{\not\zeta}$, Sig$^{\not\zeta}$, SessKey$^{\not\zeta}$ ✓^{P} (1833s)	Sig$^{\not\zeta}$-proof, DHShare$^{\not\zeta}$, SessKey$^{\not\zeta}$, $\oplus^{\not\zeta}$, AEAD$^{\not\zeta}$ ✓^{P} (259s)
	$\oplus^{\not\zeta}$ ✓^{P} (104s)		$\oplus^{\not\zeta}$ ✓^{P} (278s)	
	AEAD$^{\not\zeta}$, Sig$^{\not\zeta}$-proof, DH$^{\not\zeta}$ ✓^{P} (68s)		AEAD$^{\not\zeta}$, Sig$^{\not\zeta}$-proof, DH$^{\not\zeta}$ ✓^{P} (143s)	
	SessKey$^{\not\zeta}$, Sig$^{\not\zeta}$-proof, DH$^{\not\zeta}$ ✓^{P} (78s)		SessKey$^{\not\zeta}$, Sig$^{\not\zeta}$-proof, DH$^{\not\zeta}$ ✓^{P} (190s)	
			- ✓^{T} (426m)	
auth-IR-unique	- × (49s)	- × (12s)	- ✓^{P} (51s) ✓^{T} (30m)	- ✓^{P} (11s)
	DH-Check, Sig$^{\not\zeta}$, SessKey$^{\not\zeta}$ ✓^{P} (98s)		DH-Check, Sig$^{\not\zeta}$, SessKey$^{\not\zeta}$ ✓^{P} (67s) ✓^{T} (30m)	
	DH-Check, AEAD$^{\not\zeta}$, SessKey$^{\not\zeta}$ ✓^{P} (48s)		DH-Check, AEAD$^{\not\zeta}$, SessKey$^{\not\zeta}$ ✓^{P} (52s) ✓^{T} (114m)	
	DH-Check, $\oplus^{\not\zeta}$, SessKey$^{\not\zeta}$ ✓^{P} (104s)		DH-Check, $\oplus^{\not\zeta}$, SessKey$^{\not\zeta}$ ✓^{P} (107s)	
data-authentication-IR	DHShare$^{\not\zeta}$, Sig$^{\not\zeta}$ × (116s)	DHShare$^{\not\zeta}$, Sig$^{\not\zeta}$ ✓^{P} (20s)	DHShare$^{\not\zeta}$, Sig$^{\not\zeta}$ × (125s)	DHShare$^{\not\zeta}$, Sig$^{\not\zeta}$ ✓^{P} (21s)
	$\oplus^{\not\zeta}$ ✓^{P} (73s)	$\oplus^{\not\zeta}$ ✓^{P} (13s)	$\oplus^{\not\zeta}$ ✓^{P} (75s)	$\oplus^{\not\zeta}$ ✓^{P} (14s)
	SessKey$^{\not\zeta}$ Sig$^{\not\zeta}$-proof, DH$^{\not\zeta}$ ✓^{P} (66s)	SessKey$^{\not\zeta}$, AEAD$^{\not\zeta}$ Sig$^{\not\zeta}$-proof ✓^{P} (12s)	SessKey$^{\not\zeta}$ Sig$^{\not\zeta}$-proof, DH$^{\not\zeta}$ ✓^{P} (89s) ✓^{T} (1110m)	SessKey$^{\not\zeta}$, AEAD$^{\not\zeta}$ Sig$^{\not\zeta}$-proof ✓^{P} (14s)
	AEAD$^{\not\zeta}$ Sig$^{\not\zeta}$-proof, DH$^{\not\zeta}$ ✓^{P} (65s)		AEAD$^{\not\zeta}$ Sig$^{\not\zeta}$-proof, DH$^{\not\zeta}$ ✓^{P} (64s)	

Table B.3.: Detailed results
Attackers capabilities: $\text{Sig}^{\lightning}$: precise signatures (for attack finding), $\text{Sig}^{\lightning}$-proof: precise signature (stronger guarantees), $\text{DH}^{\lightning}$: precise DH with small subrougps, $\text{AEAD}^{\lightning}$: precise AEAD model, $\oplus^{\lightning}$: malleable xor encryption, $\text{SessKey}^{\lightning}$: compromise of session keys, $\text{DHShare}^{\lightning}$: compromise ephemeral shares
Protocol optional checks: DH-Check: neutral DH element check, Cred-Check: own identity check

