Contributions to the Verification of Cryptographic Protocols

David Baelde

▶ **To cite this version:**

David Baelde. Contributions to the Verification of Cryptographic Protocols. Computer Science [cs]. Université Paris-Saclay, 2021. tel-04549522

HAL Id: tel-04549522
https://theses.hal.science/tel-04549522

Submitted on 17 Apr 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Habilitation à Diriger des Recherches

David Baelde

Contributions à la Vérification des Protocoles Cryptographiques

Habilitation à Diriger des Recherches de l'Université Paris-Saclay
Spécialité Informatique
Présentée et soutenue publiquement le 10 février 2021 au
Laboratoire Méthodes Formelles (LMF)

Rapporteurs:

Martín Abadi	Principal Scientist, Google & Professor emeritus, UC Santa Cruz
Gilles Barthe	Scientific Director, Max Planck Institute for Security and Privacy
Frank Pfenning	Professor, Carnegie Mellon University

Composition du jury:

Martín Abadi	Principal Scientist, Google & Professor emeritus, UC Santa Cruz
Gilles Barthe	Scientific Director, Max Planck Institute for Security and Privacy
Véronique Cortier	Directrice de Recherche, CNRS
Paul Gastin	Professeur, ENS Paris-Saclay
Catuscia Palamidessi	Directrice de Recherche, Inria
Frank Pfenning	Professor, Carnegie Mellon University

Résumé

Les méthodes formelles mettent les outils de l'informatique théorique au service de la conception et de la vérification de systèmes fiables. Depuis les années 80, la vérification des protocoles cryptographiques a été un sujet de recherche actif dans ce domaine, ce qui a rendu possible la découverte automatique d'attaques et la preuve formelle de propriétés de sécurité pour des classes de protocoles, d'attaquants et de propriétés s'élargissant avec le temps: On peut notamment se restreindre à certaines classes d'attaquants spécifiées symboliquement, ou considérer des machines de Turing arbitraires. Par ailleurs, les propriétés de sécurité peuvent être vues comme des problèmes d'accessibilité, ce qui convient par exemple pour la confidentialité, ou plus généralement comme des problèmes d'équivalence comportementale, ce qui permet aussi de rendre compte de diverses propriétés liées au respect de la vie privée.

Nous présentons dans ce mémoire d'habilitation plusieurs contributions à ce domaine. Nous nous intéressons d'abord à la modélisation de la propriété de non-traçabilité et à sa preuve en nombre non-borné de sessions, via la vérification de conditions suffisantes au moyen d'outils existants. Nous développons ensuite plusieurs techniques de réduction d'ordres partiels qui ont permis l'amélioration des outils actuels de vérification d'équivalences en nombre de session borné. Nous présentons enfin des travaux en cours sur une nouvelle approche pour la preuve de protocoles dans le modèle calculatoire, fondée sur le développement d'une meta-logique au dessus de la logique CCSA.

Summary

Formal methods use techniques from theoretical computer science for the design and verification of trustworthy systems. Since the 80', the verification of cryptographic protocols has been the topic of active research in this domain, which has made it possible to automatically discover attacks and to formally prove security properties for ever-increasing classes of protocols, attackers and properties: Particular classes of attackers may be described symbolically, or arbitray Turing machines may be considered. Security properties may be viewed as reachability problems, which is appropriate e.g. for secrecy, or more generally as behavioural equivalences, which allows to capture various privacy-type properties.

This habilitation manuscript presents several contributions to this domain. We first consider the formal modelling of unlinkability and a technique for proving unlinkability with unbounded sessions, via the verification of sufficient conditions using existing tools. We then develop several partial order reduction techniques that have brought performance improvements in state-of-the-art equivalence verification tools for bounded sessions. We finally present ongoing work on a new approach for proving protocols in the computational model, based on the development of a meta-logic over the CCSA logic.

Contents

1 Preamble

The purpose of an habilitation manuscript is to demonstrate that one is suitable to teach as a university professor, and independently supervise the research work of others. This is usually done through the presentation of the research work that one has carried out after his or her PhD. In my case, I have worked on topics that are so distinct that it would not be possible to present them all in this document at an interesting level of detail. Therefore, I shall start with a short description of the main areas of my work, and will focus on just one for the rest of the document.

1.1 Biography

In December 2009, I have obtained a PhD in Computer Science from École Polytechnique, titled *A Linear Approach to the Proof Theory of Least and Greatest Fixed Points* and supervised by Dale Miller.

After that, I have worked as a post-doc for slightly more than two years. First at the University of Minnesota with Gopalan Nadathur, then at Université Paris-Sud with Christine Paulin, and finally at the IT University of Copenhagen in Carsten Schürmann's DemTech project on electronic democracy.

In September 2012, I started working as an assistant professor in the Computer Science department of École Normale Supérieure (ENS) de Cachan, and I joined the security axis of the Laboratoire Spécification and Vérification (LSV). I still hold this position, though the school is now called ENS Paris-Saclay and LSV has merged with the Vals team of LRI to form a new research laboratory called Laboratoire Méthodes Formelles.

My research interests have slowly shifted during this time. Starting from linear logic and logical frameworks, I have worked at Université Paris-Sud on formal security proofs for watermarking schemes (in Coq) and I have developed an interest for electronic voting

in Copenhagen. I have finally joined LSV with a research project that notably proposed to apply techniques from linear logic to the verification of security protocols. This manuscript will focus on my work in security, although my activities at LSV have only partially been dedicated to it.

1.2 Infinitary proof theory

During my PhD I have studied the proof theory of least and greatest fixed points in linear logic, and investigated the use of fixed point logics in several contexts. In particular, I have been interested in reasoning about finite automata [Bae09] using linear logic with least fixed points. This had lead me to develop an interest for circular proofs, and in particular for the works of Santocanale [San02] and of Brotherston and Simpson [BS11].

I presented my work on finite automata at the 2012 North American meeting of the ASL in Madison, Wisconsin, where I got challenged by Moshe Vardi to make it "scale" to infinite words automata. In collaboration with Amina Doumane, Lucca Hirschi, and Alexis Saurin, we finally managed to do so in the context of the linear-time μ-calculus [Dou+16]. In this work, we design an infinitary proof system for that μ-calculus, where derivations are finitely branching but non-well-founded trees, and infinite branches need to be justified by a validity condition based on threads, i.e. the sequences of formula occurrences that are explored in the branch. We have shown that a particular class of circular proofs in that system can be translated to standard finitary proofs using (co)induction rules. These results are finally applied to obtain a completeness result for inclusions of Büchi automata suitably encoded as μ-calculus formulas: we first obtain circular proofs of inclusions by exploiting Safra's determinization technique, then translate these circular proofs to finitary ones.

I have worked more generally on studying and developing the foundations of infinitary proof theory. I have notably obtained, with Amina Doumane and Alexis Saurin, an infinitary cut elimination result [BDS16] for multiplicative additive linear logic with least and greatest fixed points (μMALL). In non-well-founded sequent calculus derivations for that logic, the usual cut reductions generally yield infinite reduction sequences. However, if one restricts to reduction sequences that only reduce bottom-most cuts, one would hope that reductions produce, at the limit, cut-free derivations. Santocanale and Fortier have shown that this is the case in the purely additive fragment [FS13]. With Doumane and Saurin, we have shown that it also holds for μMALL [BDS16]. Our argument significantly differs from the one used by Fortier and Santocanale in the additive case, and makes a surprising use of a Boolean semantics for our linear logic.

Amina Doumane has completed a PhD in 2017 on infinitary proof theory, supervised by Alexis Saurin and myself. She has notably built on our joint work [Dou+16] to develop a new, constructive proof of completeness for the full μ-calculus [Dou17].

1.3 Sequent calculi for tense logics

While at LSV, I have embarked on a project led by Sylvain Schmitz to study data logics. As modal logics, data logics express properties of relational structures whose nodes are labelled by propositions from a finite alphabet, but also carry a datum from an infinite domain which can be compared for (dis)equality. For example, $w \models \mathsf{F}_{\neq}\phi$ holds when there exists some node w' such that $w \to^* w'$ and the two worlds carry distinct data. These logics are relevant to database theory and, more specifically, XML query langages such as XPath. They have been widely studied and several fragments have been identified for which satisfiability is decidable [GK02; Hid04; GF05; CL09; Boj+09; JL11; FS09; FS17; Cze+18].

With Anthony Lick and Sylvain Schmitz, we have developed a benchmark to study the practical relevance of the various decidable fragments [BLS19]. We first identified precisely which fragment of XPath actually corresponds to each theoretical fragment, considering the addition of practical features of XPath that had been ignored in theoretical investigations, as long as these additions did not change the complexity of satisfiability. We then gathered XPath code from real-world open-source projects and measured the coverage of each fragment on this benchmark. This allowed us to evaluate which theoretical decision procedures could have the highest impact in practice, and identify research directions for future improvements.

More theoretically, our main line of research has been an attempt to unify the study of data logics through the use of proof theory. Indeed, model theory and automata theory are commonly used in this field, but they tend to yield impractical decision procedures and do not seem to give a modular view of the landscape of logics that would help to understand it. In contrast, we have strived to develop proof systems for data logics that can yield practical decision procedures and, by virtue of the sequent calculus, a more modular understanding of the logics. With Simon Lunel and Sylvain Schmitz, we have first developed a sound and complete sequent calculus proof system for a data logic on trees with only (transitive) downward navigation [BLS16]. This system enjoys optimal complexity backward proof-search. Then, with Anthonly Lick and Sylvain Schmitz, we have studied logics with converse navigation and obtained proof systems for tense logic over words [BLS18a] and ordinals [BLS18b]. Again, the obtained systems are complete and enjoy optimal-complexity proof-search. In order to succesfully capture bidirectional navigation, we have had to move from sequents to (enriched) hypersequents. Further, in order to obtain a terminating proof-search, we have made use of annotations expressing maximality conditions, so that proof search becomes a search for some minimal countermodels.

Anthony Lick has obtained a PhD in 2019, supervised by Sylvain Schmitz and myself. As part of his PhD work, he has managed to extend our calculus for ordinals [BLS18b] to data ordinals [Lic19].

1.4 Security protocol verification

While at LSV, I have developped my interest for the application of formal methods to the analysis of security protocols. This area of research is interesting for many reasons. First, it deals with problems that are crucial for modern individuals. Security protocols are involved in many aspects of our daily lives: personal communications, financial operations, access control in private buildings and public transportation, voting, etc. and news recall us regularly that security issues in any of these domains can have devastating consequences. Second, applying formal methods to security protocols requires a careful modelling of systems and security properties. This task can be quite delicate, as security requirements are sometimes very informal, and the complex models of security protocols can easily hide subtle modelling defects. This problem has driven the design of modelling languages that are both rich enough and mathematically simple. Third, working in these formal frameworks offers opportunities to import techniques from more abstract fields of computer science, such as rewriting, logic, type systems, concurrency theory, etc. All these reasons have attracted many fundamental computer scientists to the field, myself included.

I have worked on automatic techniques for verifying equivalences of security protocols, which are commonly used to express privacy-type properties such as anonymity or unlinkability. I have considered this problem in symbolic models using (variants of) the applied pi-calculus [AF01]. In 2012, precise (semi-)decision procedures already existed and were implemented in a few tools [TD10; Che14; Cha+16], working under the assumption that protocol traces are bounded. These procedures performed a relatively naive exploration of all protocol executions, testing at each step that the two protocols under consideration cannot be distinguished based on their output messages. With Stéphanie Delaune and Lucca Hirschi, we have designed partial-order reduction (POR) techniques to avoid exploring redundant executions. Our first technique [BDH14; BDH17], inspired by focused proof systems from linear logic, works under some action-determinism assumption and has brought significant performance improvements in most equivalence verification tools. In a subsequent work, we have managed to re-use the classic POR techniques of persistent and sleep sets [God95], lifting in this way the action-determinism assumption [BDH18]. These results are to a large extent orthogonal to the heart of the (semi-)decision procedures for verifying equivalence. I have also worked one such procedure with Stéphanie Delaune, Steve Kremer and Ivan Gazeau, to bring support for exclusive or in the Akiss tool [Bae+17].

In another important line of work, I have been concerned with the verification of unlinkability for unbounded executions. Intuitively, unlinkability holds when an outside observer cannot tell if two uses of the same protocol are from the same user or not. Formally expressing this property is tricky, and has been the subject of a string of imperfect papers in the literature. More importantly, verifying the property is out of reach of general-purpose tools for unbounded sessions such as Proverif [Bla+01] or Tamarin [Bas+12], since they rely on an overly constraining notion of equivalence, namely *diff-equivalence*, that does not hold for unlinkability. With Stéphanie Delaune and Lucca Hirschi, we have proposed two sufficient conditions that imply unlinkability

and can be verified with Proverif [HBD16; HBD19]. Later on, with Stéphanie Delaune and Solène Moreau, we have extended this work to protocols with state and a centralized reader, which has necessitated a modification of the formal definition of unlinkability, the addition of a third condition [BDM20] and the move from Proverif to Tamarin for the verification of the conditions on case studies.

I have supervised with Stéphanie Delaune the PhD thesis of Lucca Hirschi (completed in 2017) and we are currently supervising the PhD thesis of Solène Moreau.

1.5 Acknowledgments

I am honored that Martín Abadi, Gilles Barthe and Frank Pfenning have accepted to review this manuscript, and I thank them for their work and comments. I also thank Véronique Cortier, Paul Gastin and Catuscia Palamidessi for being part of my jury.

I am most grateful for my close collaborators Stéphanie Delaune, Alexis Saurin and Sylvain Schmitz who have deeply influenced my work and accompanied me on exciting journeys for the past nine years. I also warmly thank Amina Doumane, Lucca Hirschi, Anthony Lick and Solène Moreau, who have embarked as PhD students under our supervision; I consider myself lucky to have had such talented students.

More generally, I have benefited from an exceptional work environment at LSV. I am thankful to all my colleagues there who have contributed to make it a pleasant and exciting place to work. Teaching at the Computer Science department has also been a delightful experience. I would like to thank all the students who have made my classes lively and interesting, as well as my colleagues, particularly Paul Gastin and Serge Haddad. Last but not least, teaching with Hubert Comon has been a great experience which has strongly influenced me. Surprisingly, Hubert's influence on my research topics has only started to show very recently, as shown in the last chapter of this manuscript.

I would finally like to thank Steve Kremer, Vincent Cheval and Itsaka Rakotonirina for our interesting discussions on POR and for their remarks on preliminary versions of this manuscript, Bruno Blanchet for many interesting discussions notably during my stay at Prosecco, and Charlie Jacomme and Adrien Koutsos for being part of the Squirrel team.

1.6 Outline

The rest of this document presents and discusses my main lines of work in security.

Chapter 2 defines the formal framework that is used to model protocols in most of my work. This framework belongs to the category of symbolic models and more specifically relies on the applied pi-calculus. The chapter is based on lecture notes from my past teaching of symbolic proofs of security protocols, and its technical content is significantly more detailed than in other chapters. The definitions and results that are presented are however important for some of the discussion in the next chapters.

Chapter 3 presents my contributions to the verification of unlinkability through sufficient conditions. I use it as an opportunity to carefully discuss the formal modelling of

unlinkability, addressing several formal definitions that have been used in the literature.

Chapter 4 is dedicated to my works on partial-order reductions for security protocol equivalences. This is historically my first line of research in the domain, drawing explicitly from my past expertise in proof-search and linear logic. The contributions that resulted from it are my most mature in the field of security. I attempt to put it in perspective and identity questions that are left open.

Chapter 5 describes some recent work on a new approach to obtain proofs of security protocols in the computational model. It is based on a logic that Bana and Comon have recently proposed for proving computational indistinguishability [BC14], on which we elaborate by developing a meta-logic and a dedicated proof assistant. This work opens up several directions for future work which we plan to explore in the coming years.

2

A Symbolic Model for Security Protocols

One of the most successful frameworks for specifying and verifying protocols in the symbolic model is the applied pi-calculus [ABF17], introduced by Abadi and Fournet [AF01] and popularized notably through its use in Blanchet's system Proverif [Bla+01]. Most of my work is based on variants of the applied pi-calculus, and we present in this chapter one such variant. The goal is to introduce the technical notions that will be used in the next chapters, providing enough background to be able to justify their relevance in the context of the formal verification of security protocols. In particular, we shall introduce and compare several notions of equivalence, even though only trace equivalence is used in other chapters. We also give an overview of the state of the art in automated verification of security protocols in symbolic models.

This chapter does not present any contribution of mine, and its only originality might be in the presentation. It is based on lecture notes from my past teaching of the course on symbolic proofs of protocols at the Master Parisien de Recherche en Informatique (MPRI).

We shall not present our process algebra in the style of [AF01] featuring active substitutions and extended processes (which is kept in, e.g., [ABF17; CK14; CCD13; BCK20]) but work instead with configurations, similar to, e.g., [Bla16]. Although the difference is shallow, we find that avoiding active substitutions makes the concepts more accessible, and we have thus followed this approach both in our research papers and when teaching.

The process algebra given here is a superset of the ones used in our main line of POR works [BDH14; BDH15a; BDH17]: we shall see in due time how replication and private channels need to be restricted. However it does not feature the destructors and repetition operators present in [HBD19] nor the state manipulations of [BDM20], which we will use in chapter 3.

2.1 Terms

Terms will be used to model messages and computations over messages. We assume several disjoint and countable sets of elementary objects: a set $\mathcal{X}$ of *variables*, which will be denoted by w, x, y, z; a set $\mathcal{N}$ of *names*, which will be denoted by n, m, k; a set $\mathcal{C}$ of *channels*, which will be denoted by c, d. Intuitively, names will be used to represent the secrets of honest participants (nonces, keys, etc.) which the attacker will not know *a priori*. We rely on a lightweight sort system featuring two sorts message and channel. Names have sort message, channels have sort channel, and each variable comes with a sort attached. We will however not specify the sorts of variables and terms when they are irrelevant or can be inferred from the context. We assume that there are infinitely many variables available for each of our two sorts.

Then, we assume a *signature* Σ, that is a set of *function symbols* whose sorts are of the form $\mathsf{message}^k \to \mathsf{message}$. Given a set elementary objects B, the set $\mathcal{T}(B)$ of *terms* generated from B using Σ is defined as the least set containing B that is closed by application of function symbols respecting their sorts. Terms will be noted s, t, u, v.

Example 2.1. *One possible signature is* $\Sigma = \{\mathsf{senc}, \mathsf{sdec}, \mathsf{pair}, \mathsf{fst}, \mathsf{snd}, \mathsf{ok}\}$. *The symbols* senc *and* sdec*, of arity 2, are meant to represent symmetric encryption and decryption. Pairing is modeled using* pair *of arity 2 and projection functions* fst *and* snd*, both of arity 1. Finally, the symbol* ok *is of arity* 0*, i.e. it is a constant. When using this signature, we will often write* $\langle s, t\rangle$ *rather than* $\mathsf{pair}(s,t)$*, and* $\{m\}_k$ *for* $\mathsf{senc}(m,k)$.

Given a term t, we define $\mathsf{fv}(t)$ as the set of variables that occur in t. Similarly, $\mathsf{fn}(t)$ (resp. $\mathsf{fc}(t)$) is the set of names (resp. channels) occurring in t. For convenience, we also define $\mathsf{fnc}(t)$ as $\mathsf{fn}(t) \cup \mathsf{fc}(t)$. A term is said to be *closed* when it contains no variable, i.e. when it belongs to $\mathcal{T}(\mathcal{N} \cup \mathcal{C})$; it is these terms that are going to be communicated between processes.

A *substitution* is a finite domain map from $\mathcal{X}$ to $\mathcal{T}(B)$ for some B, that respects sorts: variables of sort s must be mapped to terms of the same sort. Substitutions will be denoted by θ or σ, their domain will be noted $\mathsf{dom}(\cdot)$. The application of a substitution θ to a term t is defined as usual and noted $t\theta$.

Two mechanisms have been used in the literature to provide a meaning to terms. Terms may be considered up to an equational theory that indicates when two computations yield the same result [AF01]: for instance, one expects that $\mathsf{sdec}(\mathsf{senc}(s,t),t)$ and s represent the same message. Special function symbols, called destructors, may also be subject to rewrite rules [AB05]. This yields a distinction between computations (arbitrary terms, featuring destructors) and messages (destructor-free terms, obtained after some number of rewriting steps) which is convenient to model computations that may fail. For example, we may have that $\mathsf{sdec}(\mathsf{senc}(s,k),k)$ rewrites to s but $\mathsf{sdec}(\mathsf{ok},k)$ fails, indicating an encryption scheme where it is possible to distinguish random messages from actual ciphertexts. Rewrite rules can be encoded as equations [ABF17], but equations are more general since they allow, e.g., to model the algebraic properties of exclusive or. Rewrite rules are often easier to handle in automated verification, and the two styles are thus often combined, as is the case in Proverif. For simplicity, we only consider equations here, though the technical development would extend naturally to

destructors.

Our equational theory is going to be generated from equations between terms that may contain variables but no names — intuitively, computations should not distinguish one name from another, as they are just random values. We thus assume a set of equations $\mathsf{E} \subseteq \mathcal{T}(\mathcal{X})^2$ over terms of sort message; we will use an infix notation for it, writing $s\ \mathsf{E}\ t$ rather than $(s,t) \in \mathsf{E}$. We then define the binary relation $=_{\mathsf{E}}$ over $\mathcal{T}(\mathcal{N} \cup \mathcal{X})$ as the least equivalence relation that contains E and is closed under substitution and context closure.

Example 2.2. *With the signature of example 2.1, consider* E *made of three equations:*

$$\mathsf{sdec}(\mathsf{senc}(x,y),y)\ \mathsf{E}\ x,\ \ \mathsf{fst}(\mathsf{pair}(x_1,x_2))\ \mathsf{E}\ x_1,\ \ \textit{and}\ \ \mathsf{snd}(\mathsf{pair}(x_1,x_2))\ \mathsf{E}\ x_2.$$

We then have $\mathsf{fst}(\mathsf{sdec}(\mathsf{senc}(\mathsf{pair}(\mathsf{ok},n),k),k)) =_{\mathsf{E}} \mathsf{ok}$ *but* $\mathsf{ok} \neq_{\mathsf{E}} \mathsf{pair}(\mathsf{ok},n)$.

Proposition 2.1. *Let* $\sigma : \mathcal{N} \to \mathcal{N}$ *be a bijective renaming, and* u, v *be terms. We have* $u =_{\mathsf{E}} v$ *iff* $u\sigma =_{\mathsf{E}} v\sigma$.

2.2 Processes

Protocols are distributed programs manipulating messages using cryptographic primitives. They will be modelled as processes in a dialect of Abadi and Fournet's applied pi-calculus [AF01; ABF17] which, like the spi-calculus [AG99], elaborates on Milner's pi-calculus [Mil99]. The main difference between the seminal pi-calculus and its cryptographic extensions is that cryptographic calculi have a first-class notion of message, while only channel names can be communicated in the pi-calculus.

Processes are generated from the following grammar:

$$\begin{array}{rcl} P,Q & ::= & 0 \quad | \quad (P \mid Q) \quad | \quad !P \\ & | & \mathbf{in}(u,x).P \quad | \quad \mathbf{out}(u,v).P \quad | \quad \mathbf{new}\ x.P \\ & | & \mathbf{if}\ u = v\ \mathbf{then}\ P\ \mathbf{else}\ Q \end{array}$$

where $x \in \mathcal{X}$ and $u, v \in \mathcal{T}(\mathcal{N} \cup \mathcal{X})$. In a conditional process testing $u = v$, we require that the two terms are of sort message. In input and output constructs, we assume that the term u is of sort channel.

We allow the communication of channels and the creation of private channels, thus we do not constrain the sort of x and v in input, output, and **new** constructs. For instance, $\mathbf{new}\ x.\mathbf{out}(c,x).\mathbf{in}(x,y).0$ is a valid process if x is a variable of sort channel and y a variable of sort message. For simplicity, we do not consider a type system that would indicate whether a channel is meant to transport messages or channels, though we never consider examples where such a discipline could not be imposed.

We will often omit the null process, writing e.g. $\mathbf{out}(c,u)$ instead of $\mathbf{out}(c,u).0$ or $(\mathbf{if}\ u = v\ \mathbf{then}\ P)$ instead of $(\mathbf{if}\ u = v\ \mathbf{then}\ P\ \mathbf{else}\ 0)$.

In a process, variables may be bound by **in** and **new** constructs. The set of *free variables* of a process P is noted $\mathsf{fv}(P)$. We say that P is *closed* when $\mathsf{fv}(P) = \emptyset$. We write $\mathsf{fn}(P)$ for the set of free names of P, $\mathsf{fc}(P)$ for its free channels and $\mathsf{fnc}(P)$ for $\mathsf{fn}(P) \cup \mathsf{fc}(P)$. Processes will be considered up to α-equivalence, i.e. up to the renaming of bound variables.

$$(E, \mathcal{P} \cup \{0\}) \rightsquigarrow (E, \mathcal{P})$$
$$(E, \mathcal{P} \cup \{P \mid Q\}) \rightsquigarrow (E, \mathcal{P} \cup \{P, Q\})$$
$$(E, \mathcal{P} \cup \{!P\}) \rightsquigarrow (E, \mathcal{P} \cup \{P, !P\})$$
$$(E, \mathcal{P} \cup \{\mathbf{in}(c, x).P, \mathbf{out}(c, u).Q\}) \rightsquigarrow (E, \mathcal{P} \cup \{P\{x \mapsto u\}, Q\})$$
$$(E, \mathcal{P} \cup \{\mathbf{if}\ u = v\ \mathbf{then}\ P\ \mathbf{else}\ Q\}) \rightsquigarrow (E, \mathcal{P} \cup \{P\}) \text{ when } u =_{\mathsf{E}} v$$
$$(E, \mathcal{P} \cup \{\mathbf{if}\ u = v\ \mathbf{then}\ P\ \mathbf{else}\ Q\}) \rightsquigarrow (E, \mathcal{P} \cup \{Q\}) \text{ when } u \neq_{\mathsf{E}} v$$
$$(E, \mathcal{P} \cup \{\mathbf{new}\ x.P\}) \rightsquigarrow (E \cup \{n\}, \mathcal{P} \cup \{P\{x \mapsto n\}\}) \text{ when } n \notin E \cup \mathsf{fn}(\mathcal{P}, P)$$
$$(E, \mathcal{P} \cup \{\mathbf{new}\ x.P\}) \rightsquigarrow (E \cup \{c\}, \mathcal{P} \cup \{P\{x \mapsto c\}\}) \text{ when } c \notin E \cup \mathsf{fc}(\mathcal{P}, P)$$

The last rule only applies when x is of sort channel, and the previous one only applies when the bound variable is of sort message.

Figure 2.1: Internal reduction rules

2.3 Internal reduction

We now define how processes execute, by providing them with a reduction semantics. Reduction steps need to occur under some **new** and parallel constructs, which is achieved formally by defining our reduction relation $\rightsquigarrow$ not on processes but on *internal reduction configurations* of the form $(E, \mathcal{P})$ where $E \subseteq_{\text{fin}} \mathcal{N} \cup \mathcal{C}$ is called an *environment* and $\mathcal{P}$ is a finite multiset of closed processes[1]. Intuitively, the configuration

$$(\{n_1, \ldots, n_p, c_1, \ldots, c_q\}, \{P_1, \ldots, P_r\})$$

represents the process

$$\mathbf{new}\ x_1 \ldots \mathbf{new}\ x_p.\mathbf{new}\ y_1 \ldots \mathbf{new}\ y_q.\ (P_1 \mid \ldots \mid P_r)\{n_i \mapsto x_i\}_{i \in [1;p]}\{c_j \mapsto y_j\}_{j \in [1;q]}.$$

Definition 2.1 (Internal reduction)**.** *The binary relation $\rightsquigarrow$ on internal reduction configurations is given by the rules of figure 2.1.*

Note that **new** constructs are always executable, because we consider processes up to α-renaming. The only constructs on which reduction may be stuck are inputs and outputs, when their counterparts are not available in the multiset.

Example 2.3. *Assuming that n_1 and n_2 are two distinct names, we have:*

$$\begin{array}{ll} & (\emptyset, \{(\mathbf{new}\ n.\mathbf{out}(c, n)) \mid (\mathbf{new}\ n.\mathbf{out}(c, n)) \mid (\mathbf{in}(c, x).\mathbf{in}(c, y).\mathbf{out}(c, \langle x, y \rangle))\}) \\ \rightsquigarrow^8 & (\{n_1, n_2\}, \{\mathbf{out}(c, \langle n_1, n_2 \rangle)\}) \end{array}$$

Building on proposition 2.1, we easily obtain that names are interchangeable in internal reductions. From now on, a bijective renaming is a bijection over $\mathcal{N} \cup \mathcal{C}$ that sends names to names and channels to channels.

[1] The notation $X \subseteq_{\text{fin}} Y$ means that X is a finite subset of Y. Multisets are simply noted $\{P_1, \ldots, P_n\}$ and the union of multisets is noted $\cup$.

Proposition 2.2. *Let σ be a bijective renaming. For any internal reduction configurations $(E, \mathcal{P})$ and $(E', \mathcal{P}')$ we have $(E, \mathcal{P}) \rightsquigarrow (E', \mathcal{P}')$ iff $(E\sigma, \mathcal{P}\sigma) \rightsquigarrow (E'\sigma, \mathcal{P}'\sigma)$.*

This result notably implies that the precise choice of name or channel is inessential when reducing **new** constructs, as long as the freshness condition is respected. We shall often use it with bijections that exchange only two elements, which we note $\{t \leftrightarrow t'\}$.

Remark 2.1. *Internal communications make internal reductions non-deterministic. Given two processes P_1 and P_2, let us define:*

$$P_1 + P_2 \stackrel{def}{=} \textbf{new}\ x, y.\ \textbf{out}(x, y) \mid \textbf{in}(x, z).P_1 \mid \textbf{in}(x, z).P_2$$

For any E and $\mathcal{P}$, and any $i \in \{1, 2\}$, we have $(E, \mathcal{P} \cup \{P_1 + P_2\}) \rightsquigarrow^* (E, \mathcal{P} \cup \{P_i\})$.

Application to modelling security properties

Several security properties can be formally defined using internal reduction. Such properties, including e.g. secrecy and authenticity, are broadly called *reachability* properties. We detail next the case of secrecy, and refer the reader to [CK14; Bla16] for accounts of authentication.

Intuitively, a value remains secret if no attacker can derive it. The only difficulty[2] here is to precisely define what we mean by attacker. We actually consider an attacker in the style of Dolev-Yao [DY81], which can intercept and inject messages on public channels, and derive new messages from the intercepted ones by applying function symbols and the equational theory. Such attackers are elegantly captured by processes interacting with the protocol's processes, which leads us to the following definition.

Definition 2.2 (Secrecy)**.** *Let $(E, \mathcal{P})$ be an internal reduction configuration, and s be a closed term. We say that $(E, \mathcal{P})$ does* not *ensure the secrecy of s when there exists a closed process A such that $E \cap \mathsf{fnc}(A) = \emptyset$ and*

$$(E, \mathcal{P} \cup \{A\}) \rightsquigarrow^* (E', \{\textbf{out}(c, u)\} \cup \mathcal{P}')$$

for some E', $\mathcal{P}'$, $c \notin E'$ and $u =_{\mathsf{E}} s$. Otherwise, we say that P ensures the secrecy of s.

Intuitively, the process A in this definition represents an arbitrary attacker. It can perform any computation allowed in our semantics, communicating with $\mathcal{P}$ in any way, possibly intercepting and injecting messages. The attacker can also generate new names and compute new messages modulo $=_{\mathsf{E}}$ by applying function symbols on known messages. The set of symbols E models the initial secrets of the protocol. Accordingly, we only consider attacker processes which do not mention these secrets initially. Secrecy then amounts to verify that the term s, relying on these *a priori* secrets, cannot be learned by the attacker after some interaction with the protocol. If $\mathsf{fnc}(s)$ is disjoint from E, then the secrecy of s obviously does not hold.