Lemma	Draft 12	Draft 12 KEM	Draft 14	Draft 14 KEM
data-authentication-RI	$\text{AEAD}^{\lightning}$ × (56s)	$\text{AEAD}^{\lightning}$ × (13s)	$\text{AEAD}^{\lightning}$ $\checkmark^{P}$ (60s) $\checkmark^{T}$ (546m)	$\text{AEAD}^{\lightning}$ $\checkmark^{P}$ (14s)
	$\text{DHShare}^{\lightning}$, $\text{Sig}^{\lightning}$ × (863s)	$\text{DHShare}^{\lightning}$, $\text{Sig}^{\lightning}$ × (27s)	$\text{DHShare}^{\lightning}$, $\text{Sig}^{\lightning}$ × (892s)	$\text{DHShare}^{\lightning}$, $\text{Sig}^{\lightning}$ × (28s)
	$\text{SessKey}^{\lightning}$, $\text{Sig}^{\lightning}$ × (180s)	$\text{SessKey}^{\lightning}$ $\text{Sig}^{\lightning}$-proof × (15s)	$\text{SessKey}^{\lightning}$, $\text{Sig}^{\lightning}$ $\checkmark^{P}$ (90s) $\checkmark^{T}$ (343m)	$\text{SessKey}^{\lightning}$ $\text{Sig}^{\lightning}$-proof $\checkmark^{P}$ (15s)
	$\oplus^{\lightning}$ $\checkmark^{P}$ (160s)	$\oplus^{\lightning}$ $\checkmark^{P}$ (17s)	$\oplus^{\lightning}$ $\checkmark^{P}$ (187s)	$\oplus^{\lightning}$ $\text{Sig}^{\lightning}$-proof $\checkmark^{P}$ (30s)
honest-auth-RI-non-inj	$\text{DHShare}^{\lightning}$, $\text{Sig}^{\lightning}$, $\text{SessKey}^{\lightning}$ × (846s)	$\text{DHShare}^{\lightning}$, $\text{AEAD}^{\lightning}$, $\oplus^{\lightning}$, $\text{SessKey}^{\lightning}$ $\text{Sig}^{\lightning}$-proof $\checkmark^{P}$ (504s)	$\text{DHShare}^{\lightning}$, $\text{Sig}^{\lightning}$, $\text{SessKey}^{\lightning}$ $\checkmark^{P}$ (780s) $\checkmark^{T}$ (391m)	$\text{DHShare}^{\lightning}$, $\text{AEAD}^{\lightning}$, $\oplus^{\lightning}$, $\text{SessKey}^{\lightning}$ $\text{Sig}^{\lightning}$-proof $\checkmark^{P}$ (69s)
	$\text{SessKey}^{\lightning}$ $\text{Sig}^{\lightning}$-proof, $\text{DH}^{\lightning}$ $\checkmark^{P}$ (91s)		$\text{SessKey}^{\lightning}$ $\text{Sig}^{\lightning}$-proof, $\text{DH}^{\lightning}$ $\checkmark^{P}$ (108s) $\checkmark^{T}$ (846m)	
	$\text{AEAD}^{\lightning}$ $\text{Sig}^{\lightning}$-proof, $\text{DH}^{\lightning}$ $\checkmark^{P}$ (84s)		$\text{AEAD}^{\lightning}$ $\text{Sig}^{\lightning}$-proof, $\text{DH}^{\lightning}$ $\checkmark^{P}$ (90s)	
	$\oplus^{\lightning}$ $\checkmark^{P}$ (104s)		$\oplus^{\lightning}$ $\checkmark^{P}$ (139s)	
repudiation-soundness	- × (81s)	- × (103s)	- $\checkmark^{P}$ (34s)	- $\checkmark^{P}$ (15s)
	DH-Check, $\text{Sig}^{\lightning}$ × (125s)		DH-Check, $\text{Sig}^{\lightning}$ × (36s)	
	DH-Check × (74s)		DH-Check $\checkmark^{P}$ (25s)	
	DH-Check, $\text{AEAD}^{\lightning}$ × (86s)		DH-Check, $\text{AEAD}^{\lightning}$ $\checkmark^{P}$ (28s)	

Appendix for user-defined AC symbols in Tamarin

C.

C.1. Proof of Theorem 5.2.1

We prove the main correctness Theorem 5.2.1 by adapting the proofs from [49] and [50].

[49]: Dreier et al. (2017), *Beyond Subterm-Convergent Equational Theories in Automated Verification of Stateful Protocols (extended version)*

[50]: Dreier et al. (2020), *Verification of Stateful Cryptographic Protocols with Exclusive OR*

Proof of Theorem 5.2.1. Given Lemmas 4.2.1 and 4.2.2, it remains to be proven that for all sets P of protocol rules,

$$\{\overline{trace(dg)} \mid dg \in dgraphs(\lceil P \cup MD \rceil^{\mathscr{C}}_{insts}) \wedge dg \downarrow_{\mathscr{C}} \text{-normal}\} =_{\mathscr{AC}} \overline{trace(ndgraphs(P))}.$$

We prove both inclusions by induction on graphs. The normal form conditions **N14** does not affect the majority of cases. Hence we can treat them as in [50].
$\subseteq_{\mathscr{AC}}$: We prove that, for all $dg \in dgraphs(\lceil P \cup MD \rceil^{\mathscr{C}}_{insts})$ with $dg \downarrow_{\mathscr{C}}$ -normal, there is $ndg \in ndgraphs(P)$ such that:

$$\mathtt{known}(dg) \subseteq_{\mathscr{AC}} \mathtt{known}^{\updownarrow}(ndg) \tag{C.1}$$

$$\mathtt{stfacts}(dg) \subseteq_{\mathscr{AC}} \mathtt{stfacts}(ndg) \tag{C.2}$$

$$\mathtt{created}(dg) =_{\mathscr{AC}} \mathtt{created}(ndg) \tag{C.3}$$

$$\overline{trace(dg)} =_{\mathscr{AC}} \overline{trace(ndg)} \tag{C.4}$$

Notice that the inclusion of (2) applies to multisets.
The four properties hold for $dg = ([], \emptyset)$. Let $dg = (I, D) \in dgraphs(\lceil P \cup MD \rceil^{\mathscr{C}}_{insts})$ with $dg \downarrow_{\mathscr{C}}$ -normal, and $ndg = (\tilde{I}, \tilde{D}) \in ndgraphs(P)$ such that (1)-(4) hold. Let $ri \in ginst(\lceil P \cup MD \rceil^{\mathscr{C}}_{insts} \cup \{Fresh\})$ such that $dg' = (I \cdot ri, D \uplus D') \in dgraphs(\lceil P \cup MD \rceil^{\mathscr{C}}_{insts})$. We must show that there is a deduction extension ndg' of ndg that satisfies (1)-(4) with respect to dg'.
We perform a case distinction on ri:

- for the following cases, we perform the same proof as in [50]:
  - $ri \in ginst(\lceil P \rceil^{\mathscr{C}}_{insts} \cup \{Fresh\})$
  - rule instances ri of the form

$$ri \in \{[Out(m)] -[]\rightarrow [K(m)], \quad [K(m)] -[K(m)]\rightarrow [In(m)], \quad [Fr(n)] -[]\rightarrow [K(n)], \quad [] -[]\rightarrow [K(c)]\} \subset ginst(\lceil MD \rceil^{\mathscr{C}}_{insts})$$

  - rule instances of the form

$$[K(m_1), \ldots, K(m_n)] -[]\rightarrow [K(f(m_1, \ldots, m_n))] \in ginst(\lceil MD \rceil^{\mathscr{C}}_{insts})$$

  - ri in bilinear-pairing, Diffie-Hellman or XOR theory

- for the rules $[K(m_1), \ldots, K(m_k)] -[]\rightarrow [K(m)]$ with $m = f(m_1, \ldots, m_k)\downarrow_{\mathscr{E}}$ where $f \in \Sigma_{Op}$ and $m \neq_{AC} f(m_1, \cdots, m_k)$, the proof is similar as in [49]. If in the extension ndg' the added rule would violate **N14**, this means that ndg' ends with two chained instances, r_a and r_b, of deconstruction rules in $\mathscr{RV}_f$ and f has the NDC property. By definition of the property, there exists a deduction extension de of ndg such that $v \in \mathtt{known}^{\updownarrow}(de)$ and de contains no chain of repeated instantiations of deconstruction rules for f, so we can use de instead of ndg'.

$\supseteq_{\mathscr{AC}}$: We show that for all $ndg \in ndgraphs(P)$, there is a $dg \in dgraphs(\lceil P \cup MD\rceil^{\mathscr{E}}_{insts})$ with $dg\downarrow_{\mathscr{E}}$ -normal, such that:

$$\mathtt{known}(ndg) \subseteq_{\mathscr{AC}} \mathtt{known}^{\updownarrow}(dg) \tag{C.1}$$

$$\mathtt{stfacts}(ndg) \subseteq_{\mathscr{AC}} \mathtt{stfacts}(dg) \tag{C.2}$$

$$\mathtt{created}(ndg) =_{\mathscr{AC}} \mathtt{created}(dg) \tag{C.3}$$

$$\overline{trace(ndg)} =_{\mathscr{AC}} \overline{trace(dg)} \tag{C.4}$$

It holds for $ndg = ([], \emptyset)$. Let $ndg = (I, D) \in ndgraphs(P)$, and $dg = (\tilde{I}, \tilde{D}) \in dgraphs(\lceil P \cup MD\rceil^{\mathscr{E}}_{insts})$ with $dg\downarrow_{\mathscr{E}}$ -normal such that (1)-(4) hold. Let $ri \in ginst(\lceil P\rceil^{\mathscr{E}}_{insts} \cup ND \cup \{Fresh\})$ such that $ndg' = (I \cdot ri, D \uplus D') \in ndgraphs(P)$. We must show that there is a deduction extension dg' of dg that satisfies (1)-(4) with respect to ndg'.
Note that there is no normal dependency graph violating **N14** by definition, so no special care needs to be taken for that restriction in this direction.
We still perform a case distinction on ri and all the cases are proven as in [50].

□

C.2. Modified Constraint-Solving Rules

C.2.1. Extending $\rightsquigarrow_p$ with N14

We added one constraint-solving rule that implements **N14**, as shown in Figure C.1. The proof of soundness and completeness of the constraint solving is then a straightforward extension of the proofs of [48, Lemma 6] and [18, Theorem 2]: It is easy to see that this rule does not create any new solutions, and it only removes solutions violating **N14** as we only remove systems containing two chained instances of deconstruction rules for a function symbol with the NDC property.

N14 :	$\Gamma \rightsquigarrow_p \bot$
	if $i : ri, j : ri', (i,1) \dashrightarrow (j,1)$, ri and ri' are instances of deconstruction rules for a function symbol with the NDC property

Figure C.1.: New constraint-reduction rule for **N14**.

C.2.2. Extending $\rightsquigarrow_P$ with $\mathbf{DG}_{+,\perp}$

Flattening relation $\rightarrow_{\mathsf{f},+}$

We can define a flattening relation $\rightarrow_{\mathsf{f},+}$ which performs one flattening step as follows: $(I, D) \rightarrow_{\mathsf{f},+} (I', D')$ for an AC symbol $+$, and

- $I = [l_1 \dashv\!\!\!

3. $\mathbf{S}_{\approx}$ which replaces a constraint system Γ by $\{\Gamma\sigma \mid \sigma \in unif_{\mathcal{AC}}(x^1 + x^2 =^? \sum_i u_i)\}$.

Hence, there must exist $j < k < \ell < i$ such that

- $\Gamma^j \in \Omega_j$ such that $\Gamma^j \rightsquigarrow_P^* \Gamma^i$ and constraints $n_a : (\mathsf{K}^\uparrow(x^1), \mathsf{K}^\uparrow(x^2) \dashv\vdash \mathsf{K}^\uparrow(x^1 + x^2))$ and $(i, 1) \rightarrowtail p$ for some open $\mathsf{K}^\uparrow(\sum u_i)$-premise p and fresh n_a have been added by rule **DG1$_2$**;
- $\Gamma^k \in \Omega_k$ such that $\Gamma^j \rightsquigarrow_P^* \Gamma^k \rightsquigarrow_P^* \Gamma^i$ and the constraint $\mathsf{K}^\uparrow(x^1 + x^2) \approx \mathsf{K}^\uparrow(\sum u_i)$ has been added by rule **DG1$_2$**;
- $\Gamma^{\ell_1} \in \Omega_\ell$ such that $\Gamma^j \rightsquigarrow_P^* \Gamma^k \rightsquigarrow_P^* \Gamma^{\ell_1} \rightsquigarrow_P^* \Gamma^i$ obtained by applying $\sigma_1 \in unif_{\mathcal{AC}}(x^1 + x^2 =^? \sum u_i)$.

However, by Lemma 5.4.3 there exists $\sigma_2 \in unif_{\mathcal{AC}}(x^1 + x^2 =^? \sum u_i)$ which shifts variables that will end up in the trivial premise p_1 into $x^2\sigma_2$. Repeatedly applying Lemma 5.4.3 we can build a + constructor chain such that all non variable terms remain in the same nodes, but pushing the variables of p_1 into p_2, ending up with a single trivial premise $p_3 = \mathsf{K}^\uparrow(\sum_i^{n^3} x_i^2)$. This is illustrated in Figure C.2.
Importantly, the number of variables in p_3 is less or equal to the number of variables in p_1 and p_2, i.e., $n^3 \leq n^1 + n^2$. Therefore, from the solution of (dg, θ) for Γ^i we can built a solution (dg', θ') of Γ_2^ℓ such that $dg \downarrow_{\mathsf{f}} = dg' \downarrow_{\mathsf{f}}$.
Indeed, (dg', θ') are as (dg, θ) except that we

- remove node n_1, and any edge pointing to n_1 will point directly to n_1's successor.
- suppose that θ instantiates the trivial premises p_1 and p_2 to $\mathsf{K}^\uparrow(t_1)$ and $\mathsf{K}^\uparrow(t_2)$. As p_3 has less variables than p_1 and p_2 it is possible to instantiate p_3 to $\mathsf{K}^\uparrow(t_1 + t_2)$.

As we only modify the + constructor chained we directly have that $dg \downarrow_{\mathsf{f},+} = dg' \downarrow_{\mathsf{f},+}$.