It is important that the internal reduction rules for **new** prevent the re-use of past private symbols that have become unusued — freshness wrt. $\mathsf{fn}(\mathcal{P}, P)$ alone would not

[2] There is in fact another difficulty: one should carefully define the scenario in which secrecy is considered. Many parameters such as the number of agents, sessions, or the possibility of corrupt agents, have an impact on secrecy.

be adequate. Without this, secrecy would fail for bad reasons: for instance the secrecy of a name n would not be ensured by the null process (or any process that can reduce to a process where n does not occur anymore) using the closed attacker $\mathbf{new}\ x.\mathbf{out}(c, x)$ and choosing n for the new name.

Example 2.4. *Assume the theory of example 2.2, and consider the following process:*

$$P \stackrel{def}{=} \mathbf{in}(c, x).\mathbf{if}\ x = \mathsf{senc}(\mathsf{sdec}(x, k), k)\ \mathbf{then}\ \mathbf{out}(c, k)\ \mathbf{else}\ \mathbf{out}(c, \mathsf{senc}(n, k))$$

We have that $(\{n, k\}, \{P\})$ *ensures the secrecy of* n*, because the equation* $u =_{\mathsf{E}} \mathsf{senc}(\mathsf{sdec}(u, k), k)$ *holds iff* $u =_{\mathsf{E}} \mathsf{senc}(v, k)$ *for some* v*, and the attacker cannot provide* P *with an input of that form. However,* $(\{n, k\}, \{P, P\})$ *does not ensure the secrecy of* n*, as witnessed by the following attacker process:*

$$A \stackrel{def}{=} \mathbf{out}(c, \mathsf{ok}).\mathbf{in}(c, x).\mathbf{out}(c, x).\mathbf{in}(c, y).\mathbf{out}(\mathsf{bad}, \mathsf{sdec}(x, y))$$

Indeed, we have

$$(\{n, k\}, \{P, P, A\}) \rightsquigarrow^* (\{n, k\}, \{\mathbf{out}(\mathsf{bad}, \mathsf{sdec}(\mathsf{senc}(n, k), k))\})$$

and $\mathsf{sdec}(\mathsf{senc}(n, k), k) =_{\mathsf{E}} n$.

2.4 May-testing

We now turn to defining our reference notion of indistinguishability, through may testing. Intuitively, we would like to declare that two processes are indistinguishable when any experiment yields some outcome with one process iff the same outcome can be obtained with the other process. Our experiments should include communications, message manipulations and tests, which will be obtained by taking processes as experiments. The outcome of experiments could be an output on some channel, but we prefer to take an abstract outcome by enriching processes with a **success** construct — this does not change the resulting equivalence and slightly simplifies the theory. The new construct plays no role in executions but its presence at toplevel in configurations indicates a successful test.

The notion of test will be relative to an environment, indicating which names and channels are private and should thus not be mentioned in tests. It will be useful for technical developments to allow tests to mention past outputs of processes. This is achieved through the notion of *frame*, from which we derive a notion of *configuration* that generalizes internal reduction configurations and on which we define may testing.

Definition 2.3 (Frame)**.** *A* frame*, noted* $E.\sigma$*, is composed of an environment* $E \subseteq_{\mathsf{fin}} \mathcal{N} \cup \mathcal{C}$ *and a mapping* $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{N} \cup \mathcal{C})$ *that respects sorts and has finite domain. Frames will be denoted by* Φ *or* Ψ*.*

A frame $\Phi = E.\sigma$ may be extended in two ways: we write $\Phi \cup \{t\}$ for $(E \cup \{t\}).\sigma$ and we write $\Phi \cup \{w \mapsto t\}$ for $E.(\sigma \cup \{w \mapsto t\})$. Frames are often used as substitutions: we simply write $t\Phi$ for $t\sigma$ and $\mathsf{dom}(\Phi)$ for $\mathsf{dom}(\sigma)$. The first component of a frame Φ is noted $E(\Phi)$.

Definition 2.4 (Configuration). *A* configuration *is a pair* $K = (\mathcal{P}, \Phi)$ *where* Φ *is a frame and* $\mathcal{P}$ *is a multiset of closed processes.*

When K is a configuration, $\Phi(K)$ denotes its frame. In frames and configurations, the sets of private names and channels act as a binder. We thus define

$$\mathsf{fn}(\Phi) \stackrel{def}{=} \bigcup_{x \in \mathsf{dom}(\Phi)} \mathsf{fn}(\Phi(x)) \setminus E(\Phi) \qquad \mathsf{fn}((\mathcal{P}, \Phi)) \stackrel{def}{=} \mathsf{fn}(\Phi) \cup \mathsf{fn}(\mathcal{P}) \setminus E(\Phi)$$

and similarly for $\mathsf{fc}(K)$ and $\mathsf{fnc}(K)$.

Definition 2.5 (Test). *Let* Φ *be a frame. A* Φ*-test is a process* T *that may feature the special* **success** *construct, such that* $T\Phi$ *is closed and the names and channels of* $E(\Phi)$ *do not occur in* T*.*

Definition 2.6 (May testing). *Let* $(\mathcal{P}, \Phi)$ *be a configuration and* T *a* Φ*-test. We say that* $(\mathcal{P}, \Phi)$ passes T*, noted* $(\mathcal{P}, \Phi) \models T$*, when* $(E(\Phi), \mathcal{P} \cup \{T\Phi\}) \rightsquigarrow^* (E', \mathcal{P}')$ *for some* $(E', \mathcal{P}')$ *such that* **success** $\in \mathcal{P}'$*.*

Testing yields a notion of indistinguishability between configurations that have the same sets of candidate tests, which is captured by the following *compatibility* condition.

Definition 2.7 (Compatibility). *We say that two configurations* $(\mathcal{P}, \Phi)$ *and* $(\mathcal{Q}, \Psi)$ *are compatible when* $E(\Phi) = E(\Psi)$ *and* $\mathsf{dom}(\Phi) = \mathsf{dom}(\Psi)$*.*

Definition 2.8 (May testing equivalence). *We say that two compatible configurations* K_1 *and* K_2 *are* may-testing equivalent*, written* $K_1 \approx_m K_2$*, when they satisfy the same tests: for all* $\Phi(K_1)$*-tests* T*, we have* $K_1 \models T$ *iff* $K_2 \models T$*.*

Application to modelling security properties

Process indistinguishability can be used to model various security properties. For instance, we may say that a protocol is anonymous when a session of the protocol corresponding to identity A is indistinguishable from a session of B — to obtain an actual definition, one would have to consider these two sessions in a sufficiently rich context. It is also possible to derive an alternative definition of secrecy: we may say that a configuration $(\mathcal{P}, \Phi)$ ensures the secrecy of s when

$$(\mathcal{P} \cup \{\mathbf{in}(c, x).0\}, \Phi) \approx_m (\mathcal{P} \cup \{\mathbf{in}(c, x).\mathbf{if}\ x = s\ \mathbf{then}\ \mathbf{out}(c, \mathsf{ok})\}, \Phi)$$

for some arbitrary channel c that does not occur in $(\mathcal{P}, \Phi)$.

Interestingly, indistinguishability allows to define a stronger notion of secrecy: instead of requiring that the attacker cannot deduce the secret, we require that the attacker cannot distinguish this term from any other one. The following definition is adapted from [Bla04; Bla16].

Definition 2.9 (Strong secrecy). *Let* $(E, \mathcal{P})$ *be an internal reduction configuration with free variables. We say that it preserves the strong secrecy of its free variables when, for any closed substitutions* θ *and* θ' *of domain* $\mathsf{fv}(\mathcal{P})$ *and such that* $\mathsf{fnc}(\theta, \theta') \cap E = \emptyset$*, we have* $(\mathcal{P}\theta, E.\emptyset) \approx_m (\mathcal{P}\theta', E.\emptyset)$*.*

Example 2.5. *Variable x is not strongly secret in $(\{k\}, \{\textbf{if } x = \text{ok } \textbf{then out}(c, \text{ok})\})$ but it is in $(\{k, c\}, \{\textbf{if } x = \text{ok } \textbf{then out}(c, \text{ok})\})$ and $(\{k, r\}, \{\textbf{if } x = r \textbf{ then out}(c, \text{ok})\})$. Variables x, y are not strongly secret in $(\{k\}, \{\textbf{out}(c, \{x\}_k), \textbf{out}(c, \{y\}_k)\})$ but they are in the randomized version $(\{k, r_1, r_2\}, \{\textbf{out}(c, \{\langle x, r_1\rangle\}_k), \textbf{out}(c, \{\langle y, r_2\rangle\}_k)\})$.*

Linear tests

We have naturally defined may-testing equivalence using the most general class of test at hand, in order to account for all possible adversarial environments. However, a more limited class of tests turns out to already yield the same discriminating power.

Definition 2.10 (Linear test)**.** *We define the syntactic classes of* static tests S *and* linear tests L *by the following grammar, where C and R denote open terms:*

$$\begin{array}{rcl} S & ::= & \textbf{if } R = R' \textbf{ then } S \textbf{ else } 0 \\ & | & \textbf{if } R = R' \textbf{ then } 0 \textbf{ else } S \\ & | & \textbf{success} \\ L & ::= & \textbf{in}(C, x).L \mid \textbf{out}(C, R).L \mid S \end{array}$$

Note that linear tests may feature inputs and outputs of either sort. For example, $\textbf{in}(c, x).\textbf{out}(x, \text{ok}).\textbf{success}$ is a linear test verifying that it is possible to receive a channel x and emit on it.

Theorem 2.1. *Let K_1 and K_2 be two compatible configurations in which* **success** *does not appear as a (sub)process. We have $K_1 \approx_m K_2$ iff K_1 and K_2 satisfy the same linear $\Phi(K_1)$-tests.*

2.5 Labelled transitions

We have seen that internal reduction and may-testing equivalence can be used to model security properties. However, the resulting definitions are not directly amenable to verification: reachability properties such as weak secrecy involve a quantification over arbitrary attackers; equivalence properties such as strong secrecy similarly require to consider all possible tests. As is standard in concurrency theory, we now turn to labelled transition systems (LTS) to clarify the study of the possible interactions of a process with its environment.

We have introduced frames in the previous section to represent the attacker's knowledge, i.e. the past outputs of the protocol. The variables in the domain of a frame are often called *handles* and denoted by w, and they are used by the attacker to indirectly refer to past outputs. Concretely, the attacker derives messages by applying Φ's substitution to terms with variables in $\text{dom}(\Phi)$ that do not feature private names or channels; such terms are called recipes.

Definition 2.11 (Recipe)**.** *Let Φ be a frame. A Φ-recipe is a term of $\mathcal{T}(\text{dom}(\Phi) \cup \mathcal{N} \cup \mathcal{C} \setminus E(\Phi))$. We use the letter R for arbitrary recipes, and C for recipes of sort* channel.

$$(\{0\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\mathcal{P}, \Phi)$$

$$(\{P \mid Q\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{P, Q\} \cup \mathcal{P}, \Phi)$$

$$(\{!P\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{P, !P\} \cup \mathcal{P}, \Phi)$$

$$(\{\mathbf{in}(c, x).P, \mathbf{out}(c, u).Q\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{P\{x \mapsto u\}, Q\} \cup \mathcal{P}, \Phi)$$

$$(\{\mathbf{if}\ u = v\ \mathbf{then}\ P\ \mathbf{else}\ Q\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{P\} \cup \mathcal{P}, \Phi) \qquad \text{when } u =_{\mathsf{E}} v$$

$$(\{\mathbf{if}\ u = v\ \mathbf{then}\ P\ \mathbf{else}\ Q\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{Q\} \cup \mathcal{P}, \Phi) \qquad \text{when } u \neq_{\mathsf{E}} v$$

$$(\{\mathbf{new}\ x.P\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{P\{x \mapsto n\}\} \cup \mathcal{P}, \Phi \cup \{n\}) \text{ when } n \notin E(\Phi) \cup \mathsf{fn}(P, \mathcal{P}, \Phi)$$

$$(\{\mathbf{new}\ x.P\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{P\{x \mapsto c\}\} \cup \mathcal{P}, \Phi \cup \{c\}) \quad \text{when } c \notin E(\Phi) \cup \mathsf{fc}(P, \mathcal{P}, \Phi)$$

$$(\{\mathbf{out}(c, u).P\} \cup \mathcal{P}, \Phi) \xrightarrow{\mathbf{out}(C,w)} (\{P\} \cup \mathcal{P}, \Phi \cup \{w \mapsto u\})$$

$$\text{when } \Phi \vdash^{C} c \text{ and } w \notin \mathsf{dom}(\Phi)$$

$$(\{\mathbf{in}(c, x).P\} \cup \mathcal{P}, \Phi) \xrightarrow{\mathbf{in}(C,R)} (\{P\{x \mapsto R\Phi\}\} \cup \mathcal{P}, \Phi)$$

$$\text{when } \Phi \vdash^{C} c \text{ and } R \text{ is a } \Phi\text{-recipe}$$

Figure 2.2: Labelled transition rules

Definition 2.12 (Deduction). *Let Φ be a frame, and t a closed term. We write $\Phi \vdash^{R} t$ when R is a Φ-recipe such that $R\Phi =_{\mathsf{E}} t$. We say that Φ allows to deduce a term t, and write $\Phi \vdash t$, when there exists R such that $\Phi \vdash^{R} t$.*

Example 2.6. *Let $n, k \in \mathcal{N}$ and $\Phi = \{n, k\}.\{w \mapsto \mathsf{senc}(n, k)\}$. We have $\Phi \vdash m$ for any name $m \notin E(\Phi)$, but $\Phi \nvdash n$. However, $\Phi \cup \{w' \mapsto k\} \vdash n$ by considering the recipe $\mathsf{sdec}(w, w')$.*

The previous two definitions do apply to sort channel, though for this sort the situation is much simpler: a recipe is either a (public) channel or a variable of sort channel; deduction is also simplified in that the equational theory has no impact on terms of sort channel.

Our labelled transitions will take place between configurations $(\mathcal{P}, \Phi)$ as defined in definition 2.4. Actions are of the form τ, $\mathbf{in}(c, R)$ or $\mathbf{out}(c, w)$ where R is a recipe and w is a handle.

Definition 2.13. *The labelled transition relation $\rightarrow$ is given by the rules of figure 2.2.*

Again, these rules must be understood as respecting sorts. This means, for instance, that the rule for $\mathbf{in}(c, R)$ transitions only applies when the active process is of the form $\mathbf{in}(c, x).P$ with x and R of the same sort.

Our LTS features transitions with labels $\mathbf{out}(c, w)$ and $\mathbf{in}(c, R)$ representing communications with the environment. It also allows communications that are internal to the process, with label τ. By inspecting the rules of figure 2.2, one immediately sees that

$(\mathcal{P}, \Phi) \xrightarrow{\tau} (\mathcal{P}', \Phi')$ implies $(E(\Phi), \mathcal{P}) \rightsquigarrow (E(\Phi'), \mathcal{P}')$. The converse does not hold, but only for unimportant technical reasons: when executing a **new** process in an internal reduction $(E, \mathcal{P}) \rightsquigarrow (E', \mathcal{P}')$, the choice of fresh symbol is not necessarily valid wrt. $(\mathcal{P}, E.\sigma)$ where σ is an arbitrary substitution; however, it is always possible to choose a fresh name that is also fresh with respect to the desired σ.

Application to modelling security properties

Using the labelled transition system, we can characterize the notion of secrecy of definition 2.2 without having to explicitly quantify over all attackers.

Lemma 2.1. *The next two statements are equivalent, for any configuration $(\mathcal{P}, \Phi)$ and closed term s:*

(1) There exists a trace t and a configuration $(\mathcal{P}', \Phi')$ such that $(\mathcal{P}, \Phi) \xrightarrow{t} (\mathcal{P}', \Phi')$ and $\Phi' \vdash s$.

(2) There exists a multiset of processes $\mathcal{Q}$ in which the names and channels of $E(\Phi)$ do not occur such that $(E(\Phi), \mathcal{P} \cup \mathcal{Q}\Phi) \rightsquigarrow^* (E', \{\mathbf{out}(c, u)\} \cup \mathcal{P}')$ for some E', $\mathcal{P}'$, $c \notin E'$ and $u =_{\mathsf{E}} s$.

Corollary 2.1. *Let $(E, \mathcal{P})$ be an internal reduction configuration and s be a closed term. We have that $(E, \mathcal{P})$ does not ensure the secrecy of s iff there exist t, $\mathcal{P}'$ and Φ' such that $(\mathcal{P}, E.\emptyset) \xrightarrow{t} (\mathcal{P}', \Phi')$ and $\Phi' \vdash s$.*

In the proof of lemma 2.1, establishing that (2) implies (1) is similar to a step in the proof of theorem 2.1 where linear tests and their underlying static tests are replaced by the trace t and the deducibility of s.

2.6 Trace equivalence

We now build on the LTS to define notions of process indistinguishability. We first introduce static equivalence, which corresponds to indistinguishability of past outputs, without further interaction with the protocol. Then we define trace equivalence, which captures the indistinguishability of protocols when interactions are allowed. As we shall see, trace equivalence coincides with may-testing equivalence under a reasonable condition.

Definition 2.14 (Static equivalence, $\Phi \sim \Psi$). *Let Φ and Ψ be two frames such that*[3] $\mathsf{dom}(\Phi) = \mathsf{dom}(\Psi)$ *and* $E(\Phi) \cap \mathsf{fnc}(\Psi) = E(\Psi) \cap \mathsf{fnc}(\Phi) = \emptyset$.

We say that Φ and Ψ are statically equivalent, *noted $\Phi \sim \Psi$, when for all recipes $R, R' \in \mathcal{T}(\mathsf{dom}(\Phi) \cup \mathcal{N} \cup \mathcal{C} \setminus E(\Phi) \setminus E(\Psi))$, we have $R\Phi =_{\mathsf{E}} R'\Phi$ iff $R\Psi =_{\mathsf{E}} R'\Psi$.*

Example 2.7. *Consider the theory of example 2.2 and take $\Phi := \{k, m, n\}.\{w \mapsto \{n\}_k\}$ and $\Psi := \{k, m, n\}.\{w \mapsto m\}$. We have $\Phi \sim \Psi$ but $\Phi \cup \{w' \mapsto k\} \not\sim \Psi \cup \{w' \mapsto k\}$, by*

[3] Note that the second condition can always be obtained by α-renaming frames, i.e. changing the names of $E(\Phi)$. Without this condition, static equivalence would not be compatible with α-renaming.

considering the recipes w *and* $\mathsf{senc}(\mathsf{sdec}(w, w'), w')$. *The extended frames are however still statically equivalent in the theory extended with the equation* $\mathsf{senc}(\mathsf{sdec}(x, y), y)$ E x, *which expresses that encryption is surjective: intuitively, decryption can never fail in that case, so the attacker has no way to distinguish a ciphertext of an unknown name from another unknown name.*

We observe that static equivalence preserves the satisfaction of static tests, as stated in the next proposition. We write $\Phi \models S$ when $(\emptyset, \Phi) \models S$.

Proposition 2.3. *Let* Φ *and* Ψ *be two frames such that* $\Phi \sim \Psi$. *Let* S *be a static test that is both a* Φ*-test and a* Ψ*-test. We have* $\Phi \models S$ *iff* $\Psi \models S$.

We now seek to express indistinguishability with respect to attackers that can interact with protocols, expressing these interactions concisely as traces in our LTS. We first introduce some notations to focus on the actions that can be observed by the attacker, e.g. exclude τ actions.

Definition 2.15 (Observable action and trace, $\Rightarrow$)**.** *We say that an action is* observable *when it is not* τ, *and that a trace is observable when it contains only observable actions. We write* $K \stackrel{t}{\Rightarrow} K'$ *when* t *is an observable trace, and* $K \stackrel{t'}{\rightarrow} K'$ *for some trace* t' *obtained from* t *by inserting* τ *actions.*

Definition 2.16 (Trace inclusion $\sqsubseteq$, trace equivalence $\approx$)**.** *Let* K_1 *and* K_2 *be two compatible configurations. We say that* $K_1 \sqsubseteq K_2$ *when, for all* t *and* K_1' *such that* $K_1 \stackrel{t}{\Rightarrow} K_1'$, *there exists* K_2' *such that* $K_2 \stackrel{t}{\Rightarrow} K_2'$ *and* $\Phi(K_2') \sim \Phi(K_1')$. *We say that* $K_1 \approx K_2$ *when both* $K_1 \sqsubseteq K_2$ *and* $K_2 \sqsubseteq K_1$ *hold.*

In order to relate trace equivalence and may-testing equivalence, we first need to relate observable traces and linear tests.

Definition 2.17 ($L_t[S]$)**.** *Given an observable trace* t *and a static test* S, *we define the linear test* $L_t[S]$ *as follows:*

$$\begin{aligned} L_{\mathbf{in}(C,R).t}[S] &\stackrel{def}{=} \mathbf{out}(C, R).L_t[S] \\ L_{\mathbf{out}(C,w).t}[S] &\stackrel{def}{=} \mathbf{in}(C, w).L_t[S] \\ L_\epsilon[S] &\stackrel{def}{=} S \end{aligned}$$

Proposition 2.4. *For any* $(\mathcal{P}, \Phi)$, $(\mathcal{P}', \Phi')$, t *and* S *such that* $(\mathcal{P}, \Phi) \stackrel{t}{\Rightarrow} (\mathcal{P}', \Phi')$ *and* $\Phi' \models S$, *we have* $(\mathcal{P}, \Phi) \models L_t[S]$.

Proposition 2.5. *For any* $(\mathcal{P}, \Phi)$, t *and* S *such that* $(\mathcal{P}, \Phi) \models L_t[S]$, *we have* $(\mathcal{P}, \Phi) \stackrel{t}{\Rightarrow} (\mathcal{P}', \Phi')$ *for some* $(\mathcal{P}', \Phi')$ *such that* $\Phi' \models S$.

From these two basic propositions we immediately obtain that trace equivalence implies may-testing equivalence.

Lemma 2.2. *For any compatible configurations* $(\mathcal{P}, \Phi)$ *and* $(\mathcal{Q}, \Psi)$,

$$(\mathcal{P}, \Phi) \approx (\mathcal{Q}, \Psi) \text{ implies } (\mathcal{P}, \Phi) \approx_m (\mathcal{Q}, \Psi).$$

As shown in [CCD13], the converse implication can only be obtained under an additionnal assumption, called *image-finiteness*.

Definition 2.18. *A configuration* $(\mathcal{P}, \Phi)$ *is* image-finite *when, for all observable traces* t*, only finitely many frames can be obtained after executing* t *up to static equivalence, i.e.* $\{\Phi' \mid (\mathcal{P}, \Phi) \stackrel{t}{\Rightarrow} (\mathcal{P}', \Phi')\}/\sim$ *is finite.*

This assumption is quite mild. It immediately holds, in particular, for replication-free configurations. In order to break it, replication must be used to spawn an unbounded number of internal communications to create an infinite non-deterministic choice.

Lemma 2.3. *For any compatible and image-finite configurations* $(\mathcal{P}, \Phi)$ *and* $(\mathcal{Q}, \Psi)$*,*

$$(\mathcal{P}, \Phi) \approx_m (\mathcal{Q}, \Psi) \text{ implies } (\mathcal{P}, \Phi) \approx (\mathcal{Q}, \Psi).$$

Proof sketch. Assume, for the sake of absurdity, that the two configurations are may-testing equivalent but not trace equivalent. We know that the two configurations can execute the same traces because they satisfy the same tests of the form $L_t[\textbf{success}]$. Hence it must be that, starting from one of the configurations, we can obtain a frame that is not statically equivalent to any of the frames that can be obtained from the other configuration. Say, for instance, that we have

$$(\mathcal{P}, \Phi) \stackrel{t}{\Rightarrow} (\mathcal{P}', \Phi') \text{ and, for all } (\mathcal{Q}', \Psi') \text{ such that } (\mathcal{Q}, \Psi) \stackrel{t}{\Rightarrow} (\mathcal{Q}', \Psi'), \Phi' \not\sim \Psi'.$$

Since our configurations are image-finite, there exists $\Psi'_1, \ldots, \Psi'_n$ such that any of the Ψ' above is statically equivalent to some Ψ'_i. In particular, we have $\Phi' \not\sim \Psi'_i$ for all i, thus one can construct a static test S such that $\Phi' \models S$ but $\Psi'_i \not\models S$ for all i. Hence $(\mathcal{P}, \Phi) \models L_t[S]$ but $(\mathcal{Q}, \Psi) \not\models L_t[S]$, which is absurd. □

Corollary 2.2. *Trace equivalence and may-testing equivalence coincide for compatible image-finite configurations.*

Remark 2.2. *In order to simplify our presentation, we have only considered constructor symbols and equations. One might wonder how the previous results change with the addition of destructors symbols in terms, coming with some computation relation and a construction* **let** $x = u$ **in** P **else** Q *for processes which can reduce to* $P\{x \mapsto v\}$ *for any* v *such that* u *computes to* v*, and reduces to* Q *when there is no such* v*.*

If the rewrite relation is convergent up to the equational theory, our development can be adapted. Linear tests have to be enriched with **let** *constructs, static equivalence has to be enriched to account for computability tests (see e.g. [HBD19]), proofs become a little bit more technical, but trace equivalence and may-testing equivalence still coincide on image-finite configurations. However, that result breaks when the computation relation is non-deterministic; we will present a counter-example. To the best of our knowledge, this observation has not been made in the literature, but experts may be well aware of it given that the works considering calculi with destructors restrict to the deterministic case [Bla09; BAF08; ABF17].*

Consider $\Sigma = \{\mathsf{f}, \mathsf{g}, \mathsf{h}, \mathsf{ch}, \mathsf{val}\}$ *where* f*,* g *and* h *are binary constructors coming with an empty equational theory* E*, and the following rewrite rules for the destructors* ch *and* val*:*

$$\begin{array}{llll} \mathsf{ch}(\mathsf{h}(x,y)) & \to & \mathsf{f}(x,y) & \qquad \mathsf{val}(\mathsf{h}(x,y), \mathsf{f}(x,y)) \to \mathsf{h}(x,y) \\ \mathsf{ch}(\mathsf{h}(x,y)) & \to & \mathsf{g}(x,y) & \qquad \mathsf{val}(\mathsf{h}(x,y), \mathsf{g}(x,y)) \to \mathsf{h}(x,y) \end{array}$$

Consider now two processes:

$$P \stackrel{def}{=} \textbf{new}\ n,k.\ \textbf{out}(c,\mathsf{h}(n,k)).\textbf{in}(c,x).\textbf{in}(c,y).\\ \textbf{let}\ \_ = \mathsf{val}(\mathsf{h}(n,k),x)\ \textbf{in let}\ \_ = \mathsf{val}(\mathsf{h}(n,k),y)\ \textbf{in if}\ x=y\ \textbf{then out}(c,n)$$

$$Q \stackrel{def}{=} \textbf{new}\ n,k.\ \textbf{out}(c,\mathsf{h}(n,k)).\textbf{in}(c,x).\textbf{in}(c,y).\\ \textbf{let}\ \_ = \mathsf{val}(\mathsf{h}(n,k),x)\ \textbf{in let}\ \_ = \mathsf{val}(\mathsf{h}(n,k),y)\ \textbf{in if}\ x\neq y\ \textbf{then out}(c,n)$$

The following test is satisfied by P but not by Q:

$$\textbf{in}(c,w).\textbf{let}\ z = \mathsf{ch}(w)\ \textbf{in out}(c,z).\textbf{out}(c,z).\textbf{in}(c,\_).\textbf{success}$$

We show, however, that our processes are trace equivalent. This is easily checked for traces of length at most 4, i.e. for the prefixes of $\textbf{out}(c,w).\textbf{in}(c,R).\textbf{in}(c,R').\tau$. *It is also the case for traces of length at most 6, composed of two more τ actions, because both of our processes can only perform the second (resp. third) τ action when R (resp. R') is* $\mathsf{ch}(w)$. *Indeed, in order to execute a second τ action, we must be able to evaluate $R\Phi$ to either* $\mathsf{f}(n,k)$ *or* $\mathsf{g}(n,k)$, *thus R must be* $\mathsf{ch}(w)$ *where w refers to the initial output of the process. The argument is similar for the third τ.*

The only trace of length seven that could be considered is thus

$$\textbf{out}(c,w).\textbf{in}(c,\mathsf{ch}(w)).\textbf{in}(c,\mathsf{ch}(w)).\tau.\tau.\tau.\textbf{out}(c,w')$$

and this trace can be executed by P and Q, with identical resulting frames. In the case of Q, this is because the two occurrences of $\mathsf{ch}(w)$ *can be evaluated differently, to yield* $\mathsf{f}(n,k)$ *and* $\mathsf{g}(n,k)$ *respectively.*

To bridge this gap between trace equivalence and may-testing equivalence, one would have to enrich the notion of trace to keep track of when and how recipes are evaluated.

Application to modelling security properties

The previous result allows to reformulate all indistinguishability-based security properties (e.g. strong secrecy) in terms of trace equivalence, provided that protocols are modelled as image-finite processes. Less obviously, static equivalence can also be used to model some security properties.

Example 2.8 (Offline guessing [CDE05])**.** *Some protocols use low-entropy secrets (passwords, PIN code, etc.) for which brute force attacks cannot be ruled out. However, such attacks must typically be carried out* offline, *because of limitations on the number or frequency of protocol runs, which means that the attacker needs an offline means of testing whether a guessed value is the actual secret. Resistance against such attacks can thus be modelled as follows: the configuration $(\mathcal{P},\Phi)$ is resistant against the offline guessing of s if, for all t and $(\mathcal{P}',\Phi')$ such that $(\mathcal{P},\Phi)\stackrel{t}{\Rightarrow}(\mathcal{P}',\Phi')$ we have*

$$\Phi' \cup \{n\} \cup \{w \mapsto s\} \sim \Phi' \cup \{n\} \cup \{w \mapsto n\}$$

where $w \in \mathcal{X} \setminus \mathrm{dom}(\Phi')$ and n is a fresh name that does not occur in Φ', which we add to the environment of Φ'.

As reported in [CDE05] a password-based key exchange protocol that would output encryptions using the password as key would not be resistant against offline guessing in settings where the attacker can distinguish a name from a ciphertext using a known key. Formally, such a protocol could only satisfy the above condition in theories with surjective encryption (cf. example 2.7).

2.7 Observational equivalence and bisimilarity

Among the many process equivalences studied in concurrency theory, two more equivalences have been considered in the context of security protocols: labeled bisimilarity and observational equivalence.

Definition 2.19. *Observational equivalence is the largest symmetric relation $\mathcal{R}$ over configurations such that, whenever $(\mathcal{P}, \Phi)\ R\ (\mathcal{Q}, \Psi)$, the following conditions hold:*

1. $(\mathcal{P}, \Phi) \xrightarrow{\mathsf{out}(c,w)}$ *iff* $(\mathcal{Q}, \Psi) \xrightarrow{\mathsf{out}(c,w)}$;
2. *for all $(\mathcal{P}', \Phi')$ such that $(\mathcal{P}, \Phi) \stackrel{\epsilon}{\Rightarrow} (\mathcal{P}', \Phi')$,*
   there exists $(\mathcal{Q}', \Psi')$ such that $(\mathcal{Q}, \Psi) \stackrel{\epsilon}{\Rightarrow} (\mathcal{Q}', \Psi')$ and $(\mathcal{P}', \Phi')\ \mathcal{R}\ (\mathcal{Q}', \Psi')$;
3. *for any environment E' disjoint from $E(\Phi) \cup E(\Psi)$,*
   for any multiset of processes $\mathcal{A}$ not using symbols from $E(\Phi) \cup E(\Psi)$,
   $(\mathcal{P} \cup \mathcal{A}, \Phi \cup E')\ \mathcal{R}\ (\mathcal{Q} \cup \mathcal{A}, \Psi \cup E')$ holds.