□

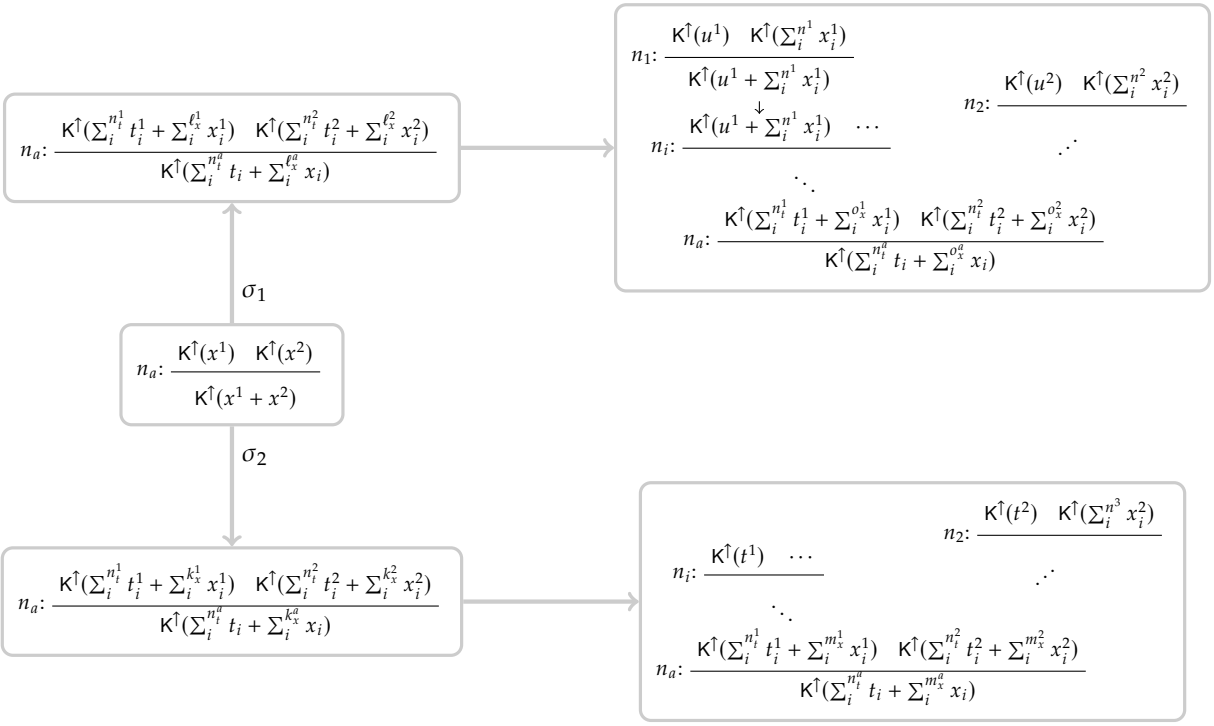


Figure C.2.: Application of different unifiers leading to the same flattened P solution.

Résumé en français D.

D.1. Contexte : Protocole & Sécurité

D.1.1. L'énigme des messagers cleptomanes

Pour illustrer les différents concepts abordés dans cette thèse, considérons la situation suivante : la princesse Alice et le prince Bob vivent dans un monde imaginaire où la correspondance ne peut se faire que par courrier. Les messagers chargés de transmettre lettres et colis sont tous très indiscrets : ils désirent connaître le contenu de tout ce qui leur est confié, et, si cela sert leurs intérêts, ils n'hésitent pas à le garder pour eux. Alice et Bob, vivant dans des régions éloignées, sont donc contraints de faire appel à ces messagers cleptomanes pour échanger. Toutefois, ils n'acceptent pas que des inconnus puissent lire leurs mots doux ou voler leurs présents de valeur. Ils ont donc besoin d'une stratégie pour éviter un tel scénario, sachant qu'ils possèdent chacun un cadenas et la clé correspondante...

Cette petite énigme logique est un énoncé classique : pour résoudre le problème, Alice enferme sa jolie lettre dans une boîte qu'elle verrouille avec son cadenas et confie le tout à un messager. Ce dernier, ne pouvant ouvrir la boîte, la remet à Bob. Celui-ci, également dans l'incapacité d'ouvrir la boîte, ajoute son propre cadenas et renvoie l'ensemble à Alice. Cette dernière peut donc retirer le cadenas dont elle a la clé et renvoyer la boîte à Bob. Il ne reste alors que le cadenas de Bob sur la boîte, et ce dernier peut l'ouvrir pour lire le merveilleux message d'Alice !

Cette combine entre les deux amoureux peut être vue comme un **protocole**. Ils se sont mis d'accord au préalable sur les étapes à suivre, illustrée dans la Figure D.1 :

1. Alice place son message dans une boîte, qu'elle ferme avec son cadenas, et l'envoie à Bob.
2. Bob reçoit une boîte avec un cadenas. Il ajoute le sien et renvoie l'ensemble à Alice.
3. Alice reçoit une boîte avec deux cadenas. Elle retire le sien et renvoie la boîte à Bob.
4. Bob enlève son cadenas et lit le contenu.

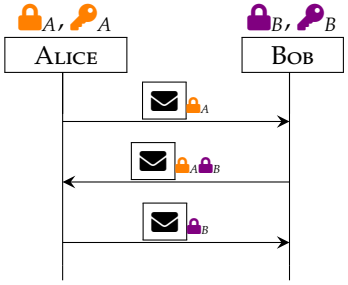


Figure D.1.: Protocole utilisé par Alice et Bob

Cette suite d'actions repose sur plusieurs postulats implicites : Alice et Bob possèdent chacun un cadenas unique, et donc une clé distincte; seule la clé correspondant au cadenas peut l'ouvrir; les cadenas ne peuvent être forcés ou détruits; les clés ne peuvent être reproduites sans avoir été vues, etc.
Ajoutons trois hypothèses supplémentaires : (1) le protocole établi par le couple n'est pas secret, il est connu de tous (d'autres personnes pourraient souhaiter protéger leur courrier !); (2) chaque individu dans le royaume possède un couple clé-cadenas; (3) Alice et Bob ne peuvent reconnaître que leur propre cadenas (ils ne sauraient être sûrs de l'identité du propriétaire de l'autre cadenas).