The first condition requires that the two configurations can perform the same immediately observable outputs. The second one requires that internal reductions of one configuration (which may change its observable outputs) can be matched by reductions of the other configuration in such a way that the relationship is maintained. The last condition imposes that our relation is reasonably contextual.

Example 2.9. *Consider the following processes, using channels a, b_1 and b_2 and the notation of remark 2.1 for non-deterministic choice:*

$$P \stackrel{def}{=} \big(\mathbf{out}(a, \mathsf{ok}).\mathbf{out}(b_1, \mathsf{ok})\big) + \big(\mathbf{out}(a, \mathsf{ok}).\mathbf{out}(b_2, \mathsf{ok})\big)$$
$$Q \stackrel{def}{=} \mathbf{out}(a, \mathsf{ok}).\big(\mathbf{out}(b_1, \mathsf{ok}) + \mathbf{out}(b_2, \mathsf{ok})\big)$$

The configurations $(\{P\}, \emptyset)$ and $(\{Q\}, \emptyset)$ can perform the same observable traces: $\mathbf{out}(a, w).\mathbf{out}(b_1, w')$, $\mathbf{out}(a, w).\mathbf{out}(b_2, w')$ and their prefixes, for any handles w and w'. They are actually trace equivalent (and thus may-testing equivalent) since static equivalence conditions are obvious.

However, the two configurations are not observationally equivalent. There exists $(\mathcal{Q}, \Psi)$ such that $(\{Q\}, \emptyset) \stackrel{\epsilon}{\Rightarrow} (\mathcal{Q}, \Psi)$, $(\mathcal{Q}, \Psi) \xrightarrow{\mathsf{out}(a,w)}$ and,

$$\text{for each } i \in \{1, 2\},\ (\mathcal{Q} \cup \{\mathbf{in}(a, x)\}, \Psi) \stackrel{\epsilon}{\Rightarrow} \xrightarrow{\mathsf{out}(b_i, w')} .$$

This cannot be matched by P*: for any* $(\{P\},\emptyset) \stackrel{\epsilon}{\Rightarrow} (\mathcal{P},\Phi)$ *such that* $(\mathcal{P},\Phi) \xrightarrow{\mathsf{out}(a,w)}$ *there is some* $i \in \{1,2\}$ *for which* $(\mathcal{P} \cup \{\mathsf{in}(a,x)\},\Phi) \stackrel{\epsilon}{\Rightarrow} \xrightarrow{\mathsf{out}(b_i,w')}$ *does not hold.*

The previous example shows why observational equivalence is stronger than trace equivalence: the former relation forces processes to make early choices, while trace equivalence allows processes to make choices based on the full trace that must be executed.

Definition 2.20. *Bisimilarity in the largest symmetric relation* $\mathcal{R}$ *on configurations such that, whenever* $(\mathcal{P},\Phi)\ R\ (\mathcal{Q},\Psi)$*, the following conditions hold:*

1. $\Phi \sim \Psi$*;*
2. *for all* $(\mathcal{P}',\Phi')$ *such that* $(\mathcal{P},\Phi) \xrightarrow{\tau} (\mathcal{P}',\Phi')$*,*
   there exists (Q',Ψ') *such that* $(\mathcal{Q},\Psi) \stackrel{\epsilon}{\Rightarrow} (\mathcal{Q}',\Psi')$ *and* $(\mathcal{P}',\Phi')\ \mathcal{R}\ (\mathcal{Q}',\Psi')$*;*
3. *for all* α *and* $(\mathcal{P}',\Phi')$ *such that*[4] $\mathsf{fnc}(\alpha) \cap E(\Psi) = \emptyset$ *and* $(\mathcal{P},\Phi) \xrightarrow{\alpha} (\mathcal{P}',\Phi')$*,*
   there exists (Q',Ψ') *such that* $(\mathcal{Q},\Psi) \stackrel{\alpha}{\Rightarrow} (\mathcal{Q}',\Psi')$ *and* $(\mathcal{P}',\Phi')\ \mathcal{R}\ (\mathcal{Q}',\Psi')$*.*

Bisimilarity obviously implies trace equivalence, but the opposite does not hold in general, as witnessed, again by the processes of example 2.9. However, it has been shown [CCD13] that the implication holds for *determinate* configurations, i.e. the configurations such that, for any trace t, the execution of t can only yield one configuration up to some equivalence — which may be trace equivalence or observational equivalence without any impact on the resulting definition.

Bisimilarity and observational equivalence have been shown to coincide for the full applied pi-calculus [ABF17]. As a corollary, bisimilarity is contextual in the sense of condition 3, definition 2.19 — the same holds for trace equivalence on image-finite configurations, as a by-product of corollary 2.2.

In the context of security protocols, modeling indistinguishability as observational equivalence rather than may-testing equivalence is too strong. Indeed, inequivalence does not imply the existence of an attacker that could effectively distinguish whether it is interacting with one process or another. However, it can be easier to verify that two processes are bisimilar rather than trace equivalent, roughly because bisimulation forces immediate choices[5].

[4] The executability of α by $(\mathcal{P},\Phi)$ imposes that the action does not mention any symbol of $E(\Phi)$, but we need to explicitly impose the same condition wrt. $E(\Psi)$, otherwise the equivalence would break for irrelevant reasons. We do not have a similar condition for trace equivalence because we have defined it for compatible configurations. That is not possible with bisimilarity since compatibility is not preserved by simultaneous executions of observable actions.

[5] The difference leads to different complexity classes for finite state processes: bisimulation is in PTIME but trace equivalence is PSPACE-complete [HS96]. However, with the added difficulties of the applied pi-calculus, the complexity analysis does not show a difference anymore: both bisimulation and trace equivalence become coNEXP-complete [CKR18].

2.8 Verification Techniques

The idea of applying formal methods to verify security protocols using symbolic models is several decades old [DY81]. Successful tools for automatically analyzing protocols in such settings have started to emerge a bit later [Low96] and have become much more powerful since then, notably covering larger classes of attackers and supporting more primitives [Bla01; Arm+05; Cre08; Mei+13]. Reachability properties such as secrecy, authentication and other correspondence properties have kept researchers busy for decades. More recently, equivalence properties have received a lot of attention, due to new technical challenges and increasing concerns over privacy in our age of surveillance capitalism and digital state surveillance. We present next an overview of the state of the art of this specific problem: automated verification of security protocol *equivalences* in the *symbolic* model. We shall broadly distinguish two classes of systems, based on their ability to analyze protocols with unbounded sessions, i.e. processes with replication.

Bounded sessions

Several tools implement *decision procedures* for various process equivalences for a bounded number of sessions. Even when the length of traces is bounded in this way, there are still infinitely many traces due to the arbitrary recipes that may be used in input actions. This *prolific attacker* problem can be tackled in various ways, which has a lead to a variety of tools, notably Akiss [Cio+12], SPEC [TNH10], Deepsec [Che+18; CKR18] and SAT-Equiv [CDD18]. Before presenting them below in more details, we can already point out their common strengths and weakenesses. A clear strength of these tools is that they are fully automated. Leaving aside the possibility that the analysis does not terminate in a reasonable amount of time, these tools always conclude with a proof or an attack, which can immediately be understood by the user. The major downside, of course, is that they can only analyze scenarios with a bounded number of sessions. Another problem is that verification time increases sharply when scenarios are enriched, at least for the tools that rely on an exploration of all possible symbolic traces, i.e. all of the above mentioned systems except SAT-Equiv. This has been mitigated with the development and integration of partial-order reduction techniques (cf. chapter 4).

The system Akiss [Cio+12] implements a semi-decision procedure based on a logical encoding of (an approximation of) trace inclusion. In order to decide $P \sqsubseteq Q$, each symbolic trace of P is considered separately, up to partial-order reductions. For each trace, a set of Horn clauses is produced that abstractly describes the possible choices and observations of the attacker for that trace. A resolution-based semi-algorithm is then used to extract from the clauses a finite symbolic description of all possible traces of one process, which can then be tested against the other process. When primitives are described through a subterm convergent equational theory, this procedure has been shown to terminate [Cha+16]. Moreover, the approximation of trace inclusion becomes exact if processes are determinate. Hence the tool provides an exact decision procedure in such cases. An initial limitation of the procedure behind Akiss is that it only supports conditional with equality tests and no **else** branches (so no disequality checking) but this limitation has been lifted later on [GK17]. A strength of the Akiss system is that it

supports a large class of primitives: the only requirement is that primitives are specified through rewrite rules that have the finite variant property [CD05], which allows to model standard primitives such as symmetric and public-key encryption, signatures, but also more advanced ones such as blind signatures. The exclusive or primitive does not fall in that category due to its associativity and commutativity, but a specific extension of the Akiss system provides support for it [Bae+17].

The system SPEC [TNH10; TNH16] implements a decision procedure [TD10] for bisimilarity in the spi-calculus, an ancestor of the applied pi-calculus that supports a fixed set of standard primitives. Like the original procedure behind Akiss, it only supports equality testing, not disequality. The system relies on a dedicated decision procedure [TD10] which analyzes all symbolic traces, and is implemented using the logic programming system Bedwyr [Bae+07] (on which I have worked with Alwen Tiu and others) to help with some high-level optimizations.

Finally, the tool Deepsec [Che+18; CKR18] implements decision procedures for trace equivalence and variants of it. It is the successor of Apte [Che13; Che14] which it improves thanks to insights from Akiss as well as low-level implementations. The procedure used by Deepsec considers all symbolic traces, and for each one computes the possible execution results for each process. This gives rise to two sets of frames with associated constraint systems, which need to be solved in order to check that each solution on one side is matched by a solution on the other side. The system features the most advanced partial order reductions: their integration is less superficial than in Akiss, and notably interacts with constraint solving to perform more reductions based on data (in)dependencies; moreover, Deepsec features some symmetry elimination techniques that further improve verification time on the many case studies featuring several copies of a role. Deepsec allows the user to model primitives using a subterm-convergent rewrite system, which covers a wide range of primitives, only leaving out the most problematic ones such as blind signatures or exclusive or.

We conclude this section by mentioning SAT-Equiv [DCD17; Cor+17; CDD18], which implements a very different procedure. It proceeds by reducing trace equivalence to the standard model-checking techniques of graph planning and SAT solving. This is made possible by a typing result which allows to bound the size of recipes when looking for attacks, which constrains the class of protocols that can be analyzed. Besides this limitations, SAT-Equiv supports a fixed set of standard primitives (symmetric and asymmetric encryption, signatures and hashes) and does not feature **else** branches. Yet, this is enough to cover many case studies, on which SAT-Equiv significantly outperforms the above-mentionned tools [CDD18]. The procedure of SAT-Equiv scales much more gracefully than its competitors, in part because it does not rely on an exploration of symbolic traces, which remains costly even with partial-order reduction techniques.

Unbounded sessions

Verifying equivalences for unbounded sessions is significantly harder than for bounded sessions: there is little hope to obtain even semi-decision procedures. However, some successful tools exist which are based on algorithms that can prove equivalences in practically relevant cases.

Proverif [Bla+01; Bla01] and Tamarin [Bas+12; Mei+13] are two systems that have initially been developed to prove reachability properties of security protocols with unbounded sessions. These two systems follow very different approaches. The former is based on the applied pi-calculus, while the latter is primarily based on multiset rewriting, though it includes a compiler from the applied pi-calculus to multiset rewriting, called SAPIC [KK16]. Proverif's verification technique is based on a translation of processes to Horn clauses, to which a dedicated resolution-based satisfiability checking is applied. Tamarin, on the other hand, performs a backward reachability analysis using symbolic protocol executions and constraint solving. Due to its approach, Proverif is imprecise: it may sometimes return with no clear answer, neither being able to prove the property nor to find an attack against it. In cases where automated proofs are not obtained, Tamarin can be guided either with the definition of intermediate lemmas, or by specifying proof steps much like in interactive theorem proving. Guidance through intermediate lemmas is the subject of ongoing work[6] in Proverif. Finally, the two tools support different cryptographic primitives: both tools allow user-defined primitives via equations or rewrite rules subject to some constraints, but Tamarin also features builtin support for some primitives that Proverif does not fully handle, such as Diffie-Hellman exponentiation and exclusive or.

Both Proverif and Tamarin have been extended with the ability to prove a strong form of equivalence, called diff-equivalence in both cases [BAF08; BDS15b]. The basic idea of diff-equivalences is that it is often useless to consider general equivalence problems, since equivalence-based security properties are often expressed as equivalences between processes that share the same structure, only differing in a few messages. Moreover, it is often the case that these two similar processes are equivalent for the simple reason that each execution trace of one process can be matched by the "same" trace of the other process; in other words, there is no need to ponder choices in the processes' execution. The precise definitions of diff-equivalences differ in Proverif and Tamarin, with an impact on the processes that are related, but both diff-equivalences are strictly (much) stronger than observational equivalence (and thus than trace equivalence). This approach has proved effective to prove strong secrecy or anonymity for various protocols, but the constraints imposed by diff-equivalence are sometimes too strong, as is the case with unlinkability (cf. chapter 3).

Other lines of work have been explored to establish equivalences for unbounded sessions. We note in particular the work of Cortier et al. [Cor+17], which provides a type system for proving equivalences: although the technique is, as the previous ones, not complete, it has brought its own unique successes.

[6] Personal communication with Vincent Cheval, October 2020.

3

Modelling and Verifying Unlinkability

Unlinkability is defined informally in the ISO 15408 standard [09] as follows: the property holds when a user may make multiple uses of a service or resource without others being able to link these uses together. This is a strong notion of privacy, which implies anonymity and more generally ensures that users cannot be tracked — unlinkability is actually sometimes called untraceability. It is particularly desirable in mobile telephony and RFID applications such as electronic passports, contactless payment, etc. Unlinkability is an interesting case study both in terms of formal modelling and verification. We present and compare several formal definitions of unlinkability in section 3.2. We then explain in section 3.3 why it is not currently possible to directly verify strong forms of unlinkability using current automatic tools, and present some conditions that imply unlinkability and can be (mostly) automatically verified.

This chapter is based on [HBD16; HBD19] where Hirschi, Delaune and myself have introduced sufficient conditions for strong unlinkability, and [BDM20] where Delaune, Moreau and myself extend this line of work to protocols with various forms of state.

3.1 Definitions

We describe informally the class of protocols that we shall consider, and the extensions of the process algebra of chapter 2 on which they rely. Although the notion of unlinkability may be considered for protocols with more than two roles (see, e.g., [Ara+10]) we only consider the case of two-party protocols. Since most of our case studies involve tags and readers, we shall use this terminology to refer to our roles.

In our protocols, tags own secret information that may be used to identify them. This information, which we call identity parameters, may be available in different forms to readers, and may even not be available at all, depending on the protocol. The most common situation is the one where each reader session has access to the identity

parameters of all tags. It is the case of RFID authentication protocols, where readers are used to control access to some building, track items in shops or factories, etc. Sometimes, it makes sense to consider reader sessions that are specific to a single identity. This is the case in some e-passport protocols which are used after an optical scan of the passport that we typically do not model: after this scan, the reader expects to interact with the passport whose key has been optically read. As pointed out in [JW09] such scenarios could also be relevant to account for the possibility of tracking a user's set of tags based on how they might be recognized or not by readers from several systems. For instance, employees of a company might have access to some buildings and not others. Finally, it is sometimes the case that readers do not need any access to identity parameters, e.g. if they can verify credentials based on a trusted authority.

Example 3.1. *The OSK protocol [OSK+03] involves tags that store a private key, updated by iterating some hash function* h*, and communicated to readers using another hash function* g*. Each session of a tag reads from its memory the current key* k*, writes* $\mathsf{h}(k)$ *in memory, and sends* $\mathsf{g}(\mathsf{h}(k))$*. Readers in the OSK protocol have access to a shared database of all known keys. Upon receipt of a message* m*, a reader session will determine whether* $m = \mathsf{g}(\mathsf{h}^n(k))$ *for some key* k *from the database and some* $n \in [0; b]$ *for some bound* b *that is a parameter of the protocol definition. If such a* k *is found, the tag is recognized and the database entry* k *is replaced by* $\mathsf{h}^{n+1}(k)$*. Otherwise, the tag is rejected.*

We now describe in more details how this variety of protocols can be described formally, illustrating each ingredient on the previous example.

Tag processes. We assume that the tag role is given as a closed process $T(\vec{k})$ whose free names $\vec{k}$ will be shared by all sessions of the same tag. These names are called the *identity parameters* of a tag, and are typically used to represent a tag's secret key. Of course, each tag session can generate its own nonces, using **new** constructs in $T(\vec{k})$.

Depending on the protocol, multiple sessions of the same tag (i.e. multiple copies of $T(\vec{k})$) may or may not be allowed to run concurrently. We thus consider process algebras extended with a weak form of sequential composition $P; Q$ and a repetition operator $!^{\mathsf{i}}\, P$ which roughly stands for $P; P; \ldots$ but with the ability to abort each session P. We assume that the protocol definition comes with the choice of an operator $\dagger_T \in \{!, !^{\mathsf{i}}\}$ so that $\dagger_T T(\vec{k})$ represents multiple sessions of the same tag: these sessions can run concurrently when $\dagger_T = !$, but only sequentially when $\dagger_T = !^{\mathsf{i}}$.

In some protocols, tags have an internal memory which is mutated from one session to the next. We model such protocols in [BDM20] using an enriched process algebra featuring mutable cells identified by references r. In such cases, each tag has access to a single tag reference containing its identity parameters: the tag process is of the form $T(r)$.

Example 3.2. *The tag process for the OSK protocol is:*

$$T(r) \stackrel{def}{=} \mathbf{get}(r, y).\mathbf{set}(r, \mathsf{h}(y)).\mathbf{out}(c_T, \mathsf{g}(y))$$

Concurrent tag sessions are not authorized for this protocol, hence we take $\dagger_T = !^{\mathsf{i}}$.

Reader processes. Depending on the protocol under consideration, we may consider reader roles of different forms. *Identity-specific* readers $R(\vec{k})$ may be considered when each session of the reader is meant to interact only with tags of a specific identity $\vec{k}$. When readers are *identity-generic*, we use a parameter-less process R. In order to model the database that most generic readers use, we consider in [BDM20] an extended process algebra whose new constructs will be illustrated in the next example.

As with tags, reader sessions may or may not run concurrently depending on the protocol, and we allow this choice to be expressed via an operator $\dagger_R \in \{!, \mathsf{i}\}$.

Example 3.3. *The OSK protocol of example 3.1 has identity-generic readers. In order to model them we introduce destructors* TestOSK *and* UpdateOSK *equipped with the following reduction rules for all* $n \in [0; b]$*:*

$$\mathsf{TestOSK}(y, \mathsf{g}(\mathsf{h}^n(x))) \to \mathsf{ok} \qquad \mathsf{UpdateOSK}(\mathsf{g}(x)) \to \mathsf{h}(x)$$

The reader process is then as follows:

$$R \stackrel{def}{=} \begin{array}{l} \mathbf{in}(c_T, x). \\ \mathbf{lookup}\ y\ \mathbf{such\ that}\ z = \mathsf{TestOSK}(y, x), y' := \mathsf{UpdateOSK}(x)\ \mathbf{in} \\ \mathbf{out}(c_R, \mathsf{ok})\ \mathbf{else\ out}(c_R, \mathsf{error}) \end{array}$$

The **lookup** *construct attempts to find some database entry* y *for which* $\mathsf{TestOSK}(y, x)$ *computes successfully, and it replaces this database entry with the value* y' *obtained as the result of the evaluation of* $\mathsf{UpdateOSK}(x)$*. Our reader process issues messages that explicitly indicate if the protocol succeeds or not: this is good practice since the outcome of authentication protocols is usually observable.*

Communication between tags and readers. The precise channels used in the tag and reader processes are not important for our technical development. The previous examples use the convention from [BDM20] where tags and readers respectively emit on distinct channels c_T and c_R, but a different convention is used in [HBD16; HBD19]. In both cases, however, we rule out private channels so that all sessions of a same role use the same channels. We will thus work with processes that are not determinate: we will discuss below in which cases this is (in)essential.

We assume that the tag and reader processes can interact successfully when they share the same identity. In order to conveniently express and use this natural assumption, we impose a few restrictions on the shape of our two role processes. We view the **else** branches of conditionals as error branches, and restrict them to a single output. Moreover, we assume that one role process consists of a succession of blocks composed of an input followed by some tests and an output — tests may be conditionals, database lookups but also the evaluation of destructors through **let**. The other role must start with an output and then follow the same structure. We finally assume that when a tag and reader processes with the same identity parameters have an *honest interaction* where each output of one role is forwarded to the input of the other role, all tests are successful, so that the execution continues until both roles successfully complete.

Remark 3.1. *As reported in [Ara+10], the French implementation of the BAC e-passport protocols is linkable to the disclosure of too much information through its error messages. At some point of the protocol the reader has to perform two tests: the first one ensures that the received message originates from a tag with the expected identity, while the second avoids replays from past sessions of that tag. The French implementation of that protocol issued different error messages in these two cases, which makes it possible to track a passport. The syntactic constraints that we impose on our processes happen to forbid this kind of problem: indeed, we impose that all tests are performed at once with a single* **else** *branch.*

Initialization. Some initialization procedure is required when readers rely on a shared database, and when tags rely on a mutable state. We thus assume a process $I(r, \vec{k})$ which is in charge of properly initializing, for a new tag of identity $\vec{k}$, both the readers' database and the tag's memory r.

Example 3.4. *For the OSK protocol, the initialization process takes the initial key k of the tag, sets it in the tag's memory r and inserts it in the reader's database:*

$$I(r,k) \stackrel{def}{=} \mathbf{set}(r,k).\mathbf{insert}(k)$$

3.2 Modelling unlinkability

Two main approaches to modelling unlinkability have been followed in the literature. We first present the approach based on an ideal functionality, which we follow in our work. Then we briefly comment on the other, game-based approach.

3.2.1 Definitions based on ideal functionality

Arapinis et al. [Ara+10] have proposed notions of unlinkability for a large class of protocols featuring an arbitrary number of roles and which can be expressed using the full expressivity of the applied pi-calculus.

They first consider a notion of *weak unlinkability* which attempts to capture the following intuition: "an attacker cannot tell when two transitions of a trace are initiated by the same user". They thus consider a process which models all possible uses of the protocol, with an arbitrary number of sessions of each role, and require that:

- for every execution trace t of this process with n observable actions,
- for every i and j such that $i \leq j \leq n$ and the i^{th} and j^{th} observable actions of t correspond to the same role R,
- there exists an indistinguishable trace t' such that the i^{th} and j^{th} observable actions of t' do not correspond to different sessions of the role R with the same identity.

In other words, weak unlinkability implies that the attacker can never be sure that he is looking at two actions that originate from two different sessions of the same user. It remains possible for the attacker to tell that two actions correspond to the same session — for example, the protocol might feature explicit session identifiers, which indeed does not contradict unlinkability.

An undesirable aspect of weak unlinkability is that it is expressed in an ad-hoc fashion, rather than as a process equivalence which could be proved using standard tools. The authors of [Ara+10] address this problem by introducing a second notion, *strong unlinkability*. Roughly, they declare that a role R is strongly unlinkable in some protocol when observational equivalence holds between two processes: the first one allowing an arbitrary number of identities, and for each one an arbitrary number of sessions of each role; the second process, however, can only perform a single session of role R for each identity. In other words, they are asking that the *real* uses of role R are indistinguishable from *ideal* uses where each new session corresponds to a new identity.

It is shown [Ara+10, Theorem 1] that weak unlinkability does not imply strong unlinkability. A counter-example protocol is given, with tag and reader roles, such that the reader emits a particular message to signal when it sees twice the same tag[1]. Unlike the authors of [Ara+10] we consider that this protocol suffers from an actual linkability attack: the reader can be used to track a tag by identifying when it is used again. Hence we consider that weak unlinkability is too weak. Unfortunately, strong unlinkability is also too strong because it is based on observational equivalence, as we shall see in the next section after having given a formal definition.

We now introduce the two notions of unlinkability that we have worked on, both of which are strongly inspired by the strong unlinkability of [Ara+10].

Identity-specific readers

We have first considered in [HBD16; HBD19] some protocols without a database nor states, and with identity-specific readers. In that setting, we have considered the following notion of unlinkability expressed in terms of trace equivalence.

Definition 3.1 ([HBD19, Definition 10])**.** *The protocol is said to be* identity-specific unlinkable *when the following trace equivalence holds:*

$$\begin{array}{rl} & !\,\mathbf{new}\ \vec{k}.\ \left(\dagger_T T(\vec{k}) \mid \dagger_R R(\vec{k})\right) \\ \approx & !\,\mathbf{new}\ \vec{k}.\ \left(\quad T(\vec{k}) \mid \quad R(\vec{k})\right) \end{array}$$

The process on the left of this equivalence is called *multiple-session* process. The process on the right, corresponding to an ideal situation, is called the *single-session* process.

Example 3.5 ([HBD19, Examples 9 and 15])**.** *We consider a toy protocol, where tags are stateless and readers are identity-specific, given informally in Alice & Bob notation*

[1] The precise counter-example of [Ara+10] uses private channels, but its general idea can be replicated without such means. In fact, we give in [BDM20, Appendix A.A] a protocol that is only weakly unlinkable, without using a database, mutable state nor private channels.

as follows:

$$
\begin{array}{ll}
1. & R \to T : \quad n_R \\
2. & T \to R : \quad n_T \\
3. & R \to T : \quad \mathsf{mac}(\langle n_R, n_T \rangle, k) \\
4. & T \to R : \quad \mathsf{mac}(\langle n_T, n_R \rangle, k)
\end{array}
$$

This protocol uses a MAC (Message Authentication Code) primitive, and the tags and readers share a MAC key k as a single identity parameter. The nonces n_T and n_R are generated respectively at the beginning of each session of the tag and reader. To avoid an obvious reflection attack (where $n_T = n_R$) we assume that the reader systematically checks that the first message it receives is not the one he sent initially. The protocol is formally given by the following processes, using the expected signature and equational theory where, in particular, the mac *symbol is free:*

$$
\begin{array}{lll}
R(k) & \stackrel{def}{=} & \mathbf{new}\ n_R.\mathbf{out}(c_R, n_R).\mathbf{in}(c_R, x_1). \\
& & \mathbf{if}\ x_1 \neq n_R\ \mathbf{then} \\
& & \quad \mathbf{out}(c_R, \mathsf{mac}(\langle n_R, x_1 \rangle, k)).\mathbf{in}(c_R, x_2). \\
& & \quad \mathbf{if}\ x_2 = \mathsf{mac}(\langle x_1, n_R \rangle, k)\ \mathbf{then}\ \mathbf{out}(c_R, \mathsf{ok})\ \mathbf{else}\ \mathbf{out}(c_R, \mathsf{error}) \\
& & \mathbf{else}\ \mathbf{out}(c_R, \mathsf{error}) \\
T(k) & \stackrel{def}{=} & \mathbf{in}(c_T, y_1).\mathbf{new}\ n_T.\mathbf{out}(c_T, n_T).\mathbf{in}(c_T, y_2). \\
& & \mathbf{if}\ y_2 = \mathsf{mac}(\langle y_1, n_T \rangle, k)\ \mathbf{then} \\
& & \mathbf{out}(c_T, \mathsf{mac}(\langle n_T, y_1 \rangle, k))
\end{array}
$$

The protocol is unlinkable according to definition 3.1 when $\dagger_R = \mathsf{i}$. Unlinkability fails otherwise since it is possible only with the multiple-session process to have a successful interaction between two sessions of the reader role with the same identity: nonces n_R and n'_R will be exchanged, after which $\mathsf{mac}(\langle n_R, n'_R \rangle, k)$ will be sent by the first reader; then the messages $\mathsf{mac}(\langle n'_R, n_R \rangle, k)$ and ok *can be obtained with the multiple-session process, when the second reader also has identity k, but this is impossible with the single-session process.*

We have shown in [HBD19] that the strong unlinkability of [Ara+10], when instantiated in the natural way for two-party protocols such as ours, is equivalent to definition 3.1 where trace equivalence is replaced by observational equivalence.

The difference between trace equivalence and observational equivalence is crucial here: in fact, the property based on observational equivalence cannot hold for most protocols! We sketch below the argument for the impossibility, considering an arbitrary protocol that is assumed to satisfy two mild conditions:

1. The honest interaction between a tag and a reader can only succeed when they share the same identity. Moreover, the adversary can distinguish between the success and the failure of such an interaction, e.g. by recognizing special public constants[2].

[2] As mentioned before, we consider this assumption to be reasonable because the outcome of protocols can often be observed in practice. This is not always the case: see, e.g. the Private Authentication protocol [Aba02]. Moreover, it might be useful to make a distinction between a physical observation

2. The interaction is initiated by the reader role.

Recall that observational equivalence coincides with labelled bisimilarity, and suppose that the multiple and single-session processes of our protocol are bisimilar. The multiple-session process may start by spawning a new reader session, with some identity $\vec{k}$, which will output a message. Then, a session of tag $\vec{k}$ may be spawned, and the first output of the reader may be forwarded to the first input of the tag. Since the multiple and single-session processes are bisimilar, there must be a way to match this sequence of observable actions in the single-session process so that bisimilarity is preserved. The output of the reader can be matched wlog. by the output of some reader of identity $\vec{k}'$ on the single-session side. Then the outputted message must be forwarded to the first input of a new tag session with some identity $\vec{k}'_1$.

- If $\vec{k}' \neq \vec{k}'_1$, bisimilarity cannot be preserved because a full honest interaction can take place between the reader and tag session in the multiple-session side, but not in the single-session one. By hypothesis, the difference will be observable.
- If $\vec{k}' = \vec{k}'_1$, consider continuing the execution of the multiple-session process by performing enough dummy interactions with the tag to make it fail (which will easily be matched on the single-session side) and then initiating a new session of tag $\vec{k}$, to which the first output of the reader is passed. This must be matched on the single-session side by a tag session with identity $\vec{k}'_2 \neq \vec{k}'$, which cannot preserve bisimilarity: indeed, a honest interaction can take place in the multiple-session process but not in the single-session one.

A related point is that it seems impossible to modify our multiple and single-session processes to obtain determinate processes, without trivializing the resulting notion of unlinkability.

Identity-generic readers

We have then considered, in [BDM20], the case of protocols with generic readers in which readers (resp. tags) may rely on a database (resp. state). In that setting, we force $\dagger_R = !$ and $\dagger_T = ¡$ as it is the most reasonable choice, which fits all the case studies that we have considered.

Definition 3.2 ([BDM20, Definition 5])**.** *The protocol is said to be* identity-generic unlinkable *when the following trace equivalence holds:*

$$\begin{aligned} & !\,R \mid !\,\mathbf{new}\ r, \vec{k}.\ \big(I(r,\vec{k});\ ¡\,T(r)\big) \\ \approx\ & !\,R \mid !\,\mathbf{new}\ r, \vec{k}.\ \big(I(r,\vec{k});\ \ T(r)\big) \end{aligned}$$

of the outcome of the protocol (e.g. a door opens) and an observation through electronic communications (e.g. in a man-in-the-middle attack). Filimonov et al. [Fil+19] seem to make such a distinction. They consider that the failure of bisimilarity-based unlinkability that we describe is an attack, and propose to fix protocols by encrypting their final message to render successful and failing interactions indistinguishable.

Example 3.6. *The OSK protocol is not unlinkable according to our definition. As reported in [JW09] an adversary can desynchronize a tag's memory and the readers' database by querying the same tag b times and dropping the tag's outputs. After this, the reader will reject messages from new sessions of the same tag. Such a desynchronization attack immediately yields a counter-example trace for our identity-generic unlinkability: after dropping b tag outputs and forwarding the next one to a reader, the adversary can observe an error only in the multiple-session scenario.*

We observe in [BDM20] that OSK suffers from linkability issues even in the idealized case where b is set to $+\infty$. Indeed, the attacker can simply trigger two sessions of the same tag, obtaining the messages $\mathrm{g}(k)$ and $\mathrm{g}(\mathrm{h}(k))$, before feeding these messages in reverse order to reader sessions: the first reader session will accept $\mathrm{g}(\mathrm{h}(k))$ and replace k with $\mathrm{h}(k)$ in its database, hence the second session will reject $\mathrm{g}(k)$. This yields another failure of identity-generic unlinkability.