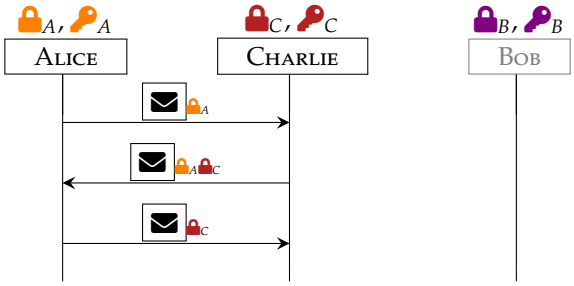


Figure D.2.: Attaque réalisée par Charlie

Avec ces nouvelles hypothèses, un messager particulièrement rusé, que nous appellerons Charlie, réussit à monter l'attaque suivante en utilisant sa connaissance du protocole[1]. Lors d'un échange entre Alice et Bob, il agit comme suit, comme illustrée dans la Figure D.2 :

1. Charlie attend qu'Alice lui remette la boîte cadenassée à destination de Bob.
2. Plutôt que de l'amener à Bob, il garde la boîte, y ajoute son propre cadenas[2] et la remet à Alice.
3. Alice reçoit une boîte avec deux cadenas[3]. Elle retire le sien et le redonne à Charlie pour qu'il l'amène à Bob.
4. Charlie conserve la boîte : il ne reste plus que son propre cadenas, il peut donc l'ouvrir et lire ou voler le contenu.

1: hypothèse (1)

2: Grâce à l'hypothèse (2), nous savons que Charlie possède son propre couple clé-cadenas.

3: L'hypothèse (3) nous dit qu'Alice reconnaît son cadenas, mais ne peut pas identifier le second.

Alice a scrupuleusement suivi le protocole, mais Bob n'a jamais eu la boîte entre les mains : Charlie a trouvé une **attaque**. Il y avait un problème d'**authentification du destinataire**, car Alice n'avait aucun moyen de s'assurer qu'elle envoyait réellement son colis à Bob, ce qui a également compromis la **confidentialité** du contenu, puisque Charlie a pu y accéder.

D.1.2. Protocoles cryptographiques

Comme décrit de manière intuitive dans Appendix D.1.1, un protocole est un ensemble de règles décrivant comment plusieurs entités s'échangent des messages, et comment elles protègent cet échange grâce à des *primitives cryptographiques*.

Dans un contexte contemporain et concret, les protocoles interviennent à de nombreux niveaux de notre vie quotidienne. Le protocole EMV (Europay Mastercard Visa) [1, 2] permet la communication entre la banque émettrice d'une carte de crédit et un terminal de paiement afin d'autoriser ou non une transaction. Les applications de messagerie, telles que Signal [3], reposent sur des protocoles pour définir les règles de transmission des messages entre plusieurs utilisateurs via Internet. De même, le protocole TLS 1.3 [4] et les protocoles d'authentification du standard 5G-AKA (Authentication and Key Agreement) [5, 6] sont essentiels au bon fonctionnement des réseaux mobiles et de l'accès sécurisé à Internet. Les protocoles peuvent même intervenir dans notre vie démocratique, notamment à travers les systèmes de vote électronique comme Belenios [7] ou celui de la Swiss Post [8].

Ces protocoles remplissent des fonctions critiques dans nos sociétés et doivent offrir des garanties solides. Par exemple, un protocole de paiement par carte bancaire doit assurer que le montant prélevé et le

[1]: Ruiter et al. (2012), *Formal Analysis of the EMV Protocol Suite*
[2]: Basin et al. (2021), *The EMV Standard: Break, Fix, Verify*
[3]: Katriel et al. (2020), *A Formal Security Analysis of the Signal Messaging Protocol*
[4]: Cremers et al. (2017), *A Comprehensive Symbolic Analysis of TLS 1.3*
[5]: Koutsos (2019), *The 5G-AKA Authentication Protocol Privacy*
[6]: Basin et al. (2018), *A Formal Analysis of 5G Authentication*
[7]: Cortier et al. (2019), *Belenios: a simple private and verifiable electronic voting system*
[8]: Cortier et al. (2022), *A privacy attack on the Swiss Post e-voting system*

compte bénéficiaire ne puissent être altérés au cours de l'opération. Un protocole de messagerie instantanée doit garantir la confidentialité des échanges, tandis qu'un protocole de vote électronique doit assurer le secret du vote ainsi que sa correcte prise en compte. Ce type de garanties de sécurité repose sur l'utilisation de **primitives cryptographiques**.

Lorsque ces garanties ne sont pas rigoureusement assurées, des vulnérabilités peuvent être exploitées par des acteurs malveillants, avec des conséquences potentiellement graves. Le protocole EMV, par exemple, a fait l'objet de nombreuses attaques [2, 9, 10], dont l'une permet de contourner la vérification du code PIN à l'aide d'une simple application mobile, rendant possible des paiements sans connaître le code de la carte. Dans un tout autre domaine, des failles ont été découvertes dans le système de vote électronique utilisé pour les élections législatives françaises de 2022, compromettant potentiellement l'intégrité du scrutin [11]. Il est donc essentiel de concevoir rigoureusement les protocoles et de les analyser en profondeur afin de détecter au plus tôt d'éventuelles failles.

[2]: Basin et al. (2021), *The EMV Standard: Break, Fix, Verify*
[9]: Ferradi et al. (2016), *When organized crime applies academic results: a forensic analysis of an in-card listening device*
[10]: Basin et al. (2023), *Inducing Authentication Failures to Bypass Credit Card PINs*

[11]: Debant et al. (2023), *Reversing, Breaking, and Fixing the French Legislative Election E-Voting Protocol*

Lors de la conception d'un protocole, plusieurs aspects doivent être soigneusement étudiés, chacun nécessitant des compétences spécifiques. En voici quelques exemples :

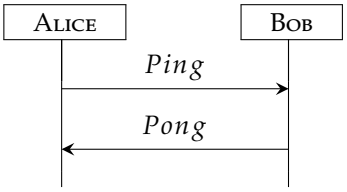
- **La cryptographie** : Quels algorithmes de chiffrement utiliser ? Quelles fonctions de hachage ? Quels types de signatures ? En d'autres termes, quels outils mathématiques sont les plus adaptés au domaine d'application du protocole ?
- **La spécification** : Quelle est la séquence de messages échangée entre les participants (combien de messages, quelles informations, chiffrées ou non) ? Combien d'acteurs sont impliqués, et quels sont leurs rôles ? À ce stade, la cryptographie est considérée comme idéale, et l'accent est mis sur la logique d'interaction entre les entités.
- **L'implémentation** : Comment le protocole est-il implémenté ? Dans quel langage ? Sur quel type d'appareil ou de système ? Avec quelles contraintes de performance ? L'implémentation suit-elle fidèlement la spécification ou nécessite-t-elle des adaptations, au risque de compromettre certaines garanties de sécurité de la spécification ?