We have finally shown in [BDM20] that the OSK protocol is unlinkable according to our definition when $b = +\infty$ and when readers do not update their databases.

As in definition 3.1, the processes used in definition 3.2 are not determinate. This is however not essential: we shall see in section 3.3.2 that identity-generic unlinkability can be expressed using some form of diff-equivalence; it could also be rephrased in terms of observational equivalence, or using determinate processes, without becoming unrealistically strong.

Comparisons

Our identity-generic unlinkability can be compared with weak unlinkability and identity-specific unlinkability: it sits in between these two notions and is the most appropriate definition when readers are generic.

It can be shown that identity-generic unlinkability is stronger than weak unlinkability [BDM20, Proposition 1] and that this implication is strict. In a nutshell, we provide an artificial protocol [BDM20, Example 16] where the reader can tell that two tag sessions out of three have the same identity, but cannot tell which two, hence identity-generic unlinkability fails but weak unlinkability holds.

We observe next that identity-specific unlinkability can fail for protocols which are identity-generic unlinkable. The next example illustrates this general phenomenon[3] on a real protocol.

Example 3.7. *We consider the Basic Hash protocol of [BCH10], which is an abstraction of the randomized hash-based access-control protocol proposed in [Wei+04]. The protocol has generic readers with access to a database which is never modified. It is*

[3] Juels & Weis make a similar observation in [JW09] when they consider cross-reader privacy of the Hash Lock protocol.

defined as follows in our setting[4]*, where* eq *is a binary destructor testing for equality*[5]*:*

$$
\begin{aligned}
T(r) &\overset{def}{=} \textbf{new}\ n.\ \textbf{get}(r, y).\textbf{out}(c_T, \langle n, \mathsf{h}(n, y)\rangle)\\
I(k) &\overset{def}{=} \textbf{set}(r, k).\textbf{insert}(k)\\
R &\overset{def}{=} \textbf{in}(c_T, x).\textbf{lookup}\ y\ \textbf{such that}\ \_ = \mathsf{eq}\big(\mathsf{snd}(x),\ \mathsf{h}(\mathsf{fst}(x), y)\big),\ y' := y\\
&\quad\ \textbf{then out}(c_R, \mathsf{ok})\ \textbf{else out}(c_R, \mathsf{error})
\end{aligned}
$$

For each new session, the tag sends a nonce n *together with the hash of* n *using the tag's private key. Readers use their database to verify that the received messages are authentic. The protocol obviously does not prevent replay attacks, but it is unlinkable according to definition 3.2.*

Consider now a model of the same protocol but with identity-specific readers:

$$
\begin{aligned}
T'(k) &\overset{def}{=} \textbf{new}\ n.\ \textbf{out}(c_T, \langle n, \mathsf{h}(n, k)\rangle)\\
R'(k) &\overset{def}{=} \textbf{in}(c_T, x).\textbf{if}\ \mathsf{snd}(x) = \mathsf{h}(\mathsf{fst}(x), k)\ \textbf{then out}(c_R, \mathsf{ok})\ \textbf{else out}(c_R, \mathsf{error})
\end{aligned}
$$

The protocol is not unlinkable according to definition 3.1 because of the replay attack: forwarding a tag's output to two reader sessions can yield ok *in the multiple-session scenario but not in the single-session one where a single reader is available for any given identity.*

The previous example shows that the right definition of unlinkability depends on the intended use case: the Basic Hash protocol is meant to have generic readers in practice, and its identity-specific model yields a false attack that is avoided with the proper identity-generic model.

We finally show that, in some sense, identity-specific unlinkability implies identity-generic unlinkability. Given a stateless protocol Π with identity-specific readers, $\dagger_R = !$ and $\dagger_T = \mathsf{i}$, we can define a protocol Π^* with identity-generic readers as follows:

$$
\begin{aligned}
I^*(r, \vec{k}) &\overset{def}{=} \textbf{set}(r, \vec{k}).\textbf{insert}(\vec{k})\\
T^*(r) &\overset{def}{=} \textbf{get}(r, \vec{k}).T(\vec{k})\\
R^* &\overset{def}{=} \textbf{lookup}\ \vec{k}\ \textbf{such that}\ x = \mathsf{ok}, y' := y\ \textbf{in}\ R(\vec{k})
\end{aligned}
$$

In other words, R^* non-deterministically picks some identity parameters $\vec{k}$ from the database, and then executes as the identity-specific reader $R(\vec{k})$. We have shown in [BDM20, Proposition 2] that Π^* satisfies our identity-generic unlinkability when Π satisfies identity-specific unlinkability. The converse should not hold in general, because the multiple-session process of definition 3.2 for Π^* still allows multiple reader sessions — note, however, that example 3.7 is not a counter-example since, if we call Π_g and Π_s the identity-generic and identity-specific protocol models in that example, we do not have $\Pi_g = \Pi_s^*$.

[4] Since the tag's memory r is never modified, we could obtain a simpler model by directly passing k to the tag role. We have used r here to stick to the convention of definition 3.2.

[5] The equality destructor is simply defined by the rewrite rule $\mathsf{eq}(x, x) \to x$.

3.2.2 Game-based definitions

Game-based definitions of unlinkability have also been proposed, and are arguably more common than the previous notions based on ideal functionality, at least in the computational model.

Juels and Weis [JW09] have proposed to define *strong privacy* for RFID protocols through a game where the adversary attempts to distinguish two tags of his choice. They consider the most common setting for RFID protocols, where readers are identity-generic and tag sessions cannot be executed concurrently. In their game, the attacker can interact with a reader and an arbitrary number of tags $\mathcal{T}_1, \ldots, \mathcal{T}_n$. The reader can spawn multiple sessions concurrently, while each tag can spawn multiple sessions sequentially. Each new session initiated by the attacker is associated to a distinguished communication channel, so that the attacker can precisely control its interactions with these sessions. The game is in two phases:

1. During the *learning phase*, the attacker can trigger an arbitrary number of sessions of the reader and tags.

2. Eventually, the attacker chooses to enter the *guessing phase* where the attacker asks for a challenge on two tags $\mathcal{T}_i$ and $\mathcal{T}_j$ of his choice. The challenger then chooses an identity $x \in \{i, j\}$ which the attacker will have to guess. In order to do so, the attacker is only allowed to interact (an arbitrary number of times) with the reader, tags $\mathcal{T}_k$ with $k \notin \{i, j\}$, and with a special tag interface $\mathcal{T}$ which is an alias for $\mathcal{T}_x$.

In other words, the game captures the ability for the attacker to distinguish two tags of his choice. The resulting privacy notion is stronger than anonymity because the attacker may distinguish tags without knowing their identities.

For example, Juels and Weis have analyzed the OSK protocol in light of their notion of strong privacy and have discovered desynchronization attacks that break it (cf. example 3.6) despite an earlier analysis by Avoine [Avo05] concluding that the protocol provides the strongest forms of privacy.

Many variants of this game have been used in the literature, both in the computational model [OSK+03; JW09] and the symbolic one [BMU08; CK17]. In computational definitions, the tags, readers, and the game itself are interactive probabilistic Turing machines. The attacker is a probabilistic polynomial time Turing machine interacting with these machines, and wins if it can guess x at the end with non-negligible probability. In symbolic definitions, the game is expressed as an equivalence between processes. Since each new session is created on a dedicated channel, these processes can be determinate hence the specific choice of process equivalence becomes irrelevant.

Besides the unimportant distinction between the computational and symbolic definitions, game variants differ on a few essential aspects. First, tag sessions may be allowed to run concurrently. This obviously enables more attacks, and may be more reasonable in some applications: it is the case with the DAA protocol in [BMU08] but also for RFID protocols where, for instance, several copies of some RFID tag may be distributed in group-based access-control applications. Second, the complete definitions may allow

various forms of corruption, as is the case in [JW09; CK17; BMU08]. This is required to capture some forms of *forward privacy*: in the event of a tag corruption, one wishes that past interactions of this tag remain private. Third, the guessing phase may be repeated. For instance, [BMU08] express their game as an observational equivalence between two processes (choosing $\mathcal{T}_i$ or $\mathcal{T}_j$ respectively) that do not feature phases but are instead composed of replicated subprocesses corresponding to the two phases: several learning and guessing processes can thus be executed concurrently.

Comparisons

As shown by the previous brief discussion, there are lots of different games modelling unlinkability. This variety is to a large extent justified, because it reflects a variety of applications. However, it is important to understand the differences between various definitions, at least when restricting to a particular kind of application.

We have observed in [HBD19] that the game of [JW09] systematically misses attacks on unlinkability that exploit concurrent tag sessions, such as the one in example 3.5. This is actually not very surprising because this game has been designed specifically for tags with sequential sessions. The problem disappears with other games where concurrent tag sessions are allowed, e.g. [BCH10; BMU08].

We believe that our generic-reader unlinkability based on the ideal functionality implies the game-based notion of Jules and Weis, suitably represented in the applied pi-calculus. The converse is not obvious, though it is likely that only artificial examples could separate the two.

Brusò et al. [Bru+12] have attempted to give a unified account of various notions of unlinkability using epistemic logic. They consider a broad class of so-called *two-agent games* that have been used both in the computational and symbolic models, and propose a unified definition of the resulting privacy notion using epistemic logic. To achieve this, they necessarily leave out several details, resulting in the unification of game-based notions that disagree on some protocols. For example, the games of Juels and Weis [JW09] and Avoine [Avo05] are unified despite the fact that they disagree on OSK. Brusò et al. also consider a class of *three-agent games* but, in their framework, two and three-agent games are equivalent, and are also equivalent to a further generalized game-based notion. Finally, they also consider the weak and strong unlinkability notions of Arapinis et al. [Ara+10] — in order to incorporate strong unlinkability into their trace-based framework they actually consider the variant of strong unlinkability expressed in terms of equivalence, i.e. our notion of identity-specific unlinkability. They show that all these notions of unlinkability, suitably expressed in their framework, coincide under some conditions [Bru+12, Theorem 4]. They also provide some artificial but interesting protocol examples where the notions do not coincide. Allowing themselves to consider some very particular cases which are ruled out in some other frameworks, they obtain some surprising results. By considering a protocol with a bounded number of identities [Bru+12, Example 1] they show that weak unlinkability does *not* imply strong unlinkability (which breaks because the bound on the number of identities is reached earlier in the single-session scenario). They also consider a protocol where one bit of information of tag identities leaks to the attacker, allowing him to distinguish between

two types of readers [Bru+12, Example 2]: this leads to a failure of game-based notions but notions based on an ideal functionality such as strong unlinkability can still hold if there is an infinite supply of fresh identities of each type.

To conclude, we note that the OSK protocol is declared unlinkable in [BCH10] according to some definitions based on a three-agent game expressed as observational equivalence in the applied pi-calculus, but this does not show that this game is weaker than e.g. strong privacy or identity-generic unlinkability. Indeed, this observation results on an abusive assumption: the verification method of [BCH10] deals with protocols where the tag role consists of a single output, and is based on the incorrect assumption that the reader role can be nullified in such protocols without affecting unlinkability. As seen in example 3.6, the reader can actually be used in several ways to track tags in the OSK protocol.

3.3 Verification through sufficient conditions

We now turn to the problem of verifying the unlinkability notions of definitions 3.1 and 3.2. In both cases, we will see that off-the-shelf systems do not allow the automatic verification of our properties, we will provide conditions that imply unlinkability and which can be verified more easily, and use this approach to carry case studies.

3.3.1 Identity-specific unlinkability

In all of this section we consider two-party protocols with an identity-specific reader role, not making use of a database nor states. This is the setting of [HBD19].

We have seen that identity-specific unlinkability crucially needs to be expressed in terms of trace equivalence: using observational equivalence instead yields a notion that no reasonable protocol would satisfy. Hence we cannot expect to directly verify our unlinkability by using the major tools for proving equivalences of protocols with bounded sessions, namely Proverif [Bla+01] and Tamarin [Bas+12], which only support diff-equivalences[6]. Instead, we have proposed an approach based on sufficient conditions: we propose two reasonable conditions that imply unlinkability and which can be verified (mostly) automatically using off-the-shelf verification systems.

We present next these two conditions. We shall only provide their high-level intuition. Their formal definitions require some heavy technical setup, notably involving annotations of configurations and actions with agent identities.

[6]We note, among the more recent approaches, the TypeEquiv system which relies on a type system to guarantee trace equivalence [Cor+18]. It is however not as automatic as Tamarin and Proverif and, more importantly, still requires that the two processes to be proved equivalent have a similar structure. In light of the complex trace mappings performed in identity-specific unlinkability proofs, we doubt that this type-based approach could work, but it would deserve a careful consideration.

Well-authentication

The well-authentication conditions requires that, whenever a test is successful in an execution of the multiple-session process, the involved agent a must be having an honest interaction with an agent a' of the other role with the same identity. Moreover[7], a' must not be having an honest interaction with an agent other than a.

Example 3.8. *The identity-specific model of the Basic Hash protocol in example 3.7 does not satisfy well-authentication. If the output of a tag session a is sent to two reader sessions a_1 and a_2, it will be accepted in both cases. Our condition fails after the second successful test: a_1 is having an honest interaction with a, but a is also having an honest interaction with a_2.*

The complete definition [HBD19, Definition 19] excludes some trivial conditionals which do not bring knowledge to the attacker, such as the replay-prevention test in the reader of example 3.5. It also handles in a particular way the case where tags and readers do not share secrets that we use to analyze the DAA and ABCDH protocols.

Frame opacity

The second condition [HBD19, Definitions 14] requires that, for any execution of the multiple session process, the resulting frame is indistinguishable from its *idealization*. This idealization is computed by associating to each protocol output a message which may depend only on session nonces and past outputs.

Example 3.9. *The protocol of example 3.5 ensures frame opacity when the MAC messages are idealized as nonces. The Basic Hash protocol ensures frame opacity when the tag output is idealized as a pair of nonces.*

Intuitively, frame opacity ensures that no information regarding the agents' identities leaks through messages. Well-authentication ensures that the attacker cannot gain any information on the agent's identities (not even the relationships between the agent's identities) through the outcome of tests — this matters because, even though tests are not directly observable, they are often indirectly observable through error messages. Together, these two conditions ensures unlinkability.

Theorem 3.1 ([HBD19, Theorem 1])**.** *If the protocol ensures well-authentication and frame opacity, then it is identity-specific unlinkable.*

Mechanization and case studies

The interest of our two conditions is that they are much easier to verify than unlinkability. This is obvious for well-authentication: it is a typical correspondence property, which both Proverif and Tamarin can easily verify for a large class of protocols.

The case of frame opacity is a little more complicated. The general idea is to prove a diff-equivalence between the original multiple-session process and a variant of

[7] This additional condition is not imposed in the case where identity parameters are unused in the reader role, called the *non-shared* case in [HBD19]. All the examples detailed in this chapter belong to the *shared* case.

it which outputs idealized messages instead of real ones. The difficulty, however, is that tests performed on the received messages in the real process are likely to fail in the counterpart idealized execution, which leads to failures of diff-equivalence that are irrelevant to frame opacity. For example, the reader of the Basic Hash protocol tests that $\mathsf{snd}(x) = \mathsf{h}(\mathsf{fst}(x), k)$, which succeeds when x is a real tag output $\langle n, \mathsf{h}(n, k)\rangle$ but not when x is the idealization $\langle n_1, n_2\rangle$ of that same output. In [HBD16] we handle this difficulty by removing tests, thus over-approximating the possible traces of the protocol; we found that this is sufficient to verify frame opacity on many case studies. We have then proposed in [HBD19] a more precise approach based on a modification of Proverif, where we exploit the internal encoding of diff-equivalence as Horn clauses to allow generalized bi-processes where one side of the process may refer to inputs of the other side.

We have implemented a tool called UKAno [Hir16] which takes a protocol specification as input, and then automatically derives Proverif files encoding our two conditions. For frame opacity, our system can use a number of heuristics for choosing an appropriate idealization, and allows user annotations when heuristics do not allow to conclude.

This methodology and its implementation in UKAno have allowed us to carry out a number of case studies. This has allowed us to obtain (often for the first time) formal proofs of unlinkability in unbounded sessions, and to find new attacks. We have generally observed that a failure of one of our two conditions can easily be translated to an attack against unlinkability, i.e. our conditions are tight in practice.

We have been able to verify the unlinkability of simple protocols such as the sequential version of the protocol of example 3.5 and the Hash-Lock protocol [JW09] which is similar to the Basic Hash protocol but prevents replay attacks. We have also studied the LAK protocol [LAK+06], idealized without state updates, and discovered that it fails unlinkability: in fact LAK readers do not properly authenticate tags, due to the malleability of exclusive or, as reported in [DR08]. We have then verified the unlinkability of the LAK protocol where exclusive or is replaced by pairing. Our approach has also lead to a proof of unlinkability for the Basic Access Control (BAC) protocol used in electronic passports, which had long been sought. In fact, we have also verified the unlinkability of the combination of BAC with the subsequent Passive and Active Authentication protocols.

We have also analyzed the PACE e-passport protocol. On this case study, our approach has proved useful to highlight the importance of some fine points of the informal specification. In particular, the protocol is linkable when concurrent reader sessions are allowed. We have proposed a modification of the protocol, which we could prove to be unlinkable using Proverif. We have also turned to Tamarin for verifying the original protocol with sequential sessions. Surprisingly, the more complete modelling of Diffie-Hellman exponentiation of Tamarin lead to the discovery of a new attack on authentication, with no impact on unlinkability: this is the only case where we found that our conditions were not tight, and we estimate that it is not worth tightening our approach to cover such risky situations.

We have finally been able to verify unlinkability for more complex protocols: we have analyzed some Direct Anonymous Authentication protocols that had been previously

been proved unlinkable according to some game-based notion [SRC15], and the ABCDH protocol [AH13] for attribute-based authentication, which is the only example where Proverif needed more than a few minutes to conclude.

3.3.2 Identity-generic protocols

In [BDM20] we have considered the case of protocols where tags rely on a mutable state and readers may use a database. In practice, such protocols are meant to have identity-generic readers, and we have proposed the notion of identity-generic unlinkability (definition 3.2) as a good model of unlinkability in that case.

Identity-generic unlinkability can be written as a diff-equivalence using the following bi-process, where R', I' and I'' will be detailed below:

$$!\, R' \mid !\, \mathbf{new}\ r, \vec{k}.\ I'(r, \vec{k});\ ¡\, \mathbf{new}\ r', \vec{k}'.\ I''(r', \vec{k}');\ T(\mathsf{choice}(r, r'))$$

The trick is to introduce the new identity $\vec{k}'$ after the second (sequential) replication, and to use only one of the two identities in each projection of the bi-process. In the left projection, multiple sessions of a tag using r are possible. In the right projection, a single tag session is allowed for each r'. We have thus managed to obtain multiple tag sessions on the left, and single session on the right. It remains to adapt the database and reader: the processes R', I' and I'' can be derived from R and I so that, in the left (resp. right) projection, only database entries inserted by I (resp. I') are taken into account by R.

On the surface, it seems that this diff-equivalence could hold for identity-generic unlinkable protocols. Indeed, a trace of the multiple-session process can be mapped to a single reasonable trace of the single-session process: unlike in the identity-specific case there is no question of which pairs of agents in the multiple-session execution should have the same identity in the single-session execution. Unfortunately, the specific diff-equivalences of both Proverif and Tamarin systematically fail to verify our diff-equivalence when readers use a database. In Proverif, the database is modelled using a table, and the two sides of a bi-process must perform a lookup in the same table. We must thus tag the keys in the table, or adapt our bi-process to only insert in the table the relevant keys (perhaps several times). In both cases, diff-equivalence fails because it requires that the table lookup retrieves, on the two sides of the bi-process, table entries e_1 and e_2 that had been inserted together using some $\mathbf{insert}(\mathsf{choice}(e_1, e_2))$; this is too strong for our processes. A similar problem occurs with Tamarin, where database entries are modelled using facts. As we will see in chapter 5, a diff-equivalence notion that allows to decouple the database entries on both sides of the bi-process will allow us to verify unlinkability.

Being faced again with the impossibility to directly verify unlinkability with existing tools, we set out to extend the previous approach to this new class of protocols and unlinkability notion. The frame opacity condition is still relevant and immediately adapted to our new setting. However, we modify well-authentication slightly: when a test is successful, we impose (as before) that the corresponding role a is having an honest interaction with an agent a' of the dual role, but we allow a' to be having honest

interactions with other agents than a — the failures of unlinkability associated to this constraint on a' in the identity-specific case disappear with generic readers, and relaxing the condition is necessary to capture examples such as the Basic Hash protocol where replays are possible. Finally, in order to obtain sufficient conditions for unlinkability, we need to add a third condition.

No-desynchronization

This new condition requires that, in any execution trace of the multiple-session protocol where two agents a and a' are having an honest interaction, any test performed by one of these agents must be successful. Note that this property, which is actually the converse of well-authentication, is obviously met when the tag and reader roles are stateless.

Example 3.10. *The failures of unlinkability due to desynchronization attacks in example 3.6 are actually failures of our no-desynchronization condition: after some trace where the attacker manages to desynchronize some tag, a honest interaction between new sessions of the tag and reader does not lead to a success.*

Theorem 3.2 ([BDM20, Theorem 1])**.** *If a protocol ensures well-authentication, frame opacity and no desynchronization, then it is identity-generic unlinkable.*

Mechanization and case studies

In order to formally verify our conditions on case studies, we have turned to Tamarin since it allows to analyze more precisely the sequential sessions and the states of our tags. We could carry out a number of case studies, summarized in figure 3.1. For each protocol we indicate the status of the well-authentication (WA), frame opacity (FO) and no-desynchronization (ND) conditions. The failure of ND for the OSK protocol leads to an unlinkability attack as explained in example 3.6, and the situation is similar for the LAK protocol [LAK+06] studied here with its state update mechanism. For both protocols, we propose and verify several fixes. Finally, we managed to verify a protocol inspired by the 5G AKA protocol — the original protocol being known to suffer from several privacy issues [Ara+10; Bor+17; Kou19b].

As mentioned before, expressing well-authentication is standard in Tamarin. The no-desynchronization condition is less usual, but could be expressed using the rich logical language available to specify trace properties. Proving these properties, however, often required significant user guidance including but not limited to the inclusion of lemmas expressing useful invariants of the protocol, e.g. the increasing number of hash applications in OSK. In these proofs, inductive reasoning was crucial to handle state updates. Frame opacity could be proved using the approximate technique from [HBD16]: it does not seem possible to extend Tamarin's diff-equivalence method in a similar way to what we did in Proverif. Again, this required a lot of guidance and careful modelling. In particular, because inductive reasoning is only available for trace properties and not equivalence properties in Tamarin, we had difficulties dealing with state updates and ended up over-approximating process behaviors by removing state updates.

	Unlink.	WA	FO	ND
Basic Hash	ok	✓	✓	✓
Hash-Lock	ok	✓	✓	✓
Feldhofer	ok	✓	✓	✓
OSK (v1)	**attack**	✓		✗
OSK (v2)	ok	✓	✓	✓
LAK (pairs)	**attack**	✓		✗
LAK (pairs, fix v1)	ok	✓	✓	✓
LAK (pairs, fix v2)	ok	✓	✓	✓
5G-AKA (simplified)	ok	✓	✓	✓

Figure 3.1: Summary of case studies: ✓ means the condition holds and is automatically proven with Tamarin, ✗ means the conditions does not hold.

Although Tamarin provides some support for exclusive or, we found that it was not sufficient to be able to analyze RFID protocols using this primitive, e.g. the MW protocol [MW04b].

3.4 Conclusion

We have shown that strong notions of unlinkability are currently out of the reach of direct verification, but that formal verification is possible through the use of sufficient conditions. This pragmatic approach had been followed before us to verify unlinkability [BCH10] on a restricted class of protocols, and has been applied since then to ballot secrecy [HC19]. After all, protocol equivalence is a very hard problem, but protocol designers achieve privacy by following some high-level intuitions, often encoded as best practices; it would be a waste to forget about these when verifying protocols.

We list in our papers on unlinkability [HBD19; BDM20] some possible technical improvements of our conditions and verification techniques. However, we have become more interested in a new approach, described in chapter 5 which provides computational guarantees but also enables the verification of xor-based protocols previously out of the reach of Tamarin. We also note that the recent notion of session-equivalence [CKR19], which implies trace equivalence and is less restrictive than diff-equivalence, may become available in future versions of Proverif[8] which would enable the direct verification of identity-specific unlinkability.

Our works on unlinkability have contributed to a better understanding of the various notions of unlinkability that have been proposed in the framework of the applied pi-calculus. Several questions remain open regarding the comparison between our notions of unlinkability and game-based definitions. Importantly, we should pursue our line of work by considering corruptions, as has been done in other settings [JW09; BCH10].

[8] Personal communication with Vincent Cheval, October 2020.

4

Partial-Order Reduction Techniques

We have seen in section 2.8 that most tools for deciding trace equivalence on security protocols with bounded sessions rely on an exploration of execution traces in some symbolic semantics. When I joined LSV in 2012, existing tools and their underlying procedures were very recent [CCD11; TD10; CCK12], and research efforts had concentrated on the verification of the generalizations of static equivalence that arise in these symbolic semantics. Unsurprisingly, the exploration of symbolic execution traces was performed naively, considering all possible interleavings of concurrent actions and thus encountering many times inessential variants of the same trace. This is a classic problem in the verification of concurrent systems, which has been tackled through the development of many partial-order reduction (POR) techniques [God96; Pel98; BK08]. Together with Stéphanie Delaune and our PhD student Lucca Hirschi, I have worked on developing such techniques that are applicable to the verification of protocol equivalences.

Baier and Katoen [BK08] point out that "[POR] is mainly appropriate to control-intensive applications and less suited for data-intensive [ones]". Compared to Petri nets or concurrent programs, security protocols have very simple control structures: when one assumes that all communications are mediated by the environment, there is no synchronization at all. However, action dependencies crucially rely on data: $\mathbf{in}(c, R)$ depends on $\mathbf{out}(c, w)$ when w occurs in R. In traditional POR, data is often ignored through approximations, but this would weaken the reductions too much in our case since any input would depend on any output. This degenerate situation arises when we move from the analysis of a program's behaviour to the analysis of all possible behaviours of a protocol composed with arbitrary attackers: operations on distinct channels that are independent in the former situation become dependent in the latter one. In order to obtain powerful POR techniques for security, we thus need to take data into account. Moreover, this must be done in a way that can be efficiently supported in the context of symbolic execution and constraint solving.

Several earlier works[1] have designed POR techniques for the verification of cryptographic protocols. Some techniques, e.g. [CJM03; CM05; FDW10], rely on or are inspired by abstract POR concepts such as ample sets [Pel93] or sleep sets [God90]. Others, e.g. [MVB10], are designed directly in the concrete setting used for protocol verification. The works presented in this chapter differ from these earlier works in two important ways. First, our POR techniques take data into account in a way that is suitable for symbolic execution. In contrast, all of the above works either approximate data away or assume a finite set of possible messages. The notable exception is the *constraint differentiation* technique of Mödersheim et al. [MVB10], from which we take inspiration. Second, our POR techniques are appropriate for the verification of trace equivalence. All of the above works only deal with reachability properties, except for the work of Fokkink et al. [FDW10]. The techniques designed in that paper aim at reducing the LTS of some process into a simpler LTS that must remain equivalent to the original one. We follow a less constraining approach, which allows much stronger reductions: we will reduce the LTS of two processes in such a way that the reduced systems are equivalent iff the original systems were equivalent.

We present in section 4.1 our first and main line of work on POR for security protocol equivalences [BDH14; BDH15a; BDH17]. It introduces the compression and reduction techniques, which are correct for a class of *action-deterministic* processes. Lifting this action-determinism condition has motivated another approach [BDH18], presented in section 4.2, which explicitly builds on the general POR concepts of persistent and sleep sets.

4.1 Compression and reduction

The starting point of this first line of work on POR was the proof-theoretical concept of focusing, on which I had worked during my PhD [BM07; BMS10; Bae12]. Originally, focusing is a complete proof-search strategy for linear logic, proposed by Andreoli [And92] as an attempt to generalize uniform proofs [Mil+91] in order to design a logic programming language based on linear logic [AP90]. The concepts behind focusing (e.g. polarity) were later found to be relevant to virtually all logics [Her95; DJS95; LM09; BMS10; Bae12] and have been used in unexpectedly varied applications in proof theory [Gir01; Lau02; JNS05; CP05; CPP06; CMS08]. Focusing is now generally considered as one of the fundamental ideas in proof theory, next to cut elimination. Given the possibility of encoding processes as linear logic formulas [Mil93; Mil03] so that process execution is encoded as proof search, it is to be expected that focusing can be transfered in some way to improve the verification of processes.

We proposed in [BDH14] two successive POR techniques: the first one, called compression, is a direct adaptation of focusing; it is further improved by a second technique, called reduction, which is inspired by the *constraint differentiation* technique of Mödersheim et al. [MVB10]. The two techniques are proved to be sound wrt. trace

[1] The interested reader can find in [BDH15b] an extensive discussion of related works, not limited to cryptographic protocol verification.

equivalence for the restricted class of *simple* processes, where parallel compositions can only occur at toplevel and parallel subprocesses rely on distinct channels. We implemented our method in a modified version of SPEC [TNH10], and obtained dramatic improvements. We showed later [BDH15a] that the same ideas could be put to work in a general process algebra, under some action-determinism condition. This provided the foundation for a mature implementation of our techniques in the main line of development of Apte [Che13]. The integration of our POR techniques in the specific framework of Apte, notably involving a complex symbolic semantics, required a careful justification which was finally provided in a journal version of our first work [BDH17].

The rest of this section describes our techniques as presented in following [BDH15a]. We start by adapting the process algebra in section 4.1.1. Then, we present in section 4.1.2 an *annotated* semantics which will be instrumental in lifting POR techniques from reachability to trace equivalence. We introduce compression and reduction in sections 4.1.3 and 4.1.4, and describe in section 4.1.5 how they can be integrated in verification tools based on symbolic execution. Section 4.1.6 concludes with further discussion, notably commenting on follow-ups of our work.

4.1.1 Model

We restrict the syntax of our processes to only consider replication and **new** constructs in a particular form. We adapt the syntax of processes as follows, where c, a are terms of sort channel, u is a term of sort message and x is a variable of sort message:

$$\begin{array}{rcl} P, Q & ::= & 0 \quad | \quad (P \mid Q) \quad | \quad \textbf{if } u = v \textbf{ then } P \textbf{ else } Q \\ & | & \textbf{in}(c, x).P \quad | \quad \textbf{out}(c, u).P \quad | \quad !^{a}_{\vec{c},\vec{x}} P \end{array}$$

The last construct combines replication with channel and name restriction:

$!^{a}_{\vec{c},\vec{x}} P$ should be understood as ! **new** $\vec{c}$. **out**$(a, \vec{c})$. **new** $\vec{x}$. P.

In other words we restrict the creation of new names and channels to take place just after a replication, which is without loss of generality. Moreover, we impose that replication is always followed by an observable action, which emits on channel a all newly generated channels. This imposes that all channels are public.

Our purpose with the constrained use of replication, which is indeed restrictive, is to allow unbounded processes in a way that can be compatible with the action-determinism condition that we will eventually define and impose on our processes. Our techniques will ultimately be applied for the verification of bounded processes, but we consider (some form of) replication anyway to show that the underlying theoretical results apply in the unbounded case. In particular, this brings more depth to the analogy with focusing. Moreover, it would be trivial to adapt our results to bounded replication, which is interesting because the verification of a bounded replication $!^{n}P$ is generally more efficient than that of n copies of P (i.e., $P \mid \ldots \mid P$) since symmetries cause redundancies that are not eliminated by POR techniques.