Chacun de ces aspects est crucial. Cette thèse se concentre principalement sur les questions de spécification, c'est-à-dire sur la logique même du protocole.

Pour analyser un protocole, il est nécessaire de commencer par le décrire. Il existe plusieurs méthodes standards permettant de représenter un protocole, selon que l'on souhaite une description visuelle et synthétique, ou au contraire plus formelle et détaillée. La première approche correspond à une documentation textuelle complète, similaire à celle présentée dans la section précédente mais enrichie de nombreux détails techniques et souvent rendue publique. Ce type de description est commune pour les documents standards telles que les RFCs (Request For Comments). La seconde méthode est une représentation très concise des échanges de messages, appelée notation Alice & Bob. Elle permet de résumer les interactions entre les entités du protocole. Par exemple, si Alice initie l'échange en envoyant "Ping" et que Bob lui répond "Pong", cela s'écrit :

$$Alice \rightarrow Bob : Ping$$
$$Bob \rightarrow Alice : Pong$$

Enfin, une représentation graphique du protocole peut être utilisée sous forme de diagramme, offrant une visualisation claire des échanges. C'est cette dernière représentation qui sera adoptée dans le corps de cette thèse pour illustrer les protocoles étudiés. Par exemple, le protocole élémentaire ci-dessus sera représenté comme dans la figure ci-contre.



D.1.3. Propriétés de sécurité

Comme mentionné précédemment, les protocoles cryptographiques sont censés garantir certaines **propriétés de sécurité**, qui varient selon leur domaine d'application. Parmi les plus courantes, on trouve :

- **Authentification** : Chaque participant doit pouvoir vérifier l'identité de l'autre. Pour reprendre l'analogie de l'énigme initiale (Appendix D.1.1), Alice doit s'assurer que ses messages parviennent bien à Bob. De son côté, Bob doit vérifier que la boîte qu'il reçoit provient bien d'Alice, et non d'un adversaire usurpant son identité. Cette notion est également cruciale dans un protocole bancaire, où la banque doit s'assurer que la carte utilisée est autorisée à effectuer la transaction sur le compte visé.
- **Confidentialité** : Lorsqu'un message est chiffré, il doit rester secret et ne pouvoir être déchiffré que par les parties autorisées. Cette propriété est essentielle dans la quasi-totalité des protocoles : garantir la confidentialité d'une conversation dans une messagerie instantanée, empêcher l'accès aux informations bancaires ou au code PIN en regardant les messages échangés durant le protocole, préserver le secret d'un vote électronique, etc.
- **Anonymat** : Les participants souhaiter que leur identité reste cachée, même vis-à-vis du serveur d'authentification. C'est le cas, par exemple, dans des systèmes comme IDEMIX (IBM) [12] ou certains protocoles de vote électronique. Cette propriété est souvent accompagnée de la *non-traçabilité*, c'est-à-dire l'impossibilité de corréler plusieurs sessions à un même utilisateur.

[12]: Camenisch et al. (2010), *Specification of the identity mixer cryptographic library*

D'autres propriétés de sécurité sont plus spécifiques à certains domaines d'application. Par exemple, dans le contexte du vote électronique, il peut être nécessaire de garantir l'*éligibilité* : le protocole doit s'assurer que seuls les bulletins des électeurs légitimes (en fonction de l'âge, de la nationalité, de l'inscription sur les listes, etc.) sont comptabilisés, et que chaque électeur ne peut voter qu'une seule fois.
Pour satisfaire ces exigences, les protocoles s'appuient sur différentes **primitives cryptographiques**, dont voici quelques exemples :

- **Chiffrement symétrique** : Il ressemble à la boîte cadenassée de l'énigme en Appendix D.1.1, à la différence que ce type de chiffrement nécessite que les participants partagent une même clé secrète (e. g., échangée préalablement via un autre protocole). L'expéditeur chiffre le message avec cette clé symétrique, obtenant un *ciphertext*, que le destinataire peut déchiffrer avec la même clé pour retrouver le message en clair (le *plaintext*). Seuls les participants possédant la clé secrète peuvent déchiffrer les messages chiffrés avec celle-ci.

- **Chiffrement asymétrique** : Ce chiffrement ressemble au précédent mais dans ce cas, chaque participant possède une paire de clés : une *clé publique*, accessible à tous, et une *clé privée*, différente et gardée secrète par son propriétaire. L'expéditeur chiffre un message avec la clé publique du destinataire, qui est le seul à pouvoir le déchiffrer avec sa clé privée. Dans l'analogie des messagers cleptomanes, Alice utiliserait le cadenas public de Bob pour fermer sa boîte, et seul Bob pourrait l'ouvrir avec sa clé privée, empêchant l'attaque de Charlie.
- **Fonctions de hachage cryptographique** : Ces fonctions associent à chaque donnée d'entrée x une sortie $\mathsf{h}(x)$, idéalement unique : s'il n'existe pas de $x' \neq x$ tel que $\mathsf{h}(x) = \mathsf{h}(x')$, on dit qu'il n'y a pas de collision. Il est peu coûteux de calculer $\mathsf{h}(x)$ à partir de x mais très coûteux de retrouver x à partir de $\mathsf{h}(x)$.
- **Signature** : Elle permet à un auteur de prouver l'origine et l'intégrité de son message. Comme le chiffrement asymétrique, elle repose sur un couple clé publique-clé privée. L'expéditeur signe un message avec sa clé privée (qui pour rappel est exclusivement possédée par l'auteur) et n'importe qui peut vérifier cette signature avec la clé publique correspondante. Une variante, appelée *signature aveugle*, utilisée notamment dans certains systèmes de vote, permet de signer un message sans en connaître le contenu, à la manière d'une lettre placée dans une enveloppe qu'on ferait signer à travers celle-ci.
- **Exponentiation Diffie-Hellman** : Cette primitive est une exponentiation modulaire dans un groupe cyclique, utilisant un générateur public g. Elle permet à deux participants de calculer une clé partagée sans révéler leur secret respectif. Par exemple, si Alice choisit un secret x et Bob un secret y, ils échangent publiquement g^x et g^y, puis chacun peut calculer g^{x*y} de son côté, obtenant ainsi la même clé partagée sans que leurs secrets ne soient exposés. En effet, grâce à l'hypothèse décisionnelle de Diffie-Hellman (pour g^x et g^y donné, il est difficile de calculer g^{x*y}), un attaquant connaissant seulement g^x et g^y ne pourra pas dériver la clé partagée.
- **Opérateur XOR** : L'opérateur $\oplus$ (dit "ou exclusif" ou "XOR") est associatif et commutatif et permet d'"annuler" une valeur : $x \oplus x = 0$. Revenons à notre énigme des messagers cleptomanes : dans la solution mise en place par le couple, remplaçons les couples clé-cadenas d'Alice et de Bob par une valeur (a pour Alice, b pour Bob). Au début, quand Alice cadenasse la boîte contenant m, elle calcule à la place $m \oplus a$. Bob reçoit cette valeur et calcule $m \oplus a \oplus b$, qu'Alice peut en partir déchiffrer en calculant $m \oplus a \oplus b \oplus a = m \oplus b$. Quand Bob reçoit cette valeur finale, il peut calculer $m \oplus b \oplus b = m$ et connaître la valeur du message d'Alice.

D.2. Modèle symbolique

Une manière d'analyser la sécurité d'un protocole cryptographique consiste à adopter l'approche symbolique des méthodes formelles.

D.2.1. Modèle de protocoles

Dans un modèle symbolique, les participants, les messages et les primitives cryptographiques sont représentés de manière abstraite, afin de se concentrer sur les interactions logiques plutôt que sur les détails de la cryptographie elle-même. En particulier, un message est modélisé comme un *terme*, qui est soit une variable x, soit un nom n, soit un symbole de fonction appliqué sur d'autres termes. Cela s'oppose au modèle calculatoire où un message est représenté comme une suite de bits (*bitstring*), plus fidèle à la réalité des implémentations. De même, la cryptographie est supposée parfaite (on parle d'*hypothèse de cryptographie parfaite*) et son comportement est décrit à l'aide d'*équations* et de *symboles de fonction*. Par exemple :

- La modélisation du **chiffrement symétrique** requiert deux symboles de fonction (`enc` pour le chiffrement et `dec` pour le déchiffrement) et une équation :

  $$\mathtt{dec}(\mathtt{enc}(m, k), k) = m$$

  Cela signifie que si l'on chiffre un message m avec la clé k ($\mathtt{enc}(m, k)$), la seule manière de le retrouver est de le déchiffrer (`dec`) avec cette même clé k.
- La modélisation de l'**opérateur XOR** repose sur deux symboles de fonction ($\oplus$ pour l'opération elle-même, et `zero` pour représenter la constante 0, élément neutre) ainsi que sur les équations suivantes :

  $$\begin{array}{ll} a \oplus (b \oplus c) = (a \oplus b) \oplus c & a \oplus b = b \oplus a \\ a \oplus a = \mathtt{zero} & a \oplus \mathtt{zero} = a \end{array}$$

  Ces équations décrivent, pour tout message, les propriétés associatives et commutatives de l'opérateur, ainsi que sa capacité à "annuler" un message, comme mentionné précédemment.

Cette abstraction des primitives cryptographiques facilite grandement l'automatisation de la vérification par des outils (voir Appendix D.2.3). Il est toutefois possible d'affiner ce modèle en introduisant des équations supplémentaires pour refléter des comportements cryptographiques plus réalistes.

Dans cette thèse, nous présentons deux manières de modéliser symboliquement un protocole : la première (Chapter 2) s'appuie sur le pi-calcul appliqué [13], et la seconde (Chapter 4) sur des règles de réécriture de multisets [14]. Les deux formalismes nous permettent de décrire les agents participant au protocole, ainsi que la manière dont ils interagissent.

[13]: Abadi et al. (2018), *The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication*

[14]: Cervesato et al. (1999), *A meta-notation for protocol analysis*

D.2.2. Adversaire de Dolev-Yao

Les modèles symboliques ne permettent pas seulement de modéliser et d'analyser les protocoles cryptographiques, mais aussi de représenter la menace contre laquelle on souhaite se protéger. Plus un attaquant dispose de capacités d'interaction avec le protocole, plus il devient difficile de garantir des propriétés de sécurité. On distingue généralement un attaquant *passif* d'un attaquant *actif*.

Un attaquant *passif* observe les communications sans interférer dans leur déroulement. Il **espionne** les échanges de messages : il lit les messages non chiffrés pour en extraire de l'information et copie également les messages (même ceux qu'il ne peut pas comprendre). Son objectif est de collecter des informations sensibles (mots de passe, clés, contenu privé), dans le but d'analyser ou de casser des algorithmes, tout en restant discret.

Un attaquant *actif*, quant à lui, interagit directement avec le système et perturbe les échanges. Il peut **intercepter** et **modifier** les messages (par exemple à l'aide d'informations précédemment espionnées), mais aussi **supprimer**, **retarder** ou **injecter** des messages qu'il construit lui-même. Son objectif est d'obtenir des données confidentielles (clés, messages. . .), d'usurper l'identité d'un participant, ou encore de contourner des mécanismes de sécurité. Un exemple typique d'attaque réalisée par un attaquant actif est celle de *l'homme du milieu* (man-in-the-middle), dans laquelle l'attaquant s'intercale entre deux participants et modifie les messages afin de se faire passer pour l'un auprès de l'autre.

Dans les modèles symboliques, la modélisation de l'adversaire remonte aux travaux fondateurs de Dolev et Yao [15] au début des années 1980. Dans le modèle dit de Dolev-Yao, on considère un attaquant **actif puissant**, qui dispose d'un contrôle total sur le réseau et possède toutes les capacités mentionnées précédemment. Il peut également agir sur les primitives cryptographiques et les symboles de fonction modélisés, mais seulement dans le cadre des équations choisies (comme évoqué ci-dessus).

[15]: Dolev et al. (1983), *On the security of public key protocols*

D.2.3. Propriétés et vérification

Les propriétés de sécurité, telles que celle présentées dans la section précédente, sont modélisées par des formules logiques. On dira qu'une propriété de sécurité est vérifiée si la formule logique qui la modélise est vraie pour toutes les exécutions possibles du protocole. Cette vérification est toujours effectuée en présence d'un attaquant. Comme certains outils permettent de traiter un nombre illimité de sessions, le problème de vérification devient alors indécidable, et l'analyse peut ne pas aboutir.