The precise LTS definition of [BDH15a] is formulated in a specific way to ease the technical development. We can ignore such details here, except for one essential point:

all of this work takes place in a semantics without internal communications. From now on we thus assume that $\rightarrow$ is defined as in figure 2.2 but without the fourth rule. In other words, we force all communications to be mediated by the attacker — this is possible because we do not consider private channels, but this choice will be discussed further in section 4.1.6. We also impose wlog. that unobservable actions other than replication are executed eagerly, and force the transitions corresponding to the several parts of $!^a_{\vec{c},\vec{n}}P$ to be executed in one step (i.e., replication must be followed by **new** transitions and the output on a).

Finally, we shall work under an action-determinism condition, as is common in POR [BK08]. Intuitively, it imposes that any sequence of actions that the system may perform can only be performed in one way. In other words, action-deterministic systems respond deterministically when they are controlled by their environment. In the context of process algebras, this can be made formal in several ways. The notion of determinacy [CCD13] is quite permissive, as it only requires that for any trace, the executions from the initial configuration can only lead to one configuration up to trace equivalence. While this notion is the right one to consider for some theoretical investigations, it is sometimes useful to consider a more structural criterion, such as the one given next[2].

Definition 4.1 (Action-determinism)**.** *We say that a process is an input (resp. output, or session) process on channel a when it is of the form* **in**$(a, x).P$ *(resp.* **out**$(a, u).P$*, or* $!^a_{\vec{c},\vec{x}}P$*). A configuration $(\mathcal{P}, \Phi)$ is* action-deterministic *when, for any t and $(\mathcal{P}', \Phi')$ such that $(\mathcal{P}, \Phi) \xrightarrow{t} (\mathcal{P}', \Phi')$, the multiset $\mathcal{P}'$ does not contain two input (resp. output, or session) processes on the same channel.*

The difference between action-determinism and determinacy is often irrelevant in pratical examples. In particular, the discussions of chapter 3 regarding which unlinkability notions can be modelled as equivalences of determinate processes carry over to action-determinism. Our notion of action-determinism is the right one for our technical development, but its meaning can be questioned; we will come back to this in section 4.1.6.

4.1.2 Annotated semantics

The first step in our development is to enrich the semantics with annotations that indicate the provenance of actions in processes. In particular, annotations will indicate when two actions originate from concurrent processes in a configuration. The key result about this enriched semantics is that trace equivalence for the regular semantics coincides with trace equivalence for the annotated semantics when configurations are action-deterministic. In other words, the attacker does not gain any distinguishing power when annotations are made observable.

We omit the specific definition of the annotated semantics, and refer the interested reader to [Hir17] which provides definitions and proofs, fixing a problem in the definition of [BDH15a]. For our present purpose it suffices to know that there are objects called

[2] Our notion of action-determinism is still not practical: checking that a configuration is action-deterministic is undecidable. When implementing our POR techniques, more constraining syntactical checks are used to ensure action-determinism.

labels, noted ℓ, equipped with an *independence* relation. Labels are used to identify processes in a configuration and its successors, and independent labels will intuitively identify concurrent processes. We define an *annotated semantics* by enriching the regular LTS. Actions of the annotated LTS are labelled: they are of the form $[\alpha]^\ell$ where α is a regular action and ℓ is a label. Configurations are similarly enriched: they contain labelled processes $[P]^\ell$ and are subject to some *well-labelling* condition which notably imposes that process labels are pairwise independent. Finally, transitions are adapted to update labels in an appropriate way, notably ensuring that well-labelling is preserved by transitions.

Example 4.1. *Consider the labelled configuration* $(\{[P]^\ell\}, \emptyset)$ *where*

$$P \stackrel{def}{=} \mathbf{in}(c_1, x) \mid \mathbf{out}(c_2, u_2).\mathbf{out}(c_3, u_3).$$

As in the regular semantics, two kinds of actions can be performed by this configuration in the annotated semantics: $[\mathbf{in}(c_1, R)]^{\ell_1}$ *and* $[\mathbf{out}(c_2, w)]^{\ell_2}$*. After executing an action of the second form, actions* $[\mathbf{out}(c_3, w')]^{\ell_3}$ *become available. The annotated semantics is such that* ℓ_1 *and* ℓ_2 *are independent, but* ℓ_2 *and* ℓ_3 *are dependent. Indeed, the actions on channels* c_1 *and* c_2 *may be executed in any order, without any impact on the resulting configuration, but it is clearly not the case for the two outputs. Note, however, that labels are not always sufficient to define action dependencies: when* w *occurs in* R*,* $\mathbf{in}(c_1, R)$ *can only be executed after* $\mathbf{out}(c, w)$*.*

The previous example illustrates that control-flow and data-flow dependencies need to be accounted for in our LTS. This is done formally in the next definition.

Definition 4.2. *We say that the labelled actions* α *and* β *are* sequentially dependent *when their labels are dependent, and* recipe dependent *when*

$$\{\alpha, \beta\} = \{[\mathbf{in}(c, R)]^\ell, [\mathbf{out}(c', w)]^{\ell'}\}$$

with w *occurring in* R*. They are* dependent *when they are sequentially or recipe dependent. Otherwise, they are* independent*.*

Action independence precisely captures when two actions can be permuted:

Lemma 4.1 ([Hir17, Lemma 1])**.** *Let* A *be a (well labelled) configuration,* α *and* β *be two independent labelled actions. We have* $A \xrightarrow{\alpha.\beta} A'$ *iff* $A \xrightarrow{\beta.\alpha} A'$*.*

More surprisingly, under the action-determinism condition, labels and unobservable actions can be considered as observables without changing the resulting trace equivalence: we show next that configurations which are trace equivalent (wrt. the regular semantics) must also have the same labelled traces. Of course, this result relies on the specific definition of the annotated semantics, which updates labels in a sufficiently canonical way. We also have to assume that the initial configurations have the same *skeleton* skel$(\bullet)$, which roughly means that they are labelled in the same way, and we show that this property can be maintained throughout executions.

Lemma 4.2 ([Hir17, Lemma 2])**.** *Let* K *and* K' *be two action-deterministic configurations with the same skeleton, such that* $K \approx K'$*. For any execution*

$$K \xrightarrow{[\alpha_1]^{\ell_1}} K_1 \xrightarrow{[\alpha_2]^{\ell_2}} K_2 \ldots \xrightarrow{[\alpha_n]^{\ell_n}} K_n$$

there exists an execution

$$K' \xrightarrow{[\alpha_1]^{\ell_1}} K'_1 \xrightarrow{[\alpha_2]^{\ell_2}} K'_2 \dots \xrightarrow{[\alpha_n]^{\ell_n}} K'_n$$

such that $\Phi(K_i) \sim \Phi(K'_i)$ *and* $\mathsf{skel}(K_i) = \mathsf{skel}(K'_i)$ *for all* $i \in [1, n]$.

Example 4.2. *Take* $K = (\{[\mathbf{in}(a, x).(\mathbf{out}(b, u).P_1 \mid P_2)]^{\ell}\}, \Phi)$ *with* $P_1 = \mathbf{in}(c, z)$ *and* $P_2 = \mathbf{in}(d, z)$. *Let* K' *be the configuration obtained from* K *by swapping* P_1 *and* P_2. *The configurations* K *and* K' *are compatible. In the annotated semantics, we have*

$$K \xrightarrow{[\mathbf{in}(a,R)]^{\ell}.[\tau]^{\ell_1}} K_1 \xrightarrow{[\mathbf{out}(b,w)]^{\ell_2}.[\mathbf{in}(c,R')]^{\ell_3}} K_2$$

where each label is dependent with the previous one. We also have

$$K' \xrightarrow{[\mathbf{in}(a,R)]^{\ell}.[\tau]^{\ell_1}} K'_1$$

but $\mathsf{skel}(K_1) \neq \mathsf{skel}(K'_1)$, *reflecting the fact that the two configurations do not exhibit the same immediately available actions: only* K'_1 *can perform an input on* c. *By the previous lemma, this suffices to declare that* $K \not\approx K'$.

Our modified semantics gives rise to a modified trace equivalence, as will be the case for the modified semantics corresponding to our POR techniques. We say that two labelled configurations are *compatible* when they are well-labelled, have the same skeletons, and the underlying regular configurations are compatible.

Definition 4.3 (Annotated trace equivalence, $\approx_a$)**.** *Let* K_1 *and* K_2 *be two compatible labelled configurations. We say that* $K_1 \approx_a K_2$ *when, for any annotated execution* $K_1 \xrightarrow{t} K'_1$, *we have* $K_2 \xrightarrow{t} K'_2$ *for some* K'_2 *such that* $\Phi(K'_1) \sim \Phi(K'_2)$.

As a slight abuse of notation, we allow ourselves to compare labelled configurations using regular trace equivalence, implicitly erasing all annotations.

Corollary 4.1. *Let* K *and* K' *be two compatible annotated configurations that are action-deterministic. We have* $K \approx K'$ *iff* $K \approx_a K'$.

The rest of this section only uses the annotated semantics. In particular, from now on, α stands for a labelled action and $\mathcal{P}$ stands for a well-labelled multiset of labelled processes. However, we shall omit annotations in examples when they are irrelevant.

4.1.3 Compression

Our first POR technique is directly inspired by the notion of focusing from proof-theory [And92]. This idea emerged in the context of backward proof-search in the sequent calculus for linear logic: starting from a sequent, one attempts to find a rule instance with this sequent as conclusion, and then attempts to recursively find a proof for the rule's premisses. One immediately observes that this search procedures features some inessential non-deterministic choices: some *invertible* rules can be applied eagerly without any risk of losing provability. Focusing shows that the other rules also enjoy a strong property: one does not loose proofs when they are applied in a chained fashion. This idea can be transposed to process calculi to yield a partial-order reduction technique. Moreover, for the case of action-deterministic processes, the technique is correct for trace equivalence verification.

Definition 4.4. *A process P is* positive *if it is of the form* $\mathbf{in}(c, x).Q$*, and it is* negative *otherwise. A multiset of processes is* initial *if it contains only positive or* replicated *processes, i.e., of the form* $!^a_{\vec{c},\vec{n}}Q$*.*

The compressed semantics is essentially a restriction of the annotated semantics following to a particular strategy that alternates between *negative* and *positive* phases. Execution starts in the negative phase, where outputs and unobservable actions are executed until the configuration becomes initial. At this point a process of the initial configuration is chosen: if it is positive, it is distinguished as the process under *focus*; if it is replicated, a new copy of it is formed as placed under focus. Then, the positive phase consists of executing actions of the process under focus as long as it remains positive. When this is no longer the case, the focus is released and the compressed execution gets back to a negative phase.

Between any two initial configurations, the compressed semantics executes a sequence of actions, called *blocks*, consisting of the choice of a focus, followed by a (possibly empty) sequence of input actions, the release of the focus, and a (possibly empty) sequence of output and unobservable actions. By forcing the execution of blocks, the compressed strategy prevents some interleavings, and yields a higher-level view of traces as sequences of blocks rather than individual actions.

Example 4.3. *Consider a configuration* $(\{P\}, \Phi)$ *with*

$$P \stackrel{def}{=} !^a_{c,k}\mathbf{in}(c, x).\mathbf{out}(c, \mathsf{senc}(x, k)).$$

According to the compressed strategy, once a replication is performed on P (together with the associated **new** *actions) the resulting process is under focus and must be completely executed. Hence, sessions cannot be interleaved.*

Our compressed strategy can be encoded explicitly as a *compressed* LTS [BDH15a], which yields a compressed trace equivalence $\approx_c$. It can be shown that any execution in the annotated semantics can be completed to obtain a longer execution that is permutation-equivalent to a compressed execution [BDH15a, Lemma 17]. Thus compression can be used to reduce the search space when verifying reachability properties. Further, because the compressed strategy is completely determined by labels and skeletons, we actually have a coincidence between $\approx_c$ and $\approx_a$ for action-deterministic configurations. By lemma 4.2 this means that regular trace equivalence coincides with compressed trace equivalence for action-deterministic configurations.

Theorem 4.1 ([BDH15a, Theorem 19])**.** *The equivalences $\approx$ and $\approx_c$ coincide for compatible action-deterministic configurations.*

We note that the completeness lemma [BDH15a, Lemma 17] mentioned above uses a simplified version of Miller and Saurin's completeness proof for focusing [MS07]. This argument is based on the existence of specific rule permutations, structured by polarity. We note that, in the case of security protocols, these permutations are obvious, but they have remained incompletely exploited so far even in the context of reachability analyses. In fact, [CJM00] explicitly lists the possible permutations between input and output actions, but they are only used in a modified ample set technique which does not eliminate the interleavings of concurrent sequences of inputs.

4.1.4 Reduction

Compression achieves a partial order reduction by forcing the execution of whole blocks. However, much redundancy remains in the compressed LTS, because it allows arbitrary interleavings of blocks, regardless of block independences — we say that two blocks are independent when their actions are pairwise independent.

The idea of the reduced semantics [BDH15a] is to select, for each class of equivalent traces, a single representative that can be executed. The precise definition of when two traces are equivalent, given in [BDH15a], captures two ideas: first, permuting adjacent independent blocks in a sequence of blocks yields an equivalent trace; second, changing some input action $\mathbf{in}(c, R)$ into $\mathbf{in}(c, R')$ when $R\Phi =_{\mathsf{E}} R'\Phi$ yields an equivalent trace. The latter case, of course, requires to talk of this equivalence in the context of a specific frame. When t and t' are two equivalent traces wrt. $\Phi(K')$, we have $K \xrightarrow{t}_c K'$ iff $K \xrightarrow{t'}_c K'$; it is precisely this sort of redundancy that our reduced LTS eliminates.

The recipe dependencies of definition 4.2 are sometimes called second-order data dependencies, while first-order data dependency is defined in terms of messages rather then recipes, as in the previous definition. First-order dependency is much stronger than second-order dependency, and can thus yield stronger partial-order reductions.

Example 4.4. *Consider a signature featuring pairs, a hash function* h *and some constant* init *used to initiate sessions of Basic Hash tags. Consider the following configuration, for distinct channels c and d and distinct names n and m (we omit labels because they are irrelevant here):*

$$\begin{aligned} T(c,n) &\stackrel{def}{=} \mathbf{in}(c,x).\mathbf{if}\ x = \mathsf{init}\ \mathbf{then}\ \mathbf{out}(c,\langle n, \mathsf{h}(n,k)\rangle) \\ K &\stackrel{def}{=} (\{T(c,n), T(d,m)\}, \emptyset) \end{aligned}$$

The following trace can be executed by K in the compressed semantics:

$$\mathbf{in}(c,\mathsf{init}).\tau.\mathbf{out}(c,w).\mathbf{in}(d,\mathsf{snd}(\langle w,\mathsf{init}\rangle)).\tau.\mathbf{out}(d,w')$$

This trace features two blocks that are dependent, due to the occurrence of w in the recipe $\mathsf{snd}(\langle w,\mathsf{init}\rangle)$*. However, that recipe obviously yields the same message as* init *alone. After simplifying the recipe in this way, the two blocks are no longer dependent and can be permuted. Thus, our trace is equivalent to the following one:*

$$\mathbf{in}(d,\mathsf{init}).\tau.\mathbf{out}(d,w').\mathbf{in}(c,\mathsf{init}).\tau.\mathbf{out}(c,w)$$

It is clearly useless to consider both traces when verifying reachability properties. They are actually also redundant wrt. equivalence verification when configurations are action-deterministic.

We now define the reduced semantics. In order to select a representative in each class of equivalent traces, we assume an arbitrary partial order $\prec$ on blocks, that is insensitive to recipes and such that independent blocks are always comparable. The lexicographic extension $\prec_{\mathsf{lex}}$ of $\prec$ will be used to compare sequences of blocks.

Definition 4.5 ([BDH15a, Definition 23]). *We define the authorization relation $t \rhd b$, where t is a sequence of blocks and b is a block, by the next three rules:*

$$\frac{}{\epsilon \rhd b} \qquad \frac{b \text{ and } b' \text{ are dependent}}{t'.b' \rhd b} \qquad \frac{b \text{ and } b' \text{ are independent} \quad b' \prec b \quad t' \rhd b}{t'.b' \rhd b}$$

The reduced semantics is then defined as a restriction of the compressed one: we have $K \xrightarrow{b_1 \dots b_n}_r K'$ when this execution holds in the compressed semantics and, for all i, $b_1 \dots b_i \rhd b_{i+1}$.

It can then be shown ([BDH15a, Lemma 26]) that, if some trace t can be executed in the compresses semantics, the only equivalent traces that are executable in the reduced semantics are the ones that are are minimal wrt. $\prec_{\text{lex}}$ — these minimal traces only differ in their recipes.

Example 4.5. *Consider the following configuration, for some $n \in \mathcal{N}$:*

$$(\{\mathbf{in}(c, x).\mathbf{in}(c, y).\mathbf{out}(c, n), \mathbf{in}(d, z).\mathbf{out}(d, n)\}, \{n\}.\emptyset)$$

Assume moreover that the order $\prec$ prioritizes blocks on channel c over those on d, i.e., $\mathbf{in}(c, R_x).\mathbf{in}(c, R_y).\mathbf{out}(c, w) \prec \mathbf{in}(d, R_z).\mathbf{out}(d, w')$. In the reduced semantics, this process has traces of length up to two blocks. Traces of length two are as follows:

- $\mathbf{in}(c, R_x).\mathbf{in}(c, R_y).\mathbf{out}(c, w).\mathbf{in}(d, R_z).\mathbf{out}(d, w')$ *for any plausible*[3] *recipes R_x, R_y and R_z;*
- $\mathbf{in}(d, R_z).\mathbf{out}(d, w').\mathbf{in}(c, R_x).\mathbf{in}(c, R_y).\mathbf{out}(c, w)$ *when w' occurs in either R_x or R_y.*

In other words, the block on d is only allowed first when data dependencies require it.

As before, the reduced semantics induces a reduced trace equivalence $\approx_r$. It can be shown that it coincides with $\approx_c$. This crucially relies on the fact that minimal traces wrt. some frame are also minimal wrt. statically equivalent frames.

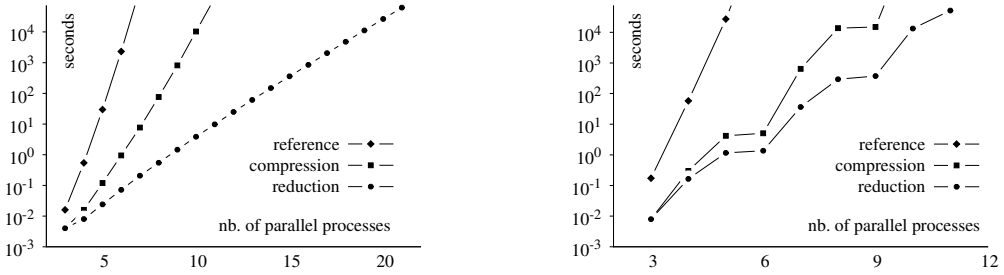
Theorem 4.2 ([BDH15a, Theorem 28]). *The equivalences $\approx$ and $\approx_r$ coincide for action-deterministic compatible configurations.*

4.1.5 Integration with symbolic semantics and verification tools

We have seen in section 2.8 that most tools for verifying bounded trace equivalence rely on some form of symbolic semantics. We now discuss how our POR techniques can be put to use in that context.

The compression technique can be immediately applied in a symbolic semantics, since it does no rely on recipes. This already brings a significant reduction of the number of traces to explore. Compression has actually been implemented in Akiss in order to restrict the number of symbolic traces to be analyzed — however, that system does not benefit from our second POR technique so far.

[3] This means that each recipe can only mention handles of previous outputs. For instance, w' may not occur in R_y.



(a) Toy example

(b) Denning-Sacco protocol

Figure 4.1: Impact of our POR techniques in Apte (see [BDH15a] for details)

In order to also leverage our reduction technique, one has to somehow reflect the authorization predicate $t \rhd b$ at the symbolic level. We have shown how a new kind of symbolic constraint, called dependency constraint, can be added to various symbolic semantics: this is done in [BDH14] for a standard symbolic semantics, and we address in [BDH17] the much more technical case of Apte's specific symbolic execution framework. These dependency constraints are then passively maintained throughout constraint resolution, and only serve to discard some solutions when they become unfeasible. This general integration scheme has lead to excellent results in SPEC [BDH14], Apte [BDH15a] and Deepsec [CKR18].

Figure 4.1 shows the performance of Apte without POR, with compression alone and with reduction, on a toy example and on the verification of strong secrecy for the Denning-Sacco protocol. The graphs show how the verification time (in logarithmic scale) increases with the number of protocol sessions. The expected exponential speedup is confirmed, with a slightly lesser impact of reduction in the real-life protocol than in the toy example.

The intended use of our POR techniques in symbolic verification tools justifies our construction in two steps, building the reduced LTS on top of the compressed one. Conceptually, one could apply the main idea behind our reduced LTS directly on the regular LTS. This would yield a semantics that can execute only minimal representatives of each class of equivalent sequences of actions. Going back to example 4.5, the reduced traces of maximal length would be as follows:

- $\mathbf{in}(c, R_x).\mathbf{in}(c, R_y).\mathbf{out}(c, w).\mathbf{in}(d, R_z).\mathbf{out}(d, w')$ for any R_x, R_y and R_z;
- $\mathbf{in}(c, R_x).\mathbf{in}(d, R_z).\mathbf{out}(d, w').\mathbf{in}(c, R_y).\mathbf{out}(c, w)$ when w' occurs in R';
- $\mathbf{in}(d, R_z).\mathbf{out}(d, w').\mathbf{in}(c, R_x).\mathbf{in}(c, R_y).\mathbf{out}(c, w)$ when w' occurs in R.

The minimality condition expressed at the level of actions gives rise to the two interleavings of blocks, as with the block-level reduced semantics, but also introduces traces where blocks are interleaved. The action-level and block-level reduced semantics are both optimal in theory: for any concrete trace (i.e. where recipes are explicit) there is a single minimal permutation that is allowed in any of the two reduced semantics. However, when used in the context of symbolic execution, the action-level reduced semantics

allows a priori more interleavings and would thus be less efficient than the block-level reduced semantics. In contrast, the block-level reduced semantics benefits from the cost-free pre-processing performed by compression — which is close to optimal when data cannot be taken into account.

4.1.6 Discussion

Our work on POR started from the observation that the concurrent nature of security protocols was too naively handled in the equivalence verification tools of the time, and the intuition that focusing could be an inspiration for more efficient techniques. Since the publication of our compression and reduction techniques, and our implementations of these techniques in SPEC and Apte, our ideas have been recognized and adopted in the community of researchers working on the topic. Kremer has implemented compression in Akiss [4] and Cheval, Kremer and Rakotonirina have implemented both compression and reduction in the first release of Deepsec [Che+18], the successor of Apte. In [CKR19], the same authors have also (reformulated and) generalized our POR techniques, notably accounting for private channels, and complementing them with symmetry reduction techniques[5].

Overall, it seems that our compression and reduction techniques are simple, effective ideas that are here to stay. Their main limitation is the action-determinism condition on which they crucially rely; one possible way to avoid this condition is explored in the next section. Before moving on to this, we discuss how our techniques relate to the verification of may-testing equivalence.

Semantical concerns

As pointed out at the beginning of this section, our techniques are designed for a semantics without internal communication. Myself and my coauthors have informally and incorrectly justified this choice by claiming that it only gives more power to the attacker [BDH15a]:

> [This] is reasonable when studying security protocols faced with the usual omnipotent attacker. In such a setting, we end up considering the worst-case scenario where any communication has to be made via the environment.

Such a reasoning is correct for reachability properties, where the attacker chooses a trace to reach a bad state. It is not so for equivalence properties, where the attacker chooses an observable trace to distinguish two systems, and the systems may exploit the possibility of internal communications to achieve indistinguishable behaviour. This

[4] `https://github.com/akiss/akiss/commit/fad7128c`

[5] Symmetries are often present in processes used in cryptographic protocol verification, which typically consist of parallel compositions of several instances of the same role (i.e., several renamings of the same process). Two traces of such processes are then redundant when they are equal modulo renaming, which is not captured by POR techniques. The problem can be limited if multiple sessions are represented by means of (bounded) replication rather than parallel compositions, but dedicated symmetry reduction techniques can eliminate more redundancies.

mistake has been pointed out and studied in depth in [BCK20]. In particular, it is shown that may-testing equivalence (as in definition 2.8) is incomparable with the trace equivalence derived from the LTS without internal communication (i.e. $\approx$ in this section). In other words, our POR techniques are proved correct wrt. a modified trace equivalence, but this modified equivalence is not appropriate wrt. our ultimate goal, i.e., proving may-testing equivalence. We attempt to clarify here the status of our work in light of these new observations.

The issue could be resolved, in principle, by adapting our POR techniques for the unmodified LTS, thus restoring the close correspondence between trace equivalence and may-testing equivalence (corollary 2.2). Unfortunately, our technical developments would be significantly complicated by the inclusion of unobservable internal communications. In the LTS without internal communication, all unobservable actions are only relevant to one process. This allows us to execute unobservable actions eagerly and ignore them when permuting observable ones. Internal communications obviously introduce a form of synchronization, and prevent some rule permutations that we use to comply to the compressed strategy. In order to account for internal communications, we would thus have to relax the strategy, allowing the interruption of blocks at any point to perform internal communications. It is in fact how internal communications on private channels are handled in [CKR19]. However, handling public internal communications in this way may significantly weaken the partial-order reduction achieved on many examples.

Another solution would be to show that our modified trace equivalence coincides with may-testing equivalence for a sufficiently restricted class of processes. In fact, [BCK20, corollary 2] establishes that the two equivalences coincide for *strongly action-determinate* processes, which essentially correspond to our action-deterministic processes. However, there is a gap in the proof of that result, acknowledged by the authors[6]. It is thus currently unknown whether the equivalences coincide, or even whether our modified trace equivalence implies may-testing for action-deterministic processes.

There is however a natural restriction of our class of action-deterministic processes for which our modified trace equivalence coincides with may testing equivalence. Instead of requiring that processes are action-deterministic wrt. the semantics without internal communication, consider requiring action-determinism wrt. the original semantics. For example, $\mathbf{out}(c,u).\mathbf{out}(c,v) \mid \mathbf{in}(c,x)$ is action-deterministic in the former sense but not the latter, because the output on c can be executed in two ways in the semantics with internal communication. This new take on action-determinism is actually in line with our original intuition: the system should respond deterministically when controlled by the attacker — as in standard cryptographic games[7]. Moreover, it can be shown that this "true" action-determinism condition implies a complete absence of internal communications [BCK20, lemma 6]. Hence, for the class of truly action-deterministic processes, the removal of internal communications in our LTS has no impact on the resulting trace equivalence, which coincides with may-testing equivalence.

We note that the issue with our work on POR actually remains in its continua-

[6] Private communication with Steve Kremer and Vincent Cheval, September 2020.

[7] More precisely, games in the computational model rule out non-determinism but allow probabilistic choices: systems respond probabilistically when controlled by the attacker.

tion [CKR19]. The authors of that paper propose a new way of avoiding the action-determinism condition: they define a new equivalence, called *session equivalence*[8], which implies trace equivalence; then they observe that POR techniques (including compression and reduction) are actually correct for session equivalence *without* the need for an action-determinism assumption. This is an appealing approach, because their notion of session equivalence is quite natural — for instance, it could be shown that the sufficient conditions of [HBD16] imply the session-equivalence version of unlinkability. However, our semantic concern remains: since they do not consider a semantics with (public) internal communication, there is no guarantee that their trace equivalence coincides (or even implies) the standard may-testing equivalence.

4.2 Persistent and sleep sets

We describe a second approach [BDH18] to POR for protocol equivalences. Our main goal with this approach was to lift the action-determinism condition, but it is also an opportunity to make use of classic POR concepts and explain their relationship with our previous approach.

The specific setup of [BDH18] slightly differs from the one of e.g. [BDH15a]. For the present discussion, we can assume to be working with the same LTS as before (i.e. without internal communication) except for a specific treatment of handles: instead of allowing arbitrary fresh identifiers w in **out**(c, w) actions, we assume a fixed set of handles of the form $\mathsf{w}_{c,i}$ where $c \in \mathcal{C}$ and $i \in \mathbb{N}$, and require that the i^{th} output on channel c uses the handle $\mathsf{w}_{c,i}$. This inessential difference avoids some spurious dependencies in the transition system, allowing us to directly and effectively use abstract POR techniques without having to work modulo α-equivalence.

4.2.1 Two classical POR techniques

We briefly present the notions of persistent and sleep sets [God96] which are defined for an arbitrary action-deterministic transition system. We thus assume a set of states Q, a set of transitions T, and a partial transition function $\delta : Q \times T \to Q$. We write $s \xrightarrow{\alpha} s'$ when $s' = \delta(s, \alpha)$. Given a state s, we define the set $\mathsf{En}(s)$ of *enabled transitions* in state s as $\{\alpha \mid (s, \alpha) \in \mathsf{dom}(\delta)\}$. A set s is said to be *final* when $\mathsf{En}(s) = \emptyset$.

Definition 4.6. *Independence is the greatest relation $\leftrightarrow \subseteq T \times Q \times T$ that is symmetric, irreflexive and such that, for all $\alpha \leftrightarrow_s \beta$, we have:*

- *if $s \xrightarrow{\alpha} s'$ then $\beta \in \mathsf{En}(s)$ iff $\beta \in \mathsf{En}(s')$;*
- *if $s \xrightarrow{\alpha} s_1$ and $s \xrightarrow{\beta} s_2$, then $s_1 \xrightarrow{\beta} s'$ and $s_2 \xrightarrow{\alpha} s'$ for some s'.*

[8] Session equivalence is essentially our annotated trace equivalence, but the contribution of [CKR19] is to observe that this technical device in our development can be reformulated as a meaningful equivalence for end-users.

Persistent sets

The idea of persistent sets is to select, at a given state s, a subset of enabled transitions that is sufficiently independent of other actions, so that one does not loose any reachable state when restricting to the execution of actions in persistent sets.

Definition 4.7. *A set $T \subseteq \mathsf{En}(s)$ is* persistent *in some state s_0 when, for all sequences of actions $s_0 \xrightarrow{\alpha_0} s_1 \dots s_n \xrightarrow{\alpha_n} s_{n+1}$ such that $\alpha_i \notin T$ for all $0 \leq i \leq n$, we have that $\alpha_n \leftrightarrow_{s_n} \alpha$ for all $\alpha \in T$.*

Note that $\mathsf{En}(s)$ is always persistent in s, so that any state that is not final admits a non-empty persistent set. We can thus assume some function $\mathsf{p}_{\mathsf{set}} : Q \to 2^Q$ which associates to any state $s \in Q$ such that $\mathsf{En}(s) \neq \emptyset$ a non-empty set which is persistent in s. A trace $s_0 \xrightarrow{\alpha_0} s_1 \dots \xrightarrow{\alpha_n} s_{n+1}$ is *persistent*, written $s_0 \xrightarrow{\alpha_0 \dots \alpha_n}_{\mathsf{p}_{\mathsf{set}}} s_{n+1}$, if $\alpha_i \in \mathsf{p}_{\mathsf{set}}(s_i)$ for all $0 \leq i \leq n$.

Proposition 4.1. *Let s be a state. Any final state s' that is reachable from s is also reachable from s through a trace that is persistent.*

Example 4.6. *Let $(\mathcal{P}, \Phi)$ be a configuration. Assume that it is action-deterministic, so that our LTS rooted in that configuration can be presented in terms of a δ function.*

If $\mathbf{out}(c, \mathsf{w}_{c,i}) \in \mathsf{En}(\mathcal{P}, \Phi)$, then $\{\mathbf{out}(c, \mathsf{w}_{c,i})\}$ is persistent in $(\mathcal{P}, \Phi)$: indeed, if some sequence of actions can be executed that does not contain this output, it is executed by parallel processes, and is thus independent of the output.

Assume now that $\mathcal{P} = \{\mathbf{in}(c, x).\mathbf{out}(c, u), \mathbf{in}(d, y).\mathbf{out}(d, v)\}$. We have:

$$\mathsf{En}(\mathcal{P}, \Phi) = \{\mathbf{in}(c, R), \mathbf{in}(d, R) \mid R \text{ is a } \Phi\text{-recipe}\}$$

Note that two input transitions $\mathbf{in}(c, R)$ and $\mathbf{in}(c, R')$ are dependent because one disables the other. Thus a persistent set that contains some transition $\mathbf{in}(c, R)$ must contain all transitions $\mathbf{in}(c, R')$ where R' is a Φ-recipe. Assume now that a persistent set $T \subseteq \mathsf{En}(\mathcal{P}, \Phi)$ contains $\mathbf{in}(c, R)$ for all Φ-recipes R, but but no input transition on d. Then the transitions $\mathbf{in}(d, R).\mathbf{out}(d, \mathsf{w}_{d,i}).\mathbf{in}(c, R')$ are executable, for some i, for any Φ-recipe R and Φ'-recipe R' where Φ' is the frame enriched with the output on d. These transitions are also outside T provided that R' is not a Φ-recipe, i.e. provided that it actually relies on $\mathsf{w}_{d,i}$. Observe finally that $\mathbf{in}(c, R')$ is dependent with all $\mathbf{in}(c, R'')$ transitions of T, which contradicts the fact that T is persistent. Hence, non-empty persistent sets in $(\mathcal{P}, \Phi)$ must contain all input transitions on both c and d, for all Φ-recipes.

Sleep sets

If a persistent set contains two independent actions, then the associated search has redundancies. This has lead to the introduction of *sleep sets*, described next assuming some arbitrary total ordering $<$ on transitions.

Definition 4.8. *A* sleep set execution *is an execution*

$$(s_0, \emptyset) = (s_0, z_0) \xrightarrow{\alpha_0} (s_1, z_1) \dots \xrightarrow{\alpha_n} (s_{n+1}, z_{n+1})$$

with states in $Q \times 2^T$ *such that* $s_0 \xrightarrow{\alpha_0 \dots \alpha_n}_{\mathsf{p_{set}}} s_{n+1}$, *and for any* $0 \leq i \leq n$ *we have* $\alpha_i \notin z_i$ *and*

$$z_{i+1} = \{\beta \in z_i \mid \alpha_i \leftrightarrow_{s_i} \beta\} \cup \{\beta \in \mathsf{p_{set}}(s_i) \mid \beta < \alpha_i,\ \alpha_i \leftrightarrow_{s_i} \beta\}.$$

The sets z_i are called sleep sets, and specify some subset of enabled transitions whose execution is not allowed anymore. Sleep set executions refine persistent executions by prioritizing small actions: when some action is executed, smaller independent alternatives are added to the sleep set. Transitions exit the sleep set when they are dependent with the executed transition.

Proposition 4.2. *If a final state is reachable from* s *in the original LTS, then it is also reachable from* $(s, \emptyset)$ *through a sleep set execution.*

Example 4.7. *Consider now the configuration of example 4.5:*

$$(\mathcal{P}, \Phi) \stackrel{def}{=} (\{\mathbf{in}(c, x).\mathbf{in}(c, y).\mathbf{out}(c, n), \mathbf{in}(d, z).\mathbf{out}(d, n)\}, \{n\}.\emptyset)$$

As in example 4.6, the only non-empty persistent set in $(\mathcal{P}, \Phi)$ *is* $\mathsf{En}(\mathcal{P}, \Phi)$. *Thus, regardless of the choice of* $\mathsf{p_{set}}$, *there is a persistent execution of* $\mathbf{in}(c, R_x).\mathbf{in}(d, R_z)$ *for any valid recipes* R_x *and* R_z.

Assume that the ordering on transitions is such that $\mathbf{in}(c, R) < \mathbf{in}(d, R')$ *for all* R *and* R'. *A sleep set execution could start with an input on* c:

$$((\mathcal{P}, \Phi), \emptyset) \xrightarrow{\mathbf{in}(c, R_x)} ((\mathcal{P}', \Phi), \emptyset)$$

After that, it is actually possible to execute a whole block on c, *followed by a block on* d, *keeping an empty sleep set all along.*

If we start instead with an input on d, *we obtain:*

$$((\mathcal{P}, \Phi), \emptyset) \xrightarrow{\mathbf{in}(d, R_z)} ((\mathcal{P}_1, \Phi), \{\mathbf{in}(c, R) \mid R \text{ is a } \Phi\text{-recipe}\})$$

In other words, all inputs on c *that were executable from the beginning have been put to sleep. At this point, the persistent execution would likely force the output on* d:

$$((\mathcal{P}_1, \Phi), \{\mathbf{in}(c, R) \mid R \text{ is a } \Phi\text{-recipe}\}) \xrightarrow{\mathbf{out}(d, \mathsf{w}_{d,0})} ((\mathcal{P}_2, \Phi'), \{\mathbf{in}(c, R) \mid R \text{ is a } \Phi\text{-recipe}\})$$

We can now execute inputs on c, *but only if their recipe is not a* Φ*-recipe: in other words, the inputs on* c *that were executable from the beginning are forced to be executed at that time in sleep set executions.*

It can be checked that we also have a sleep set execution of

$$\mathbf{in}(c, R_x).\mathbf{in}(d, R_z).\mathbf{out}(d, \mathsf{w}_{d,0}).\mathbf{in}(c, R_y).\mathbf{out}(c, \mathsf{w}_{c,0})$$

exactly when R_y *contains an occurrence of* $\mathsf{w}_{d,0}$.

As this example suggests, sleep set executions are very close to our reduced executions, when reduction is applied on individual actions rather than blocks (cf. section 4.1.5). Sleep sets allow further optimizations when blocks do not contain outputs.

We have actually called such blocks *improper* and considered in our works a refinement of compression where executions must stop after one such block [BDH14; BDH15b; BDH17]. Sleep sets suggest alternative treatments: improper blocks could be treated as outputs, i.e. their execution could be forced as soon as they become available. While this does not seem to be stronger in our setting, we note that the slightly different treatment of improper blocks in session equivalence [CKR19], where interleavings of improper blocks are considered, might be improved in light of the sleep set technique.

4.2.2 Trace equivalence as a reachability property

In order to apply the sleep sets technique to the verification of protocol equivalences, we reframe trace equivalence as a reachability problem in some action-deterministic transition system.

In order to account for the multiple executions that arbitrary processes may perform for a given trace, we need to consider sets of states. Starting with a formal state $\langle A \approx B \rangle$ where A and B are two configurations, we are thus lead to consider, after the execution on some trace t, formal states $\langle A_1 + \ldots + A_n \approx B_1 + \ldots + B_m \rangle$ where $A_1, \ldots, A_n$ are the configurations such that $A \xrightarrow{t} A_i$, and similarly for B. Note that when the initial processes A and B are action-deterministic, we always have $n \leq 1$ and $m \leq 1$.

We say that such a state is left-bad when some A_i is such that $\Phi(A_i) \not\sim \Phi(B_j)$ for all j, and symmetrically for right-bad states. A state is bad when it is left or right-bad. We then have $A \not\approx B$ iff some bad state is reachable from $\langle A \approx B \rangle$. Because sleep set executions only preserve the reachability of *final* states, we need to elaborate on the previous ideas to make sure that bad states remain bad after the execution of more transitions. This is done by introducing *ghost configurations*. The resulting definition of the *trace equivalence LTS* in [BDH18] is quite technical; we only illustrate its general principle here, through an example.

Example 4.8. *Assume some constants* $\mathsf{ok} \neq_{\mathsf{E}} \mathsf{ko}$ *and public channels* $c \neq d$*, and consider the following multiset of processes:*

$$\mathcal{P}(u) \stackrel{def}{=} \{\, \mathbf{in}(c,x).\mathbf{if}\ x = \mathsf{ok}\ \mathbf{then}\ \mathbf{out}(c,u),\ \mathbf{in}(c,x).\mathbf{out}(c,\mathsf{ok}).\mathbf{out}(d,\mathsf{ok}) \,\}$$

We have $(\mathcal{P}(\mathsf{ok}),\emptyset) \not\approx (\mathcal{P}(\mathsf{ko}),\emptyset)$*, because the trace* $\mathbf{in}(c,\mathsf{ok}).\mathbf{out}(c,\mathsf{w}_{c,0})$ *can lead to the output of* ko *by* $P(\mathsf{ko})$ *but not by* $P(\mathsf{ok})$*.*

In the equivalence LTS, the state $\langle (\mathcal{P}(\mathsf{ok}),\emptyset) \approx (\mathcal{P}(\mathsf{ko}),\emptyset) \rangle$ *can perform the transitions* $\mathbf{in}(c,\mathsf{ok}).\mathbf{out}(c,\mathsf{w}_{c,0}).\mathbf{out}(d,\mathsf{w}_{d,0}).\mathbf{in}(c,\mathsf{ko})$ *to yield the following final state:*

$$\begin{array}{c} \langle\ (\emptyset, \{\mathsf{w}_{c,0} \mapsto \mathsf{ok}, \mathsf{w}_{d,0} \mapsto \mathsf{ok}\}) + (\bot_3, \{\mathsf{w}_{c,0} \mapsto \mathsf{ok}, \mathsf{w}_{d,0} \mapsto \mathsf{ok}\}) \approx \\ (\emptyset, \{\mathsf{w}_{c,0} \mapsto \mathsf{ok}, \mathsf{w}_{d,0} \mapsto \mathsf{ok}\}) + (\bot_3, \{\mathsf{w}_{c,0} \mapsto \mathsf{ko}, \mathsf{w}_{d,0} \mapsto \mathsf{ok}\})\ \rangle \end{array}$$

The non-determinism associated to the execution of the first input on c *yields two configurations on each side of the state. The first configuration corresponds to a full execution of the process, where the input on* c *is followed by an output of* ok *for both* $\mathcal{P}(\mathsf{ok})$ *and* $\mathcal{P}(\mathsf{ko})$*, hence its empty set of processes. The second configuration is a ghost, corresponding to the execution of the first output on* c*, followed by the output of* u*.*

After these first two steps the processes cannot perform $\mathbf{out}(d, \mathsf{w}_{d,0})$ *hence the ghost, represented using the special symbol* $\bot_3$ *where* 3 *indicates at which step the ghost has appeared. The ghost is crucial in this example since it carries, in the final state, the frame where* $\mathsf{w}_{c,0} \mapsto \mathsf{ko}$ *which witnesses the inequivalence.*

We prove in [BDH18, Proposition 3] that $A \not\approx B$ is equivalent to the reachability of a bad final state from $\langle A \approx B \rangle$. Since the trace equivalence LTS is action-deterministic by design, the abstract sleep set technique directly applies to it. When A and B are action-deterministic, the analysis from the previous section suggests that sleep sets allow to recover a POR technique similar to the action-level variant of our reduction technique. More importantly, sleep sets also provide a framework for designing partial-order reductions without the action-determinism assumption. Much work remains to be done, however, to obtain an effective method.

4.2.3 Computing persistent sets

At this point we know that, for any choice of persistent sets, the associated sleep set executions will be sufficient to determine the existence of bad states, and hence decide trace equivalence. In order to restrict these sleep set executions, we aim to obtain persistent sets that are as small as possible[9]. In practical uses of POR, good enough persistent sets are often determined statically: like we did in example 4.6 for action-deterministic processes, one can statically determine persistent sets for programs involving mutexes, shared memory, and other concurrency mechanisms [God96] by studying theoretically the space of potential transitions. However, it seems difficult to follow the same approach with our trace equivalence LTS, where independences depend on the non-determinism of processes.

Example 4.9. *Consider a state of the trace equivalence LTS of the following form, where the right-hand side is irrelevant:*

$$\langle (\{\mathbf{out}(c, u), \mathbf{in}(c, x).\mathbf{out}(c, v)\}, \Phi) \approx \ldots \rangle$$

Transitions $\mathbf{out}(c, \mathsf{w}_{c,0})$ *and* $\mathbf{in}(c, R)$ *are enabled in that state, for any* Φ*-recipe* R*. Moreover, these transitions do not disable each other. However, the transitions are not independent because the executions of* $\mathbf{out}(c, \mathsf{w}_{c,0}).\mathbf{in}(c, R)$ *and* $\mathbf{in}(c, R).\mathbf{out}(c, \mathsf{w}_{c,0})$ *yield two different states, respectively*

$$\langle (\{\mathbf{out}(c, v')\}, \Phi_u) \approx \ldots \rangle \quad \text{and} \quad \langle (\{\mathbf{out}(c, u)\}, \Phi_{v'}) + (\{\mathbf{out}(c, v')\}, \Phi_u) \approx \ldots \rangle$$

where $v' = v\{x \mapsto R\Phi\}$ *and* $\Phi_t = \Phi \cup \{\mathsf{w}_{c,0} \mapsto t\}$ *for* $t \in \{u, v'\}$*.*

In the absence of general theoretical criteria for determining non-trivial persistent sets in the presence of such non-determinism, we set out to search for persistent sets algorithmically: for each encountered state in the trace equivalence LTS, we will compute

[9] It is not the case that smaller persistent sets always yield a smaller state space in sleep set executions [God96]. Other factors come into play, e.g. the independences between states of the persistent set. In the absence of a better criterion, aiming for small persistent sets is a common reasonable choice.

a minimal persistent set[10] by analyzing its transition graph. Such analyses cannot be done directly in the concrete trace equivalence LTS, which features arbitrary recipes. Instead, we naturally turn to a symbolic version of our LTS.

We define in [BDH18] a *symbolic* trace equivalence LTS. Its precise definition relies on common ideas, involving second-order variables representing recipes and first-order variables representing the associated messages. A set of constraints over these variables is attached to symbolic equivalence states; it applies to the variables present in each of the state's configurations. The input actions of our symbolic equivalence LTS are however formulated in a specific way, to fit our purposes: $\mathbf{in}(c, \mathsf{X}^{c,i}, W)$ denotes the i^{th} input on channel c, for a recipe using handles from W. The fixed choice of second-order variable $\mathsf{X}^{c,i}$ avoids spurious dependencies, like the fixed choice of handles in output actions $\mathbf{out}(c, \mathsf{w}_{c,i})$. Including the support W of the recipe in the action enables analyses such as the ones carried out in example 4.6, which care crucially based on the distinction between recipes available at different points of the executions, with a varying set of available handles.

Remark 4.1. *The handling of conditionals induces branching executions in the symbolic equivalence LTS. After performing an observable action, all unobservable actions are implicitly performed to simplify the resulting state, and the two possible executions of conditionals are accounted for. Consider for example a symbolic state of the following form, where $\mathcal{C}$ is a set of constraints:*

$$\langle(\{\mathbf{in}(c, x).\mathbf{if}\ x = u\ \mathbf{then}\ P_1\ \mathbf{else}\ P_2\}, \Phi) \approx \ldots\rangle_{\mathcal{C}}$$

Assuming that the elided parts of the state do not contain conditionals[11], this state can perform the transition $\mathbf{in}(c, \mathsf{X}^{c,i}, W)$ in two ways, to become either

$$\langle(\{P_1\{x \mapsto \mathsf{x}_\Phi^{c,i}\}, \Phi) \approx \ldots\rangle_{\mathcal{C} \wedge \mathsf{x}_\Phi^{c,i} = u} \quad \text{or} \quad \langle(\{P_2\{x \mapsto \mathsf{x}_\Phi^{c,i}\}, \Phi) \approx \ldots\rangle_{\mathcal{C} \wedge \mathsf{x}_\Phi^{c,i} \neq u}$$

where $\mathsf{x}_\Phi^{c,i}$ is the first-order variable associated to $\mathsf{X}^{c,i}$. The first symbolic state represents the transitions where the condition $x = u$ is evaluated successfully, and the second one represents the others. Their sets of solutions are disjoint, which must be since the concrete equivalence LTS is action-deterministic. This new kind of branching in the symbolic LTS is not an issue, because we are applying the sleep set technique on the concrete LTS, and only using the symbolic LTS as a tool to analyze the concrete one.

Remark 4.2. *The inevitable consequence of differentiating inputs based of their support is that the symbolic LTS features several redundant actions: for any given state and channel there is a most general enabled input (the one with full support) which accounts for all enabled inputs. One might worry that these redundancies might counter-balance*

[10] Our approach is similar to dynamic POR [FG05] where dependencies are computed dynamically on visited states, with more precision than could be achieved statically. Our LTS is however more complex than the standard concurrent programs of [FG05], hence our need to analyze executions beyond a visited state to obtain an interesting persistent set for it. Moreover, dynamic POR is used with concrete executions while we must carry out our analysis in symbolic semantics.

[11] Otherwise, the branching factor would increase: when n conditionals are present at toplevel, we need to consider 2^n cases.

the effect of the partial-order reductions that we are designing. This is irrelevant because, as observed in the previous remark, we are not looking for partial-order reduction in the symbolic LTS. Concretely, we justify how a single input can be considered when relevant in our algorithms, notably through the notion of enabled cover *[BDH18].*

Once this symbolic framework is in place, we can determine whether two actions are independent by executing transitions symbolically. We actually over-approximate this notion: when two symbolic actions are declared independent a symbolic state, it must be the case that independence holds in the concrete equivalence LTS for all valid instantiations of our actions and state; however, we do not aim for the converse implication. This is enough to obtain interesting results, and allows us to work while ignoring messages and recipes (except for their support) which avoids the cost of solving constraint systems[12].

We can finally search for minimal persistent sets: roughly, we initialize a set with an arbitrary enabled transition, and saturate it until obtaining a persistent set — this saturation process is described in more detail in [BDH18]. We perform this construction for all possible first actions, and keep a minimal set among the obtained ones. As before, these computations completely ignore messages.

Example 4.10. *Let* $\mathcal{P} = \{\mathbf{in}(c,x).\mathbf{out}(c,u).P_1, \mathbf{in}(d,y).\mathbf{out}(d,v).P_2\}$ *where* P_1 *and* P_2 *are arbitrary. When searching for a minimal persistent set for* $\langle(\mathcal{P},\Phi) \approx (\mathcal{P},\Phi)\rangle$*, our algorithm will seed the construction with one input action, say* $\mathbf{in}(c, \mathsf{X}^{c,0}, \emptyset)$*, and saturate it to obtain (the symbolic representation of) a persistent set. In that case the result is the full enabled cover* $\{\mathbf{in}(c, \mathsf{X}^{c,0}, \emptyset), \mathbf{in}(d, \mathsf{X}^{d,0}, \emptyset)\}$*. The same result is obtained when taking the input on* d *as seed. Indeed, the only non-empty persistent set is the full set of enabled transitions, as explained in example 4.6.*

We point out that the above computations can be performed regardless of the details of P_1 *and* P_2*: there is no need to execute these processes to notice the dependencies that force the inclusion of the other input, i.e. the dependency between* $\mathbf{in}(c, \mathsf{X}^{c,0}, \{\mathsf{w}_{d,0}\})$ *and* $\mathbf{in}(c, \mathsf{X}^{c,0}, \emptyset)$*, and symmetrically. This is achieved algorithmically by exploring possible transitions in a breadth-first fashion, which is crucial for performance.*

In our approach, we have decided to ignore constraints. This obviously simplifies our algorithms, and also renders our algorithms independent of the trace equivalence tools where our partial-order reductions will be applied: once our persistent sets are computed, we can simply restrict the explorations of the trace equivalence verifier along sleep set executions that derive from our persistent sets. There is however one major drawback: as we have seen in remark 4.1, each conditional induces a branching factor in our symbolic equivalence LTS; when several conditionals are present at toplevel, an exponential number of cases has to be considered. In the absence of any constraint resolution, there is no way to discard cases. In other words, our symbolic approach has

[12] A trace equivalence verifier like Deepsec solves sets of constraint systems at each step of the exploration of possible traces, to determine whether they are symbolically equivalent. This is equivalent to determining whether one of our symbolic states is bad. It might be worth implementing a simpler constraint resolution procedure for our analyses: for example, it would be useful to know when a symbolic state admits no solution. The risk, however, would be to duplicate some work that will be done by the trace equivalence verifiers.

created its own state explosion issue.

In order to mitigate this issue, we propose in [BDH18] a technique for collapsing conditionals. It iteratively replaces a subprocess of the form **if** φ **then** $\alpha.P$ **else** $\beta.Q$ by $\gamma.$(**if** φ **then** P **else** Q) where the action prefixes α and β are either two outputs or two inputs on the same channel, and:

- $\gamma = \alpha = \beta$ when $\alpha = \beta = \mathbf{in}(c, x)$;
- $\gamma = \mathbf{out}(c, \Delta(u, v))$ when $\alpha = \mathbf{out}(c, u)$ and $\beta = \mathbf{out}(c, v)$, where Δ is a new function symbol.

In the case where action prefixes are inputs, the modification preserves the semantics of processes. To say the same in the case of outputs, we would need to interpret Δ as a choice function which selects u when φ holds and v otherwise. This cannot be done in our symbolic framework, and this needs not be done: the role of $\Delta(u, v)$ here is to only to make sure that collapsing the two outputs does not create new independences. We show in [BDH18, Proposition 8] that our collapsing of conditionals is sound, in the sense that symbolic persistent sets computed on the modified state are also persistent sets for the initial one. This optimization is crucial in practice on most examples.

4.2.4 Integration in verification tools

Once symbolic persistent sets have been computed, we can derive from them a notion of symbolic sleep set execution, which is a straightforward symbolic representation of sets of concrete sleep set executions.

Example 4.11. *Continuing example 4.10 and assuming that $P_1 = P_2 = 0$ for simplicity, the blocks on c and d can be interleaved in two ways in the persistent and sleep set executions. Assuming that inputs on c are prioritized by $\prec$, the interleaving starting with c has empty sleep sets but not the other one, as expected:*

$$
\begin{array}{lll}
(\langle(\mathcal{P}, \Phi) \approx (\mathcal{P}, \Phi)\rangle, \emptyset) & \xrightarrow{\mathbf{in}(d, \mathsf{X}^{d,0}, \mathsf{dom}(\Phi))} & (\langle(\mathcal{P}_1, \Phi) \approx (\mathcal{P}_1, \Phi)\rangle, \{\mathbf{in}(c, \mathsf{X}^{c,0}, \mathsf{dom}(\Phi))\}) \\
 & \xrightarrow{\mathbf{out}(d, \mathsf{w}_{d,0})} & (\langle(\mathcal{P}_2, \Phi') \approx (\mathcal{P}_2, \Phi')\rangle, \{\mathbf{in}(c, \mathsf{X}^{c,0}, \mathsf{dom}(\Phi))\}) \\
 & \xrightarrow{\mathbf{in}(c, \mathsf{X}^{c,0}, \mathsf{dom}(\Phi'))} & (\langle(\mathcal{P}_3, \Phi') \approx (\mathcal{P}_3, \Phi')\rangle, \emptyset) \\
 & \xrightarrow{\mathbf{out}(c, \mathsf{w}_{c,0})} & (\langle(\mathcal{P}_4, \Phi'') \approx (\mathcal{P}_4, \Phi'')\rangle, \emptyset)
\end{array}
$$

Intuitively, this symbolic sleep set execution expresses that, in concrete traces, the only inputs on c allowed after the block on d should be concretizations of $\mathbf{in}(c, \mathsf{X}^{c,0}, \mathsf{dom}(\Phi'))$ *but not of* $\mathbf{in}(c, \mathsf{X}^{c,0}, \mathsf{dom}(\Phi))$*, i.e. their recipe must contain an occurrence of* $\mathsf{w}_{d,0}$.

Our construction guarantees that any bad state in the concrete trace equivalence LTS can be found through one of the concretizations of a symbolic sleep set execution [BDH18, Proposition 7]. This can be used to restrict the exploration of the trace equivalence verification tools that proceed by enumerating symbolic traces.

We have implemented our method in an OCaml library called Porridge[13] [**porridge**], which weighs about 6 000 lines of code. Although it is only a prototype, we have taken

[13] Porridge is also available on `https://github.com/LCBH/deepsec`.

Test	Size	Time (ratio)	Explorations (ratio)	Time (s)
BAC (unlinkability)	4	7.6	7.23	12.23
Private Auth. (anonymity)	2	1.25	2.71	0.04
Private Auth. (anonymity)	3	1.67	4.01	0.04
Private Auth. (anonymity)	4	8.21	10.51	1.17
Private Auth. (anonymity)	5	14.89	16.61	10.57
Private Auth. (anonymity)	6	60.2	36.75	4864
Private Auth. (unlinkability)	2	2.29	9.6	0.16
Private Auth. (unlinkability)	3	14.06	29.77	79.57
Private Auth. (unlinkability)	4	46.2	46.69	7171
Feldhofer (anonymity)	2	1	4.72	0.03
Feldhofer (anonymity)	3	4.63	7.08	0.37
Feldhofer (anonymity)	4	22.47	16.3	544.93
Feldhofer (unlinkability)	4	36.27	22.58	1510.09

Table 4.1: Relative speed-up and reduction of explorations with Porridge vs. without Porridge. In the last column, we show the computation time without Porridge. The size refers to the total number of processes in parallel.

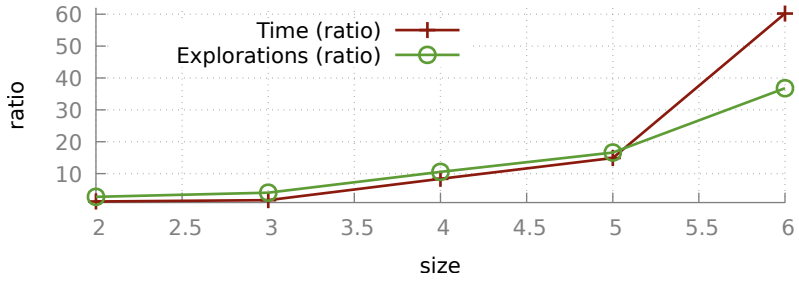


Figure 4.2: Relative speed-up and reduction of explorations with Porridge vs. without Porridge on Private Authentication (anonymity) of different sizes.

some care to make it efficient. As pointed out earlier, the symbolic state spaces that are explored are often huge. It is thus important to only explore transitions that need be, and to identify redundant computations — this is notably achieved through a careful use of hash-consing, memoization, and priority queues. Our implementation is also modular and re-usable: persistent set computation is provided as a functor, and works over an arbitrary symbolic LTS; we also provide a functor that turns a symbolic LTS into the LTS of its sleep set executions. This generality has already been exploited to implement partial-order reductions for the trace inclusion LTS (which differs from the trace equivalence LTS, and in particular does not induce the same independences) and could also be used to apply our method for trace equivalence in the standard LTS where internal communication is allowed.

Our release of Porridge [**porridge**] comes with a modified version of Deepsec with support for our new POR technique. We have evaluated the improvements brought by our technique over several examples. We reproduce in figure 4.2 and table 4.1 results from [BDH18], verifying anonymity and unlinkability on the BAC, Private Authentication and Feldhofer protocols — all these examples are not action-deterministic. The size of an example is the number of parallel sessions. Ratios compare the verification time (resp. number of explored states) without and with Porridge.

Our techniques bring speed-ups, though not on the smaller examples, e.g. Private Authentication with size 2 and 3. In these cases, the cost of the analysis performed by Porridge exceeds the benefit of the partial-order reduction when Deepsec verifies the equivalence. On larger examples the time spent computing symbolic sleep set executions in Porridge does increase, but the impact of the partial-order reduction counter-balances it. This trend is visible in Figure 4.2; we could not run large enough examples to confirm an exponential speed-up.

4.2.5 Discussion

This second approach to POR, building on the abstract techniques of persistent and sleep sets, and the algorithmic analysis of a trace equivalence LTS, has been validated by effective partial-order reductions in Deepsec. However, improvements are not as important as with our first technique for action-deterministic processes. The cost of the pre-computation of sleep set executions is very high, and it is often unfeasible on examples where the collapse of conditionals does not completely eliminate them. On action-deterministic processes, our first optimization vastly outperforms the second one.

There is room for improvements. A theoretical improvement would be to adapt our techniques to allow first-order rather than second-order dependency analyses. This would make it possible to generate, as part of our sleep set executions, dependency constraints that would be usable by tools such as Deepsec. The most promising avenue, however, would be to integrate our analyses more tightly into a trace equivalence verifier. This would make it possible to eliminate symbolic states when they have no solutions, and avoid the symbolic state space explosion caused by conditionals. However, such an integration would require to handle the complex details of the verifier's symbolic semantics, in the code and probably in the theory. A further difficulty is that our persistent set computations involve the exploration of symbolic executions before the explorations performed for trace equivalence verification: some of our explorations will turn out to be useless (because the POR avoids them) while some others will be explored later for trace equivalence verification. Achieving good performances in such a system would likely rely on delicate trade-offs between precision and efficiency, and a new architecture of the verifiers where constraint solving capabilities would not be tied to the trace equivalence verification procedure.

At a theoretical level, this second line of work has interestingly allowed to compare our compression and reduction techniques with the persistent and sleep set techniques. As illustrated above, sleep sets are similar to our reduced executions, but different. The alternative viewpoint provided by sleep sets might be useful in future developments.

4.3 Conclusion

I have presented my work on two approaches to POR with effective applications to the verification of trace equivalence for bounded processes. The first method relies on an action-determinism assumption and has been widely adopted. The second method lifts this assumption but has not been adopted; it is costly and only beneficial on large

examples where conditionals can be collapsed. We have identified some directions for improving that second approach. While these interesting technical challenges could bear some fruits, some recent and ongoing developments seem more promising.

First, session equivalence provides an attractive alternative to trace equivalence which diminishes the need for POR techniques beyond compression and reduction. Indeed, this stronger equivalence seems to be sufficient for many applications in security including anonymity and unlinkability, and it allows the compression and reduction techniques.

Second, a new generation of trace equivalence verifiers might arise which does not need POR techniques at all. The SAT-Equiv system already falls in this category. Moreover, the procedure of Akiss is being redesigned to work on partial orders rather than traces, which avoids the prior enumeration of symbolic traces. This ongoing development, which has involved Gazeau, Kremer and Laporte, is now close to completion and preliminary experimental results show massive improvements[14]. If this line of work proves successful, our POR techniques will become less relevant, but we will have contributed to bringing concurrency issues to the front stage in the domain of protocol equivalence verification.

On a completely different note, a possible continuation of our works on POR would be to bring ideas back to the field of proof search. Compression has been adapted from the logical notion of focusing, and we have then enriched it with reduction. Our second line of work also points to connections between persistent sets and focusing. It would be theoretically interesting to devise analogues of these techniques in proof search. One might for instance try to adapt the notion of lexicographically minimal trace to that of minimal derivation tree. Such results could then be used to devise new proof search techniques, probably starting with linear logic before moving on to richer settings. As in our POR methods, this would require to work symbolically, which is natural in proof search, where meta-variables and unification are classical tools.

[14] This information comes from a private communication with Steve Kremer in October 2020.

5

A Meta-Logic for Proving Protocols in the Computational Model

Squirrel!

Dogs in *Up*, Pixar, 2009

The previous three chapters deal exclusively with the symbolic model. This model, or rather this class of models, stems from the seminal idea of Dolev and Yao [DY81]: attacker capabilities can be represented symbolically using e.g. equational theories or sets of inference rules. This has made it possible to leverage techniques from, e.g., automated deduction and rewriting, to obtain automated procedures for the verification of security protocols. The resulting tools have been used to discover important flaws in real-world security protocols, and provide proofs which increase our confidence in them.