L'analyse symbolique automatisée des protocoles constitue une approche efficace pour détecter des attaques ou prouver leur absence. Les preuves formelles de sécurité, fondées sur des définitions précises des modèles de sécurité et de menace, permettent d'identifier des familles entières de vulnérabilités. Cette méthode représente un bon compromis entre la précision du modèle et le niveau d'automatisation, grâce au support d'outils, ce qui permet d'analyser de lourds protocoles déployés à grande échelle tout en prenant en compte des propriétés de sécurité avancées et des modèles de menace complexes.

Outils de vérification

Les outils de vérification les plus avancés, tels que ProVerif [16, 17] et Tamarin [18, 19], sont aujourd'hui capables de passer à l'échelle pour analyser des protocoles déployés dans l'industrie. Ces deux outils sont les analyseurs de protocoles les plus utilisés dans ce domaine de recherche,

[16]: Blanchet (2001), *An Efficient Cryptographic Protocol Verifier Based on Prolog Rules*
[17]: Blanchet et al. (2022), *ProVerif with Lemmas, Induction, Fast Subsumption, and Much More*
[18]: Schmidt et al. (2012), *Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties (extended version)*
[19]: Basin et al. (2022), *Tamarin: Verification of Large-Scale, Real-World, Cryptographic Protocols*

et ont par exemple permis d'analyser avec succès des protocoles tels que TLS [4, 20], des protocoles d'authentification du standard 5G [6], le protocole de messagerie instantanée Signal [21], les protocoles de paiement du standard EMV [2, 22], ou encore le standard OPC-UA pour les systèmes industriels [23].

DeepSec [24] est un autre outil symbolique, spécialisé dans la vérification de propriétés d'équivalence, et conçu pour fonctionner uniquement dans un contexte avec un nombre limité de sessions, contrairement à ProVerif et Tamarin. Enfin, on peut mentionner Sapic$^+$ [25], une plateforme intégrée à Tamarin, qui centralise un modèle unique de protocole et permet de l'exporter vers les trois outils mentionnés précédemment (ProVerif, Tamarin et DeepSec), utilisés comme moteurs d'analyse.

[4]: Cremers et al. (2017), *A Comprehensive Symbolic Analysis of TLS 1.3*
[20]: Bhargavan et al. (2022), *A Symbolic Analysis of Privacy for TLS 1.3 with Encrypted Client Hello*
[23]: Diemunsch et al. (2025), *A Comprehensive Formal Security Analysis of OPC UA*
[24]: Cheval et al. (2018), *DEEPSEC: Deciding Equivalence Properties in Security Protocols Theory and Practice*
[25]: Cheval et al. (2022), *Sapic$^+$: protocol verifiers of the world, unite!*

D.3. Contributions

Nous présentons dans cette thèse deux contributions principales :

- **Analyse formelle approfondie du protocole EDHOC** : EDHOC (Ephemeral Diffie-Hellman Over COSE) est un protocole d'échange de clés proposé par le groupe de travail LAKE (Lightweight Authenticated Key Exchange) de l'IETF. Il a été conçu pour minimiser la taille des messages, afin de s'adapter aux technologies de communication contraignantes utilisées dans l'Internet des objets (IoT). Nous menons une analyse formelle détaillée de la version 12 du projet, publiée en octobre 2021, en prenant en compte les différentes méthodes d'authentification proposées ainsi que les nombreuses options du protocole. Notre étude repose sur la plateforme Sapic$^+$, qui permet de compiler une spécification unique vers trois outils de vérification de protocoles (ProVerif, Tamarin et DeepSec), tirant parti des forces respectives de chacun. Nous considérons une large variété de scénarios de compromission, et exploitons également des résultats permettant de modéliser certaines faiblesses existantes dans les primitives cryptographiques, en assouplissant l'hypothèse classique de cryptographie parfaite utilisée en analyse symbolique. Bien que notre analyse ait confirmé la sécurité du protocole pour les modèles de menace les plus simples, plusieurs vulnérabilités ont été mises en évidence lorsque des modèles de menace plus avancés ont été pris en compte. Ces failles ont été reconnues par le groupe de travail, et les mesures correctives que nous proposons (et prouvons sûres) ont été intégrées dans la version 14, publiée en mai 2022.
- **Prise en charge dans Tamarin des opérateurs associatifs et commutatifs (AC) customisables** : Nous étendons l'outil Tamarin afin de permettre aux utilisateurs de spécifier des symboles AC dans leur théorie équationnelle. Afin d'éviter les problèmes de non-terminaison, nous proposons une nouvelle propriété qui borne certaines chaînes de déduction. Nous utilisons l'algorithme de déduction de Tamarin pour vérifier si cette borne est effectivement correcte pour une théorie équationnelle donnée, généralisant ainsi des travaux antérieurs. De plus, notre travail a permis d'identifier que la version intégrée de l'opérateur XOR dans Tamarin utilisait une optimisation susceptible de produire des résultats incorrects,

ce qui aurait également pu affecter nos résultats. Désactiver simplement cette optimisation entraîne une non-termination sur une grande partie des exemples du dépôt Tamarin. Nous concevons donc une nouvelle règle de réduction de contraintes qui élimine certaines chaînes de constructeurs redondantes, et nous en prouvons la correction. Nous illustrons l'efficacité et la performance de notre approche à travers divers exemples, incluant l'opérateur XOR, les multisets, le rechiffrement, l'exponentiation Diffie-Hellman et le déchiffrement distribué. Il s'avère que notre propriété est vérifiée dans tous ces cas. Par ailleurs, comparée au support intégré de Tamarin pour l'opérateur XOR ou les multisets, notre méthode n'introduit qu'un léger surcoût au démarrage (lors de la vérification de la propriété), tout en maintenant des performances similaires sur la majorité des exemples. Enfin, le rechiffrement et le déchiffrement distribué n'étaient jusqu'ici pas pris en charge, et notre modèle d'exponentiation Diffie-Hellman, bien que plus simple que celui intégré, permet la réutilisation du symbole d'exponentiation dans d'autres équations, ce qui n'est pas possible avec le modèle natif.