However, the symbolic modelling of attacker capabilities is usually incomplete wrt. the computational model, i.e. the standard model of cryptographers where messages are bitstrings and primitives, protocols and attackers are probabilistic PTIME Turing machines. We have seen a concrete example of incompleteness in section 3.3.1, where Proverif missed some attack on the PACE e-passport protocol (later discovered thanks to Tamarin) because it can only handle a subset of the Diffie-Hellman equational theory. The question of *computational soundness* has been raised in a seminal paper by Abadi and Rogaway [AR07] with a striking positive result: they showed that, despite the widely different settings, computational indistinguishability coincides with static equivalence for standard cryptographic primitives. This work has inspired many others, dealing with active adversaries [MW04a], observational equivalence [CC08], Diffie-Hellman [Bre+11], etc. However, computational soundness remains difficult to achieve, and results have a limited practical impact so far due to strong assumptions such as the absence of

key cycles [CC08], or modularity issues [CW11]. Actually, incompleteness is sometimes inevitable: this is the case for exclusive or, at least when sessions are unbounded [Unr10].

Direct approaches have thus been proposed to use formal methods for the computational analysis of protocols [CKW11; Bar+19]. In particular, the systems Cryptoverif [Bla06], Easycrypt [Bar+11], CryptHOL [BLS20] and F$\star$ [Bha+17] are quite successful at proving primitives and protocols, with varying levels of detail and automation; we discuss them in more detail below.

Together with Stéphanie Delaune, Charlie Jacomme, Adrien Koutsos and Solène Moreau, we have recently developed a new approach to obtain computational proofs of cryptographic protocols. It is based on the logic proposed by Bana and Comon [BC14; Ban+20] for equivalence properties. The Bana-Comon logic, which we briefly present in section 5.1, deals with indistinguishabilities between sequences of messages. To effectively use it to prove reachability properties and protocol equivalences, we have designed a meta-logic and corresponding proof systems, which we describe in section 5.2. We have also developed an interactive proof assistant for this meta-logic, presented in section 5.3, which has allowed us to demonstrate the usefulness of our method on various case studies. We conclude the chapter in section 5.4, where we compare our approach with other systems for proving protocols in the computational model, and discuss the research questions that stem from this new development.

The contents of this chapter have been submitted anonymously in June 2020 at the IEEE Symposium on Security and Privacy, and are currently undergoing a revision. A long version of the submission, together with our source code and case studies are available at `https://github.com/squirrel-submission-sp21/squirrel-prover`.

5.1 Base logic

The symbolic model is sometimes presented as a black-box idealization of the computational model [CKW11]. The idealization is visible already with the treament of secrets, modelled as names. In the symbolic model, the attacker cannot deduce the name n from $\mathsf{senc}(n, k)$, provided that k has only been used as encryption key in output messages. In the computational model, a name n stands for a bitstring of length η (the *security parameter*) that is chosen uniformly at random by the protocol. The attacker could randomly guess this value, but this is unlikely: more formally, this can only occur with a probability that is negligible[1] in η. The black-box nature of the symbolic model comes from the fact that attacker capabilities correspond to the functionalities of cryptographic primitives, ignoring weaknesses of potential implementations. This is very visible with (keyed) hash functions, for which there is no functionality. In the symbolic model, hash functions come with no equational theory. As a consequence, they satisfy a form of collision resistance, i.e. $\mathsf{h}(u, k) =_{\mathsf{E}} \mathsf{h}(v, k)$ implies $u =_{\mathsf{E}} v$, but are also indistinguishable from random functions, i.e. $\{k\}.\{w \mapsto h(u, k)\} \sim \{k\}.\{w \mapsto n\}$. In the computational model, asking that a keyed hash function is a pseudo-random function is very strong, and

[1] We say that a quantity $\lambda(\eta)$ is negligible in η when, for any positive polynomial $p(\eta)$, there is some η_p such that $\lambda(\eta) < 1/p(\eta)$ for any $\eta > \eta_p$.

one may wish to prove protocols secure under weaker assumptions such as unforgeability or collision-resistance — note, for instance, that a hash function could leak one bit of the hashed value while remaining unforgeable and collision-resistant.

The approach of Bana and Comon, which they call *computationally complete symbolic attacker*, is based on a crucial idea: instead of specifying symbolically what the attacker can do, as Dolev and Yao proposed, a computationally sound logic should specify what the attacker cannot do. Bana and Comon have first designed in this way a computationally sound logic for reachability properties [BC12], and then adapted the approach for equivalence properties [BC14]. Unusually, the latter system is actually simpler than the former, both in terms of justifying and using it [Ban+20]. We give below a short introduction to the Bana-Comon logic for equivalences, which we call our *base logic*. We follow the notations and definitions of [BC14] with only slight modifications to avoid ambiguities between our base and meta-logic.

5.1.1 Syntax and semantics

The base logic is simply an instance of first-order logic, with a single sort message, and a single predicate representing indistinguishability. We will be interested in the satisfaction of formulas in particular structures, called *computational models*, where terms are interpreted as probabilistic polynomial-time (PPTIME) Turing machines, and indistinguishability is interpreted as computational indistinguishability wrt. PPTIME adversaries.

In order to be able to suitably express cryptographic computations using terms, we split function symbols in three categories[2]:

1. A set $\mathcal{F}_B$ of symbols of arbitrary arity, meant to model non-adversarial computations such as cryptographic primitives. We assume that this set contains some basic symbols:

   - pairing $\langle \_, \_ \rangle$, projection functions $\mathsf{fst}(\_)$ and $\mathsf{snd}(\_)$;
   - equality $\mathsf{EQ}(\_, \_)$;
   - conditional $\mathsf{if}\ \_\ \mathsf{then}\ \_\ \mathsf{else}\ \_$ and boolean constants true and false.

   Symbols representing cryptographic primitives will be noted as in other chapters, e.g., aenc, adec, pub or h. We use the letter f to denote arbitrary symbols of $\mathcal{F}_B$.

2. A set $\mathcal{N}_B$ of name symbols, of arity zero, noted $\mathbf{n}$, $\mathbf{m}$, $\mathbf{k}$.

3. A special attacker symbol $\mathbf{att}(\_)$.

For instance, one may write $\mathsf{EQ}(\mathbf{att}(\mathsf{h}(\mathbf{n}, \mathbf{k})), \mathbf{n})$ which intuitively stands for the computation that compares some uniformly sampled nonce $\mathbf{n}$ with $\mathbf{att}(\mathsf{h}(\mathbf{n}, \mathbf{k}))$, i.e., the result of some arbitrary computation performed by an attacker who has been given the hash of $\mathbf{n}$ using key $\mathbf{k}$.

[2] We use a specific style for names from now on, writing e.g. $\mathbf{n}$ instead of n. This serves as a reminder that, although names and function symbols are both represented in the Bana-Comon framework as function symbols of first-order logic, these two kinds of symbols have different semantics.

We are interested in interpretations of terms as probabilistic computations represented by deterministic Turing machines featuring, in addition to their input and working tapes, three special tapes: the first one is used to pass the security parameter η in unary; the second and third one are infinite tapes that will serve as sources of randomness for the honest and adversarial computations. We define PPTIME Turing machines as such machines that run in polynomial time wrt. η. As a particular case, PTIME Turing machines are PPTIME Turing machines without the two random tapes.

Definition 5.1. *A* computational model $\mathcal{M}$ *is given by:*

- *for each $f \in \mathcal{F}_B$ of arity k, a PTIME Turing machine $\mathcal{A}_f^{\mathcal{M}}$ with k input tapes;*
- *a PPTIME Turing machine $\mathcal{A}_{\mathbf{att}}^{\mathcal{M}}$ with a single input tape.*

We assume that the standard symbols EQ, true, false *and* if _ then _ else _, *are interpreted in a standard way, e.g.* EQ *implements bitstring comparison.*

Definition 5.2. *The* interpretation *of a term t in some computational model $\mathcal{M}$ is defined as follows, given a semantic assignment σ which maps variables to PPTIME machines:*

$$\begin{aligned}
[\![x]\!]_\sigma^{\mathcal{M}}(\eta,\rho_1,\rho_2) &\stackrel{def}{=} \sigma(x)(\eta,\rho_1,\rho_2)\\
[\![f(t_1,\ldots,t_k)]\!]_\sigma^{\mathcal{M}}(\eta,\rho_1,\rho_2) &\stackrel{def}{=} \mathcal{A}_f^{\mathcal{M}}([\![t_1]\!]_\sigma^{\mathcal{M}}(\eta,\rho_1,\rho_2),\ldots,[\![t_k]\!]_\sigma^{\mathcal{M}}(\eta,\rho_1,\rho_2),\eta)\\
[\![\mathbf{att}(t)]\!]_\sigma^{\mathcal{M}}(\eta,\rho_1,\rho_2) &\stackrel{def}{=} \mathcal{A}_{\mathbf{att}}^{\mathcal{M}}([\![t]\!]_\sigma^{\mathcal{M}}(\eta,\rho_1,\rho_2),\eta,\rho_2)\\
[\![\mathbf{n}]\!]_\sigma^{\mathcal{M}}(\eta,\rho_1,\rho_2) &\stackrel{def}{=} \rho_1[\mathbf{n}]
\end{aligned}$$

where $\rho_1[\mathbf{n}]$ is an arbitrary portion of ρ_1 of length η, such that the portions corresponding to different names never overlap.

The base logic features a single predicate $\sim$ or rather, for each $k \in \mathbb{N}$, a predicate $\sim_k$ of arity $2k$. These predicates are interpreted, in all computational models, as computational indistinguishability. We note $\Pr(\rho_1,\rho_2 : \mathcal{A}(\eta,\rho_1,\rho_2))$ the probability that the machine $\mathcal{A}$ accepts, wrt. an appropriate probability distribution of the two infinite tapes ρ_1 and ρ_2.

Definition 5.3. *The atom $u_1,\ldots,u_k \sim v_1,\ldots,v_k$ is* satisfied *in a computational model $\mathcal{M}$ when, for any PPTIME machine $\mathcal{A}$, the following advantage is negligible in η:*

$$\begin{aligned}
|\Pr\big(\rho_1,\rho_2 : \mathcal{A}([\![u_1]\!](\eta,\rho_1,\rho_2),\ldots,[\![u_k]\!](\eta,\rho_1,\rho_2),\eta,\rho_2)\big) -\\
\Pr\big(\rho_1,\rho_2 : \mathcal{A}([\![v_1]\!](\eta,\rho_1,\rho_2),\ldots,[\![v_k]\!](\eta,\rho_1,\rho_2),\eta,\rho_2)\big)|
\end{aligned}$$

Once satisfaction is defined for atoms, it is extended in the usual way to all formulas. For instance, $\mathcal{M},\sigma \models \forall x.\phi$ when $\mathcal{M},\sigma\{x \mapsto \mathcal{A}\} \models \phi$ for any PPTIME machine $\mathcal{A}$. As usual, we say that a formula is valid when it is satisfied in all computational models. We say that a formula ϕ is a logical consequence of the set of formulas S, noted $S \models \phi$, when all models of S are also models of ϕ.

Example 5.1. *Assume that* **n** *and* **m** *are two distinct names. The formulas* $\mathbf{n} \sim \mathbf{m}$ *and* $\mathrm{EQ}(\mathbf{n},\mathbf{m}) \sim$ false *are valid: indeed, the attacker cannot distinguish between two random samplings with the same distribution, and there is a negligible probability that two independent uniform samplings of length η coincide.*

The if _ then _ else _ function symbol allows to define other boolean constructs. We write $u \mathbin{\dot\wedge} v$ for if u then v else false, and define similarly $u \mathbin{\dot\vee} v$ and $u \mathbin{\dot\Rightarrow} v$. Finally, we write $u \doteq v$ for $\mathsf{EQ}(u, v)$. These constructs allow to write propositional formulas over equalities as terms, which will be useful to express reachability properties using indistinguishability. For example, $\left(\mathbf{att}(\mathsf{h}(t, \mathsf{k})) \doteq \mathsf{h}(t', \mathsf{k}) \mathbin{\dot\Rightarrow} t \doteq t'\right) \sim \mathsf{true}$ expresses that, with overwhelming probability, if the attacker can obtain the hash of t' from the hash of t, then t is equal to t'.

Example 5.2. *Consider the following base logic formulas:*

$$(u \sim \mathsf{true}) \Rightarrow (v \sim \mathsf{true}) \quad (a)$$
$$(u \mathbin{\dot\Rightarrow} v) \sim \mathsf{true} \quad (b)$$

Formula (a) is a logical consequence of (b): if both $u \mathbin{\dot\Rightarrow} v$ and u are true with overwhelming probability, then it must also be the case for v.

However, (a) does not generally imply (b). Consider a unary function symbol f and a model $\mathcal{M}$ where f is interpreted as the machine that returns the first bit of its argument. Then, for any arbitrary name n, the term $f(\mathsf{n})$ is interpreted as the probabilistic computation returning 1 with probability $\frac{1}{2}$, and 0 otherwise. We have $\mathcal{M} \not\models (f(\mathsf{n}) \sim \mathsf{true})$ hence formula (a) is satisfied in $\mathcal{M}$ when $u := f(\mathsf{n})$, regardless of v. However, $\mathcal{M} \not\models (f(\mathsf{n}) \mathbin{\dot\Rightarrow} \mathsf{false}) \sim \mathsf{true}$. In words, $f(\mathsf{n})$ is not true with overwhelming probability, but it is also not false with overwhelming probability.

5.1.2 Cryptographic assumptions

We are usually interested in the validity of a formula in a class of computational models, typically restricting to those models where some symbol interpretations satisfy some cryptographic assumptions. In order to symbolically capture such assumptions, axiom schemes are synthesized from cryptographic assumptions.

Example 5.3. *Let t be a ground term where the names k and n do not occur. The formula $\mathsf{h}(t, \mathsf{k}) \sim \mathsf{n}$ is not valid, but it is satisfied in all models where h implements a keyed hash function satisfying the PRF assumption, i.e. it is a pseudo-random function family. Our formula is actually a logical consequence of the following axiom scheme, which is proved sound wrt. our class of models [CK17, Proposition 3]:*

$$\vec{u}, \text{if } c \text{ then true else } \mathsf{h}(t, \mathsf{k}) \sim \vec{u}, \text{if } c \text{ then true else } \mathsf{n}$$

where $\vec{u}$, t and c are (sequences of) ground terms such that

- *k only occurs in key position in $\vec{u}$, t and c;*
- *n does not occur in $\vec{u}$, t and c;*
- *c is $\dot\bigvee_{1 \le i \le n} \mathsf{EQ}(t, t_i)$ where $\mathsf{h}(t_1, \mathsf{k}), \ldots, \mathsf{h}(t_n, \mathsf{k})$ are all occurrences of hashes with key k in $\vec{u}$ and t.*

The framework generally allows natural formulations of cryptographic assumptions as axioms. Several standard cryptographic assumptions have been axiomatized, e.g. collision resistance (CR-HK), pseudo-random function family (PRF), unforgeability under chosen-message attack (EUF-CMA), indistinguishability under chosen-plaintext and chosen-ciphertext attacks (IND-CPA, IND-CCA1 and IND-CCA2) key privacy (ENC-KP) and decisional Diffie-Hellman (DDH) [BC14; CK17; Kou19b; Ban+20].

5.1.3 Analyzing protocols

In [BC14], Bana and Comon show how their logic can be used to prove equivalences between protocols with bounded sessions. They assume that protocols are represented as simple transition system and compute, for each protocol P, a term Φ_P which intuitively represents the frame that the attacker will obtain, depending on the choices of actions to execute and of input messages. This encoding is such that two protocols P and P' are computationnally indistinguishable iff $\Phi_P \sim \Phi_{P'}$ is valid. Since only a finite number of traces can be chosen, this is actually equivalent to showing, for each trace t, that $\Phi_P^t \sim \Phi_{P'}^t$ is valid, where Φ_P^t encodes the frame obtained by the attacker for the execution of t, which depends on the choice of input messages by the attacker. Concretely, these inputs are simply encoded as $\mathbf{att}(\Phi_P^{t_i})$ for all prefixes t_i of t: they are the result of an arbitrary adversarial computation from past outputs.

This method has been used successfully to give formal proofs for various protocols, including (fixed versions of) the RFID protocols LAK and KCL [CK17], the FOO electronic voting protocol [BCE18] and the 5G AKA protocol [Kou19b]. In some cases, only a particular number of sessions is considered, e.g. one session with two honest voters and a dishonest one for the FOO protocol [BCE18]. In other cases, a parametric proof is given, e.g. it is shown that a modified version of the KCL protocol is unlinkable for any fixed number of sessions [CK17]. In all cases, proofs are formal but manual, which is both tedious and error-prone.

In their seminal paper, Bana and Comon conclude with hopes that proofs in their logic could be automated and implemented efficiently. This has been tackled by Koutsos, who came up with a 3-NExpTime procedure deciding provability in the Bana-Comon logic for some fixed set of axioms [Kou19a]. The limitations of this theoretical result do not mean that automatic provers cannot be built, but attempts have not been successful so far.

A less ambitious goal is to only aim for machine-checked proofs, i.e. design a proof assistant which will be guided by an expert user for the key steps of proofs, but which might be able to automatically fill in the tedious details. Unfortunately, the verification method described before does not fit well interactive proofs: it is not realistic to require users to interactively prove one indistinguishability goal for each trace of the protocol, as the number of goals would be prohibitive and the proofs would be very repetitive. We have proposed to avoid this problem by designing a meta-logic which can uniformly describe all such goals, and to prove them without explicitly considering irrelevant details of execution traces.

```
hash h
abstract ok : message
abstract error : message
name key : index → message
channel c

process tag(i,j:index) =
  new n; out(c, ⟨n,h(n,key(i))⟩)

process reader(j:index) =
  in(c,x);
  try find i such that snd(x) = h(fst(x),key(i))
  in out(c,ok)
  else out(c,error)

system (!j R: reader(j) | !i !j T: tag(i,j)).
```

Listing 5.1: Basic Hash protocol in Squirrel

5.2 Meta-logic

Our meta-logic is a language which allows to conveniently specify properties protocol executions. In order to illustrate the various concepts involved in its definition, we shall use the Basic Hash protocol. More specifically, we consider the multiple-session process for this protocol, given in listing 5.1 using the concrete syntax of the Squirrel prover; as we shall see, our specification uses a variant of the applied pi-calculus for convenience but our tool internally relies on a more primitive description of systems.

5.2.1 Syntax

Our meta-logic has three sorts. In addition to message, we consider terms of sort timestamp for representing points in the execution of a protocol, and index for indexing unbounded collection of objects. Variables of sort index are noted i, j, k; variables of sort timestamp are noted τ; variables of sort message are noted x, y, z.

The syntax of our meta-terms and meta-formulas is given in figure 5.1. There are no constructs for terms of sort index other than variables: indices are purely abstract. For terms of sort timestamp, noted T, we have a constant init which represents the beginning of the execution, and a predecessor operation pred. We also assume some set of *action symbols* for describing the possible actions of a protocol, each one coming with an index arity: an action symbol a of arity k describes one kind of action of the protocol, whose instances $\mathsf{a}[i_1, \ldots, i_k]$ are timestamps.

Example 5.4. *In listing 5.1, we have an unbounded number of copies* `tag(i,j)` *of the tag role: intuitively,* i *corresponds to the identity of the tag and* j *identifies a session. Each copy can perform a single action, identified by the timestamp* $\mathsf{a}_T[i, j]$*. For the reader we would have two actions:* $\mathsf{a}_R[j, i]$ *corresponding to when* `reader(j)` *successfully finds*

$$
\begin{array}{rcl}
T & := & \tau \mid \mathsf{init} \mid \mathsf{a}[i_1, \ldots, i_k] \mid \mathsf{pred}(T) \\
t & := & x \mid \mathbf{n}[i_1, \ldots, i_k] \mid \mathsf{f}[i_1, \ldots, i_k](t_1, \ldots, t_n) \\
 & \mid & \mathsf{input}@T \mid \mathsf{output}@T \mid \mathsf{frame}@T \\
 & \mid & \mathsf{if}\ \phi\ \mathsf{then}\ t\ \mathsf{else}\ t' \\
 & \mid & \mathsf{find}\ \vec{i}\ \mathsf{suchthat}\ \phi\ \mathsf{in}\ t\ \mathsf{else}\ t' \\
A & := & t = t' \mid i = i' \mid T = T' \mid T \leq T' \mid \mathsf{cond}@T \mid \mathsf{exec}@T \\
\phi & := & A \mid \top \mid \bot \mid \phi \wedge \phi' \mid \phi \vee \phi' \mid \phi \Rightarrow \phi' \mid \neg\phi \mid \forall i.\phi \mid \exists i.\phi \mid \forall \tau.\phi \mid \exists \tau.\phi
\end{array}
$$

Figure 5.1: Syntax of meta-terms and meta-formulas

some i and outputs ok*, and* $\mathsf{a}_{R1}[j]$ *when no such i is found.*

Meta-terms of sort message extend terms of the base logic with indexed name and function symbols. We treat conditionals as a special construct, and also consider a more general find construct which allows to search for suitable indices among an unspecified but finite domain. Finally, we consider *macros* $\mathsf{input}@T$, $\mathsf{output}@T$ and $\mathsf{frame}@T$ which will represent the input, output and frame at some point T of the protocol's execution.

Finally, meta-formulas are first-order formulas where quantification is only allowed over sorts index and timestamp. Atoms may be equalities over meta-terms or inequalities over timestamps. We also include two macros: $\mathsf{cond}@T$ stands for the executability condition of the action at T, and $\mathsf{exec}@T$ is the conjunction of all such conditions until T.

Example 5.5. *The following meta-formula expresses a simple authentication property for the Basic Hash protocol:*

$$
\begin{array}{l}
\forall i.\forall j.\ \mathsf{cond}@\mathsf{a}_R[j,i] \Rightarrow \big(\ \exists j'.\ \mathsf{a}_T[i,j'] < \mathsf{a}_R[j,i] \wedge \\
\qquad\qquad \mathsf{fst}(\mathsf{input}@\mathsf{a}_R[j,i]) = \mathsf{fst}(\mathsf{output}@\mathsf{a}_T[i,j']) \wedge \\
\qquad\qquad \mathsf{snd}(\mathsf{input}@\mathsf{a}_R[j,i]) = \mathsf{snd}(\mathsf{output}@\mathsf{a}_T[i,j'])\ \big)
\end{array}
$$

The semantics of our meta-terms and formulas can only be defined wrt. some protocol. A protocol $\mathcal{P}$ must be defined by specifying, for each action symbol a, an action $\mathsf{a}[\vec{i}].(\phi, o)$ where ϕ is a meta-formula specifying the executability condition of the action and o is a meta-term of sort message specifying the output of the action. These actions must also be equipped with a partial order $<$ indicating their dependencies. Moreover, each action $\mathsf{a}[\vec{i}].(\phi, o)$ may only mention its dependent actions: the only timestamps allowed in ϕ and o are the $\mathsf{a}'[\vec{j}]$ such that $\mathsf{a}'[\vec{j}] < \mathsf{a}[\vec{i}]$, and $\mathsf{a}[\vec{i}]$ itself when used in $\mathsf{input}@\mathsf{a}[\vec{i}]$.

Example 5.6. *The Basic Hash protocol is modelled by the following three actions, with*

an empty dependency relation:

$$\mathsf{a}_T[i,j].\big(\ \top,\ \langle \mathbf{n}[i,j], \mathsf{h}(\mathbf{n}[i,j], \mathbf{key}[i])\rangle\ \big)$$
$$\mathsf{a}_R[j,i].\big(\ \mathsf{snd}(\mathsf{input@a}_R[j,i]) = \mathsf{h}(\mathsf{fst}(\mathsf{input@a}_R[j,i]), \mathbf{key}[i]),\ \mathsf{ok}\ \big)$$
$$\mathsf{a}_{R1}[j].\big(\ \forall i.\ \mathsf{snd}(\mathsf{input@a}_{R1}[j]) \neq \mathsf{h}(\mathsf{fst}(\mathsf{input@a}_{R1}[j]), \mathbf{key}[i])\big),\ \mathsf{error}\ \big)$$

5.2.2 Semantics

The semantics of a meta-formula will be given by translating it to a base logic term. This translation will be relative to one possible execution of our protocol, given through the notion of trace model.

Definition 5.4. *A trace model $\mathbb{T}$ wrt. some protocol $\mathcal{P}$ is a tuple $(D_\mathcal{I}, D_\mathcal{T}, <_\mathcal{T}, \sigma_\mathcal{I}, \sigma_\mathcal{T})$ such that:*

- *$D_\mathcal{I} \subseteq \mathbb{N}$ is a finite index domain;*
- *$<_\mathcal{T}$ is a total ordering on*

  $$D_\mathcal{T} := \{\mathsf{init}\} \uplus \{\mathsf{a}[k_1, \ldots, k_n] \mid \mathsf{a} \in \mathcal{A}, k_1, \ldots, k_n \in D_\mathcal{I}\}$$

  which respects the dependency ordering $<$ of $\mathcal{P}$, and such that init *is minimal;*
- *$\sigma_\mathcal{I} : \mathcal{I} \to D_\mathcal{I}$ and $\sigma_\mathcal{T} : \mathcal{T} \to D_\mathcal{T}$ are mappings that interpret index and timestamp variables as elements of their respective domains.*

In other words, a trace model fixes a finite set of indices, which induces a finite set of concrete actions $\mathsf{a}[n_1, \ldots, n_k]$ for $n_i \in D_\mathcal{I}$. The trace model also fixes an interleaving of these concrete actions. A suffix of this interleaving is usually clearly not executable: for example, with the Basic Hash protocol, we are forced to schedule both $\mathsf{a}_R[n,m]$ and $\mathsf{a}_{R1}[n]$ even though they can never both execute. This is not an issue: what matters is that any possible execution is represented as the prefix of the interleaving of some trace model.

Given a protocol $\mathcal{P}$, a trace model $\mathbb{T} = (D_\mathcal{I}, D_\mathcal{T}, <_\mathcal{T}, \sigma_\mathcal{I}, \sigma_\mathcal{T})$ and a meta-term t, we can define the translation $(t)^\mathbb{T}_\mathcal{P}$, which is a base logic term over a signature which depends on the index domain $D_\mathcal{I}$. We notably have:

$$(\mathbf{n}[\vec{i}])^\mathbb{T}_\mathcal{P} \stackrel{def}{=} \mathbf{n}_{\sigma_\mathcal{I}(i_1),\ldots,\sigma_\mathcal{I}(i_k)}$$
$$(f[\vec{i}](t_1, \ldots, t_n))^\mathbb{T}_\mathcal{P} \stackrel{def}{=} f_{\sigma_\mathcal{I}(i_1),\ldots,\sigma_\mathcal{I}(i_k)}\big((t_1)^\mathbb{T}_\mathcal{P}, \ldots, (t_n)^\mathbb{T}_\mathcal{P}\big)$$

Conditionals are translated to conditionals, and find is translated as a sequence of nested conditionals, enumerating all possible tuples over $D_\mathcal{I}$. The output and frame macros are translated using the protocol definitions, and $(\mathsf{input@}T)^\mathbb{T}_\mathcal{P} = \mathbf{att}((\mathsf{frame@pred}(T))^\mathbb{T}_\mathcal{P})$.

We finally define the translation $(\phi)^\mathbb{T}_\mathcal{P}$ of a meta-formula. It is obtained by encoding quantifiers as boolean combinations:

$$(\forall i.\phi)^\mathbb{T}_\mathcal{P} \stackrel{def}{=} \dot{\bigwedge}_{k \in D_\mathcal{I}} (\phi)^{\mathbb{T}\{i \mapsto k\}}_\mathcal{P}$$
$$(\forall \tau.\phi)^\mathbb{T}_\mathcal{P} \stackrel{def}{=} \dot{\bigwedge}_{v \in D_\mathcal{T}} (\phi)^{\mathbb{T}\{\tau \mapsto v\}}_\mathcal{P}$$

We proceed similarly for existential quantifiers. Boolean connectives are translated to their dotted counterparts. Atoms over messages $(t = t')^{\mathbb{T}}_{\mathcal{P}}$ are translated into $(t)^{\mathbb{T}}_{\mathcal{P}} \doteq (t')^{\mathbb{T}}_{\mathcal{P}}$. Finally, atoms over indices and timestamps can be fully interpreted in $\mathbb{T}$ and are translated to true or false.

Example 5.7. *Consider a trace model for the Basic Hash protocol $\mathcal{P}$ defined as explained above, and fix $D_{\mathcal{I}} = \{1, 2\}$. In other words, there are only two identities and at most two sessions per identiy. The acceptance condition of the reader is*

$$(\mathsf{cond@a}_R[j,i])^{\mathbb{T}}_{\mathcal{P}} \overset{def}{=} \mathsf{snd}(x) \doteq \mathsf{h}(\mathsf{fst}(x), \mathbf{key}_1) \dot{\vee} \mathsf{snd}(x) \doteq \mathsf{h}(\mathsf{fst}(x), \mathbf{key}_2)$$

where $x = \mathbf{att}((\mathsf{frame@pred}(\mathsf{a}_R[j,i]))^{\mathbb{T}}_{\mathcal{P}})$.

Our translation allows to define a notion of validity for meta-formulas, which intuitively means that the formula is true with overwhelming probability for any trace model and for any computational interpretation.

Definition 5.5. *A meta-formula ϕ is valid wrt. some protocol $\mathcal{P}$ when, for any trace model $\mathbb{T}$, the base logic formula $(\phi)^{\mathbb{T}}_{\mathcal{P}} \sim$ true is valid.*

We similarly define the validity of ϕ wrt. a class of computational models. For instance, the meta-formula of example 5.5 expressing an authentication property for the Basic Hash protocol is valid wrt. the class of models where h satisfies the EUF-CMA assumption. This means that, for any execution trace of the protocol, and for any computational interpretation of the primitives such that h is EUF-CMA, a PPTIME attacker has a negligible probability of fooling a reader along this particular execution trace. This is equivalent to saying that, for any bound b, and for any computational interpretation of the primitives such that h is EUF-CMA, a PPTIME attacker has a negligible probability of fooling a reader when interacting with the protocol for b actions at most. The security guarantees expressed as validity may be called *parametric*: for any fixed bound, the protocol is secure if used within the bound. It is weaker than a true *unbounded* guarantee, where one would want attackers to have a negligible probability of success, whatever the (necessarily polynomial) number of actions that they execute.

5.2.3 Equivalences

Consider two protocols $\mathcal{P}$ and $\mathcal{P}'$ over the same partially-ordered set of actions. In other words, we assume a single set of action symbols $\mathcal{A}$, a single partial order $<$ over them, but two sets of action descriptions $\mathcal{P}_{\mathcal{A}}$ and $\mathcal{P}'_{\mathcal{A}}$. Note that the notion of trace models only depends on $(\mathcal{A}, <)$, so our two protocols have the same trace models. We say that $\mathcal{P}$ and $\mathcal{P}'$ are equivalent when, for any trace model $\mathbb{T}$, the following base logic formula is valid[3]:

$$(\mathsf{frame@}\tau)^{\mathbb{T}}_{\mathcal{P}} \sim (\mathsf{frame@}\tau)^{\mathbb{T}}_{\mathcal{P}'}$$

In other words, we ask that for any trace the attacker cannot distinguish whether it is interacting, along that trace, with $\mathcal{P}$ or $\mathcal{P}'$. This is properly encoded thanks to our detailed notion of frame: our frames inform the attacker about the executability of

[3] As before, this notion of equivalence only brings parametric security guarantees.

the protocol, and provide him with the protocol's outputs in that case. Concretely, the precise definition of our translation for the frame macro is such that following meta-formula is valid:

$$\tau \neq \mathsf{init} \Rightarrow \\ \mathsf{frame@}\tau = \langle \mathsf{exec@}\tau, \langle \mathsf{if\ exec@}\tau\ \mathsf{then\ output@}\tau\ \mathsf{else\ empty}, \mathsf{frame@pred}(\tau)\rangle\rangle$$

It is important to note that our notion of equivalence treats actions as observables. This requires users to pay attention to how processes are written and translated to actions. Indeed, it is often the case that protocol equivalences rely on the inability for the attacker to observe whether some conditional has executed successfully or not; this is not compatible with the transformation of process-level conditionals into two distinct actions, as shown in example 5.4. In such cases, a process-level conditional should be pushed at the level of messages: e.g., **if** ϕ **then out**(c, u) **else out**(c, v) would be changed into **out**$(c, \mathsf{if}\ \phi\ \mathsf{then}\ u\ \mathsf{else}\ v)$. A similar transformation is used when proving the unlinkability of the Basic Hash protocol, to hide from the adversary the index that is found by a reader: thanks to a suitable modification of the process we obtain actions $\mathsf{a}_R[j]$ instead of $\mathsf{a}_R[j, i]$.

5.2.4 Proof systems

We now define some proof systems for deriving valid judgments in our meta-logic.

Reachability properties

For reachability properties, we consider sequents of the form $\Gamma \vdash_{\mathcal{P}} \phi$ where Γ is a multiset of formulas and ϕ is a meta-formula. Such a sequent is said to be valid when the meta-formula $\bigwedge \Gamma \Rightarrow \phi$ is valid with respect to the protocol $\mathcal{P}$.

Conveniently, all rules of the classical first-order sequent calculus are sound for our reachability sequents. We also have rules expressing various sound axioms regarding timestamps and actions, shown in figure 5.2, as well as an induction principle over timestamps. We also show in figure 5.3 some rules which are sound wrt. computational models where $\oplus$ is interpreted as the exclusive or. Finally, some inference rules are derived from the base logic axioms expressing cryptographic assumptions. This requires to verify the side conditions of axioms schemes on meta-terms and formulas, taking into account all possible expansions of our macros during the translation to the base logic. Such a lifting is formally defined in our submitted paper, and we only illustrate it here on simple examples.

The base logic has axioms $\neg(t = \mathbf{n})$ for any ground term t in which $\mathbf{n}$ does not occur. This can be lifted to a sequent calculus rule of the form

$$\frac{\Gamma, \ldots \vdash \phi}{\Gamma, t = \mathbf{n}[\vec{i}] \vdash \phi}$$

where the ellipsis is replaced by a formula expressing that $\mathbf{n}[\vec{i}]$ must occur in t, possibly through macro expansions. In practice, if t contains an input macro and $\mathbf{n}$ is used in the

$$\textsc{NameIndep}\ \frac{\mathsf{n} \neq \mathsf{m}}{\Gamma, \mathsf{n}[\vec{i}] = \mathsf{m}[\vec{j}] \vdash \phi} \qquad \textsc{NameEq}\ \frac{\Gamma, i_1 = j_1, \ldots, i_k = j_k \vdash \phi}{\Gamma, \mathsf{n}[i_1, \ldots, i_k] = \mathsf{n}[j_1, \ldots, j_k] \vdash \phi} \qquad \textsc{ActDep}\ \frac{\mathsf{a}[\vec{j}] < \mathsf{b}[\vec{i}]}{\Gamma, \mathsf{b}[\vec{i}] \leq \mathsf{a}[\vec{j}] \vdash \phi}$$

$$\textsc{ActIndep}\ \frac{\mathsf{a} \neq \mathsf{b}}{\Gamma, \mathsf{a}[\vec{i}] = \mathsf{b}[\vec{j}] \vdash \phi} \qquad \textsc{ActEq}\ \frac{\Gamma, i_1 = j_1, \ldots, i_k = j_k \vdash \phi}{\Gamma, \mathsf{a}[i_1, \ldots, i_k] = \mathsf{b}[j_1, \ldots, j_k] \vdash \phi} \qquad \textsc{Exec}\ \frac{\Gamma, \forall \tau' \leq \tau.\mathsf{cond}@\tau' \vdash \phi}{\Gamma, \mathsf{exec}@\tau \vdash \phi}$$

$$\textsc{Init}\ \frac{}{\Gamma, \tau \neq \mathsf{init} \wedge \tau \leq \mathsf{pred}(\tau) \vdash \phi} \qquad \textsc{Pred}\ \frac{\Gamma, \tau' = \mathsf{pred}(\tau) \vee \tau' = \tau \vdash \phi}{\Gamma, \mathsf{pred}(\tau) \leq \tau' \leq \tau \vdash \phi}$$

Figure 5.2: Some rules of our sequent calculus for reachability.

$$\oplus\text{-}\textsc{Nil}\ \frac{}{\Gamma \vdash t \oplus t = 0} \qquad \oplus\text{-}\textsc{Sym}\ \frac{}{\Gamma \vdash t \oplus t' = t' \oplus t} \qquad \oplus\text{-}\textsc{Assoc}\ \frac{}{\Gamma \vdash t \oplus (t' \oplus t'') = (t \oplus t') \oplus t''}$$

Figure 5.3: Some XOR rules of our sequent calculus for reachability.

output of some action a, our formula will cover the possibility that a has been previously executed.

We proceed similarly for the EUF-CMA axiom. The base logic axiom informally states that, if $u = \mathsf{h}(v, \mathsf{k})$ holds, then it must be that $v = v'$ for some subterm $\mathsf{h}(v', \mathsf{k})$ of u and v. When lifting this axiom to our meta-logic sequent calculus, we obtain a rule which performs a case analysis on all potential subterms $\mathsf{h}(\_, \mathsf{k})$, notably considering past outputs containing subterms of this form.

Equivalence properties

For equivalence judgments, we consider sequents of the form $\vec{u}_1 \sim \vec{v}_1, \ldots, \vec{u}_n \sim \vec{v}_n \vdash_{\mathcal{P},\mathcal{P}'} \vec{u} \sim \vec{v}$. We say that such a sequent is valid when, for any $\mathbb{T}$, the following base logic formula is valid:

$$(\vec{u}_1)^{\mathbb{T}}_{\mathcal{P}} \sim (\vec{v}_1)^{\mathbb{T}}_{\mathcal{P}'} \Rightarrow \ldots \Rightarrow (\vec{u}_n)^{\mathbb{T}}_{\mathcal{P}} \sim (\vec{v}_n)^{\mathbb{T}}_{\mathcal{P}'} \Rightarrow (\vec{u})^{\mathbb{T}}_{\mathcal{P}} \sim (\vec{v})^{\mathbb{T}}_{\mathcal{P}'}$$

We show in figure 5.4 some rules of our sequent calculus for equivalences, which are sound wrt. all computational models. We omit to specify the protocols $\mathcal{P}, \mathcal{P}'$ in sequents when they are the same in all judgments of a rule. Note that the two rules where this is not the case, i.e. Equiv-Term and Equiv-Form, make use of a reachability sequent to justify some rewriting in an equivalence sequent.

In addition to these rules we derive, for each cryptographic assumption, a sequent-calculus rule that is sound wrt. the corresponding class of models. Our full calculus

$$\textsc{Refl}\ \frac{\vec{u}\ \text{is macro-free}}{\Delta \vdash \vec{u} \sim \vec{u}} \qquad \textsc{Enrich}\ \frac{\Delta \vdash \vec{u}, \xi \sim \vec{v}, \xi'}{\Delta \vdash \vec{u} \sim \vec{v}} \qquad \textsc{Dup}\ \frac{\Delta \vdash \vec{u}, \xi \sim \vec{v}, \xi'}{\Delta \vdash \vec{u}, \xi, \xi \sim \vec{v}, \xi', \xi'} \qquad \textsc{Axiom}\ \frac{}{\Delta, \vec{u} \sim \vec{v} \vdash \vec{u} \sim \vec{v}}$$

$$\textsc{FA}\ \frac{\Delta \vdash \vec{u}, t_1, \ldots, t_n \sim \vec{v}, t'_1, \ldots, t'_n}{\Delta \vdash \vec{u}, \mathsf{f}[\vec{i}](t_1, \ldots, t_n) \sim \vec{v}, \mathsf{f}[\vec{i}](t'_1, \ldots, t'_n)} \qquad \textsc{FA-}\Diamond\ \frac{\Delta \vdash \vec{u}, \phi, \phi' \sim \vec{v}, \psi, \psi'}{\Delta \vdash \vec{u}, \phi \Diamond \phi' \sim \vec{v}, \psi \Diamond \psi'}\ \text{where } \Diamond \in \{\wedge, \vee, \Rightarrow\}$$

$$\textsc{Equiv-Term}\ \frac{\vdash_{\mathcal{P}} t = t' \qquad \Delta \vdash_{\mathcal{P},\mathcal{P}'} \vec{u}\{t \mapsto t'\} \sim \vec{v}}{\Delta \vdash_{\mathcal{P},\mathcal{P}'} \vec{u} \sim \vec{v}} \qquad \textsc{Equiv-Form}\ \frac{\vdash_{\mathcal{P}} \phi \Leftrightarrow \phi' \qquad \Delta \vdash_{\mathcal{P},\mathcal{P}'} \vec{u}\{\phi \mapsto \phi'\} \sim \vec{v}}{\Delta \vdash_{\mathcal{P},\mathcal{P}'} \vec{u} \sim \vec{v}}$$

$$\textsc{Induction}\ \frac{\Delta \vdash (\vec{u} \sim \vec{v})\{\tau \mapsto \mathsf{init}\} \quad \left\{\Delta, (\vec{u} \sim \vec{v})\{\tau \mapsto \mathsf{pred}(\mathsf{a}[\vec{i}])\} \vdash (\vec{u} \sim \vec{v})\{\tau \mapsto \mathsf{a}[\vec{i}]\}\right\}_{\mathsf{a} \in \mathcal{A}, \vec{i} \notin \mathsf{fv}(\Delta, \vec{u}, \vec{v})}}{\Delta \vdash \vec{u} \sim \vec{v}}\ \tau \notin \mathsf{fv}(\Delta)$$

Figure 5.4: Generic inference rules for equivalences

comprises rules expressing the PRF assumption for hash functions, the decisional Diffie-Hellman assumption, key privacy, IND-CCA1 as well as a rule expressing the information-hiding feature of the exclusive or.

5.3 Squirrel

We have developed a proof assistant for our meta-logic, called Squirrel. It is a standalone system, implemented in OCaml, and weighs about 12 000 lines of code. It can be used interactively in Emacs through ProofGeneral.

As shown in listing 5.1, Squirrel allows users to specify protocols using a variant of the applied pi-calculus. This specification is then automatically translated to a protocol definition as a partially ordered set of actions. A bi-process can also be used to specify at once two systems over the same set of actions, suitable for equivalence verification.

Cryptographic assumptions are, so far, implicit in symbol declarations. For example, "`hash h`" declares a symbol `h` with index arity 0 and assumes that it satisfies the PRF assumption. Tactics for collision resistance, unforgeability and pseudo-randomness are then made available for this symbol. Additional assumptions may be made by specifying axioms; two examples are given in figure 5.5.

Once the system has been properly defined in this way, Squirrel allows users to establish proofs of reachability and equivalence properties for the declared protocols. In each case, proofs are interactively constructed using tactics and tactic combinators, as is now common in proof assistants. Most elementary tactics slightly elaborate on the rules of our sequents calculi. We have also implemented two elementary tactics that encapsulate basic automated reasoning:

```
abstract tag1 : message
abstract tag2 : message
axiom tags_neq : tag1 ≠ tag2
```

Listing 5.2: Declarations made when verifying protocols where tags are used to distinguish messages.

```
abstract plus : message → message → message
axiom length :
  forall (m1:message, m2:message) len(⟨m1,m2⟩) = plus(len(m1),len(m2))
```

Listing 5.3: Declarations from the proof of anonymity of the Private Authentication protocol, in order to be able to use CCA1 tactic as expected.

Figure 5.5: Example declarations with axioms.

```
goal wa_R :
  forall (j:index),
    cond@R(j) ⇒
    (exists (i,k:index), T(i,k) < R(j) &&
     fst(output@T(i,k)) = fst(input@R(j)) &&
     snd(output@T(i,k)) = snd(input@R(j))).
Proof.
  intros.              (* introduce forall and implication *)
  expand cond@R(j,i).  (* expand cond macro *)
  euf M0.              (* apply EUF to the obtained formula *)
  exists j1.           (* conclude using index introduced by euf *)
Qed.
```

Listing 5.4: Proof of authentication for the Basic Hash protocol.

- The `constraints` tactics implements a complete decision procedure for the satisfiability of a conjunction of atoms over indices and timestamps. It is used to discharge reachability sequents which can be proved sound based on the presence of such atoms.
- The `congruence` tactic deals with message equalities, performing a congruence closure modulo equations expressing the primitives' functionalities, and the equational theory of exclusive or.

We combine basic proof search with these automatic tactics to obtain a goal simplification tactic `simpl`, which is applied by default to all new subgoals. In practice, this is often sufficient to limit user interactions to the essential cases of a proof, as is the case e.g. in the proof shown in listing 5.4.

Proofs of protocol equivalences are generally longer than for reachability properties. They are generally performed by an immediate induction over the considered time. For each action, one then has to shown the indistinguishability between the translations

Protocol	LoC	Assumptions	Security properties
Basic Hash [BCH10]	100	PRF, EUF-CMA	Auth. & Unlink.
Hash Lock [JW09]	130	PRF, EUF-CMA	Auth. & Unlink.
LAK with pairs [HBD19]	250	PRF, EUF-CMA	Auth. & Unlink.
MW [MW04b]	300	PRF, EUF-CMA, XOR	Auth. & Unlink.
Feldhofer [FDW04]	250	CCA_1, EUF-CMA	Auth. & Unlink.
Private Authentication [Aba02]	60	CCA_1, EUF-CMA, ENC-KP	Anonymity
Signed DDH [II19]	150	EUF-CMA, DDH	Auth. & Strong Secrecy
Additional case studies, using the composition framework from [CJS20]			
Signed DDH [II19]	200	EUF-CMA, DDH	Auth. & Strong Secrecy
SSH (with forwarding agent) [YL]	750	EUF-CMA, DDH	Auth. & Strong Secrecy

Table 5.1: Case Studies

of $\mathsf{frame}@\mathsf{a}[\vec{i}]$ corresponding to the two protocols, assuming that the same holds for $\mathsf{pred}(\mathsf{a}[\vec{i}])$. In order to conclude using this induction hypothesis, one has to get rid of the last items of the frame, i.e. $\mathsf{exec}@\mathsf{a}[\vec{i}]$ and $\mathsf{if}\ \mathsf{exec}@\mathsf{a}[\vec{i}]\ \mathsf{then}\ \mathsf{output}@\mathsf{a}[\vec{i}]\ \mathsf{else}\ \mathsf{empty}$. This is done using the various tactics at our disposal. Non-trivial execution conditions are typically handled by changing them into so-called *honest* meta-formulas which intuitively express high-level property of the execution trace, which can be computed by the attacker without referring to protocol-specific information — a typical honest meta-formula is the right hand-side of the implication of the previous authentication property. Once exec has been replaced with the same honest formula on both sides of the equivalence, they can be removed thanks to a special tactic which derives from the base logic axioms FA and DUP. This proof methodology has allowed us to develop proofs of unlinkability for various protocols leveraging well-authentication properties, which are usually easily proved — although this is not an application of the theorems from our earlier work on sufficient conditions for unlinkability, it takes direct inspiration from it.

We show in table 5.1 a summary of our main case studies. We have proved (authentication and) anonymity of the Private Authentication protocol [Aba02], providing a mechanized and parametric version of the proof of Bana and Comon [BC14]. We have also proved (authentication and) strong secrecy for the signed DDH key exchange [II19], again providing a parametric proof of security for unbounded sessions. For this protocol, as well as for the SSH protocol [YL], we have also obtained true unbounded security guarantees through a mechanized proof of a single session using weaker axioms, thanks to a recent composition result [CJS20] that we discuss in more detail below.

We have also established unlinkability for several protocols, using the identity-generic unlinkability notion of definition 3.2 expressed as a diff-equivalence. In addition to providing (parametric) computational guarantees, these case studies bring several improvements over our earlier work on reader-generic unlinkability. We have seen that the diff-equivalence notions of Proverif and Tamarin are too restrictive to be able to conclude in that case. Our notion of equivalence provides the necessary flexibility, by decoupling the handling of the database from the representation of the process execution. We could also prove, with no added difficulty, the MW protocol which relies on exclusive or and could not be handled using Tamarin even with interactive guidance.

5.4 Discussion

We have proposed a new method to obtain parametric proofs of security in the computational model. Starting with the Bana-Comon logic for indistinguishability, we have proposed a meta-logic which allows to describe arbitrary protocol executions, and for which we have derived rich proof systems for reachability and equivalence properties. We have implemented this meta-logic in a new proof assistant, called Squirrel, and demonstrated that proofs in our framework can be concise and natural thanks to basic automated reasoning. We discuss next some related work, before presenting directions for future developments of our project.

5.4.1 Related work in the symbolic model

Before discussing our approach in the context of computational proofs, it is actually interesting to relate it to verification techniques for the symbolic model. These techniques have several advantages. The most obvious ones are a high degree of automation and the ability to discover attacks: indeed, a failure to prove some property in our approach does not immediately yield an attack; it only encourages the user to critically think about the protocol, understand what missing axiom would allow to complete the proof, and perhaps discover an attack on the protocol when this axiom is not satisfied.

Our approach has a number of advantages over verification in symbolic models. First and obviously, we provide stronger guarantees: even though our computational guarantees are only parametric, this is already stronger than unbounded guarantees in the symbolic model. Second, our approach is modular: enriching the model with more primitives and axioms never breaks a proof, unlike in the symbolic model. Third, we inherit a nice feature of the Bana-Comon logic: in order to obtain proofs in this framework one has to clearly state (as axioms) which assumptions are made on cryptographic primitives to achieve the desired security property. It is often the case that some of these assumptions are actually discovered in the process of searching for a proof. In contrast, the symbolic model does not allow to finely express the range of possible cryptographic assumptions, and the standard symbolic models of cryptographic primitives often hides implicit cryptographic assumptions (see, e.g. [Jac+19] on signatures). There are also some less fundamental but practically important differences, for instance in our more flexible notion of diff-equivalence, and the much easier handling of protocols using exclusive or.

Despite all these differences, proofs in Squirrel have interesting similarities with proofs in Tamarin. Both provers proceed by backwards analysis: this is explicit in Tamarin's search for counter-example traces; in our case, the backwards reachability analysis is induced by the use of inference rules such as the one reflecting the EUF-CMA cryptographic assumption. As a result, the two provers do not rely on an enumeration of totally ordered traces. Instead, each subgoal of a proof assumes that some partially ordered subset of actions has been executed. We finally note that deeper similarities are found in some recent work on signatures in Tamarin [Jac+19]. This work explores alternative modelling techniques for signatures, in order to avoid implicit assumptions. Interestingly,

it considers an approach which gives up on the Dolev-Yao style of specifying attacker capabilities, and proposes instead to use axioms (*restrictions* in Tamarin terminology) expressing what the attacker cannot achieve.

5.4.2 Related work in the computational model

Several mature tools already exist for proving protocols in the computational model. The interested reader may find an extensive discussion in [Bar+19]. We focus here on the most relevant comparisons.

Cryptoverif. Cryptoverif [Bla06] mechanizes the cryptographer's standard proof technique of game hopping. It is highly automated, and can discover sequences of transformations from one game to another, but protocol proofs are often obtained by providing the tools with hints on which transformations to apply. Its language is more precise than our meta-logic, notably allowing to distinguish between messages of various sizes, and it supports a very large range of cryptographic assumptions, allowing fine-grained protocol analyses. Moreover, Cryptoverif provides concrete security guarantees, i.e. it derives bounds on the attacker's advantage, whereas our approach only provides asymptotic guarantees.

We have translated some of our case studies to Cryptoverif for a practical comparison of the tools' abilities. Since Cryptoverif supports protocol specifications in the applied pi-calculus, our models could be immediately translated. With the help of Bruno Blanchet, we have then completed the corresponding proofs in Cryptoverif. The resulting scripts are significantly simpler, thanks to the high level of automation of the tool.

A key difference between Cryptoverif and Squirrel is that the former proceeds by transforming games, while the latter proceeds by transforming meta-terms corresponding to a particular execution of the protocol under study. As a result, it would be very difficult to extend Cryptoverif with support for mutable states (used e.g. to model the memory cells of RFID tags) while our approach extends naturally to incorporate this notion — we are already working on this extension, which is partially supported in the current version of Squirrel.

Easycrypt. Easycrypt [Bar+11] is built on a general-purpose probabilistic relational Hoare logic which can be used to formalize most pen-and-paper cryptographic proofs. It makes use of SMT solvers to provide some level of automation. Like Cryptoverif, it allows very precise descriptions of the type of messages, primitives and cryptographic assumptions, and it also provides concrete security guarantees. Being based on a sequential programming language, Easycrypt is best suited for proving properties of primitives (e.g. SHA-3 [Alm+19b]) and APIs (e.g. AWS key management [Alm+19a]). It is also possible to prove protocols in Easycrypt (e.g. the TLS handshake [Bha+14]) but the representation of concurrent processes as (nested) modules can be tedious. As an illustration, we have carried out our Basic Hash unlinkability case study in Easycrypt: both the specification and proof scripts are about ten times longer in the Easycrypt version.

We note that frameworks on top of Easycrypt, such as EasyUC [CSV19], could help to develop proofs of such systems in the future.

CCSA proofs in Coq. In [Ban+20], Bana et al. describe how the Bana-Comon logic and axioms can be encoded in Coq, thus obtaining a minimal proof assistant for the logic — note that they do not define the semantics of the Bana-Comon logic in Coq, nor define the soundness of its proof system. They demonstrate this system on some simple examples, proving indistinguishabilities for fixed numbers of sessions. The authors are currently working on a more general encoding, still using Coq, that will enable parametric proofs.

Their approach is very close in spirit to ours: they use the Coq language as a meta-language over the Bana-Comon logic. Using a mature theorem prover might bring some facilities for notations, automations, and perhaps domain-specific reasoning e.g. over natural numbers. On the other hand, the custom approach that we follow with Squirrel provides more flexibility to define tactics, but also a simpler interface that is specifically tailored for the needs of protocol verification.

5.4.3 Future work

In order to confirm that Squirrel has its place next to other verification systems in the computational model, we will have to consider more complex and varied case studies, and we will have to find a way of providing truly unbounded rather than parametric security guarantees.

Richer and more precise modelling

A natural target in the near future might be to mechanize the parametric proof of unlinkability for the modified AKA protocol of [Kou19b], though it would be more satisfying to obtain first-time proofs for unmodified protocols, e.g. in the domain of e-voting or secure messenging. Such case studies will require to enrich our approach and make our prover more usable, as discussed next.

In order to handle more applications, we will have to enrich our notion of protocol, and the applied pi-calculus variant used as frontend language. We are already working on the addition of states. There is no conceptual difficulty: states can be incorporated into the semantics of protocols, and macros can be added to reason about their values over time. However, proofs quickly become unmanageable without some support to make it easy to reason over states, e.g. to express that only some specific actions can modify some state. Automatically generating and proving dedicated lemmas might provide a solution. Going much further in this direction, one might consider proving implementations of protocols as imperative programs. In a different direction, one could consider the addition of time constraints in protocols, to analyze distance-bounding protocols.

More automation will obviously be needed to tackle these more complex applications. The approach based on our `constraints` and `congruence` tactics has worked well on our first set of case studies. However, these two procedures only communicate in a poor

way, unlike the decision procedures that cooperate with SMT solvers. It might be useful to re-use techniques from the SMT community to obtain a more robust and scalable automation. A more direct solution might simply be to discharge proof obligations in first-order logic to SMT solvers, as is done in Easycrypt — note that this is possible because, although we are not working within classical first-order logic, the theorems of classical first-order logic are also theorems in our reachability meta-logic.

In order to support more applications, we will also have to support more primitives and cryptographic assumptions. We are currently lacking several classic assumptions, including IND-CPA, INT-CTXT, PRP... but applications such as electronic voting would also require support for blind signatures, commitments, and associated cryptographic assumptions. Most of these assumptions have already been translated to axioms in the Bana-Comon logic, but it remains to lift them to our meta-logic. We now have an established framework for doing so, which will certainly ease the handling of many new axioms, but difficult cases might arise. In general, the challenge is to be able to express the lifting of side conditions in our meta-logic; when doing so we allow ourselves to over-approximate the side conditions, but over-approximating too much may result in inapplicable tactics. Simply put, we are facing a general challenge in designing meta-languages: they have to be simple enough to ease specification and reasoning, but rich enough to express all required properties. Time will tell if our language strikes a good balance for an interesting class of applications.

Unbounded verification

Currently, our approach only provides parametric security guarantees. Lifting this limitation is our main theoretical challenge, which may be tackled in two different ways.

First, theoretical results could be devised to derive unbounded guarantees from parametric (or bounded) guarantees. One such result already exists, and has been used in our case studies. In [CJS20], it is shown that the security of one session of a protocol with specifically weakened axioms implies its unbounded security. The main idea behind this technique is to focus on one session and treat the other sessions as an oracle which gives more power to the adversary, hence weakening the axioms expressing cryptographic assumptions. The SSH case study of table 5.1 is actually performed in the framework of that paper, which allows to conclude that this protocol is secure for truly unbounded sessions. More examples of this kind will have to be developed to better understand the generality of this approach. Ultimately, the method should also be mechanized, by synthesizing automatically from a process with unbounded sessions the single-session process and oracles.

Second, it might be possible to analyze proofs in our meta-logic to derive a concrete bound on the attacker's advantage, and conclude that unbounded security holds when this bound does not depend on the length of the trace. In our case studies, most authentication properties are proved by only considering a few actions, so that we can clearly conclude in this way that unbounded security holds. However, proofs by induction over timestamps would not necessarily allow this kind of analysis to succeed. In practice, almost all of our observational equivalence proofs are by induction, and we know that reachability proofs for protocols with state will routinely require inductions. We will thus

need ways to precisely analyze the attacker's advantage in proofs by induction.

The Squirrel prover is, in our opinion, an appealing proposition for a simple approach to protocol verification with strong computational guarantees. Proving this thesis will require to address many challenges, and we welcome the perspective of spending years of research on this project.

Avoir une idée, c'est une espèce de fête.

Gilles Deleuze, 1987

References

[Aba02] Martín Abadi. "Private authentication". In: *International Workshop on Privacy Enhancing Technologies*. Springer. 2002, pp. 27–40.

[AB05] Martín Abadi and Bruno Blanchet. "Analyzing Security Protocols with Secrecy Types and Logic Programs". In: *Journal of the ACM* 52.1 (Jan. 2005), pp. 102–146.

[ABF17] Martín Abadi, Bruno Blanchet, and Cédric Fournet. "The applied pi calculus: Mobile values, new names, and secure communication". In: *Journal of the ACM (JACM)* 65.1 (2017), pp. 1–41.

[AF01] Martín Abadi and Cédric Fournet. "Mobile Values, New Names, and Secure Communication". In: *Proceedings of POPL'01*. ACM Press, 2001.

[AG99] Martín Abadi and Andrew D Gordon. "A calculus for cryptographic protocols: The spi calculus". In: *Information and computation* 148.1 (1999), pp. 1–70.

[AR07] Martín Abadi and Phillip Rogaway. "Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption)". In: *J. Cryptology* 20.3 (2007), p. 395.

[Alm+19a] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, Matthew Campagna, Ernie Cohen, Benjamin Gregoire, Vitor Pereira, Bernardo Portela, Pierre-Yves Strub, and Serdar Tasiran. "A machine-checked proof of security for AWS key management service". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019, pp. 63–78.

[Alm+19b] José Bacelar Almeida, Cécile Baritel-Ruet, Manuel Barbosa, Gilles Barthe, François Dupressoir, Benjamin Grégoire, Vincent Laporte, Tiago Oliveira, Alley Stoughton, and Pierre-Yves Strub. "Machine-checked proofs for cryptographic standards: Indifferentiability of sponge and secure high-assurance implementations of SHA-3". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019, pp. 1607–1622.

[AH13] Gergely Alpár and Jaap-Henk Hoepman. "A secure channel for attribute-based credentials". In: *Proceedings of the 2013 ACM workshop on Digital identity management*. ACM. 2013, pp. 13–18.

[And92] Jean-Marc Andreoli. "Logic programming with focusing proofs in linear logic". In: *Journal of logic and computation* 2.3 (1992), pp. 297–347.

[AP90] Jean-Marc Andreoli and Remo Pareschi. "Linear Objects in a Logic Processes with Built-in Inheritance". In: *Logic Programming, Proceedings of the Seventh International Conference, Jerusalem, Israel, June 18-20, 1990*. Ed. by David H. D. Warren and Péter Szeredi. MIT Press, 1990, pp. 495–510.

[Ara+10] Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark Ryan. "Analysing Unlinkability and Anonymity Using the Applied Pi Calculus". In: *Proceedings of CSF'10*. IEEE Comp. Soc. Press, 2010.

[Arm+05] Alessandro Armando et al. "The AVISPA Tool for the automated validation of internet security protocols and applications". In: *Proc. 17th Int. Conference on Computer Aided Verification (CAV'05)*. LNCS. Springer, 2005.

[Avo05] Gildas Avoine. "Adversarial Model for Radio Frequency Identification." In: *IACR Cryptol. ePrint Arch.* 2005.7 (2005), pp. 49–62.

[BCK20] Kushal Babel, Vincent Cheval, and Steve Kremer. "On the semantics of communications when verifying equivalence properties". In: *Journal of Computer Security* Preprint (2020), pp. 1–57.

[BMU08] Michael Backes, Matteo Maffei, and Dominique Unruh. "Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol". In: *Security and Privacy, 2008. SP 2008. IEEE Symposium on*. IEEE. 2008, pp. 202–215.

[BDH15b] David Baelde, Stéphanie Delaune, and Lucca Hirschi. *Partial Order Reduction for Security Protocols (long version)*. 2015. arXiv: 1504.04768 [cs.CR].

[BK08] C. Baier and J.-P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008. ISBN: 026202649X, 9780262026499.

[BCE18] Gergei Bana, Rohit Chadha, and Ajay Kumar Eeralla. "Formal analysis of vote privacy using computationally complete symbolic attacker". In: *European Symposium on Research in Computer Security*. Springer. 2018, pp. 350–372.

[Ban+20] Gergei Bana, Rohit Chadha, Ajay Kumar Eeralla, and Mitsuhiro Okada. "Verification Methods for the Computationally Complete Symbolic Attacker Based on Indistinguishability". In: *ACM Trans. Comput. Log.* 21.1 (2020), 2:1–2:44.

[BC14] Gergei Bana and Hubert Comon-Lundh. "A Computationally Complete Symbolic Attacker for Equivalence Properties". In: *ACM Conference on Computer and Communications Security*. ACM, 2014, pp. 609–620.

[BC12] Gergei Bana and Hubert Comon-Lundh. "Towards Unconditional Soundness: Computationally Complete Symbolic Attacker". In: *POST*. Vol. 7215. Lecture Notes in Computer Science. Springer, 2012, pp. 189–208.

[Bar+19] Manuel Barbosa, Gilles Barthe, Karthikeyan Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. "SoK: Computer-Aided Cryptography". In: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 1393.

[Bar+11] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. "Computer-Aided Security Proofs for the Working Cryptographer". In: *CRYPTO*. Vol. 6841. Lecture Notes in Computer Science. Springer, 2011, pp. 71–90.

[Bas+12] [SW] David Basin, Cas Cremers, Jannik Dreier, Simon Meier, Ralf Sasse, and Benedikt Schmidt, *Tamarin*, 2012. URL: `https://tamarin-prover.github.io/`.

[BDS15b] David 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*. 2015, pp. 1144–1155.

[BLS20] David A. Basin, Andreas Lochbihler, and S. Reza Sefidgar. "CryptHOL: Game-Based Proofs in Higher-Order Logic". In: *J. Cryptology* 33.2 (2020), pp. 494–566.

[Bha+17] Karthikeyan Bhargavan, Barry Bond, Antoine Delignat-Lavaud, Cédric Fournet, Chris Hawblitzel, Catalin Hritcu, Samin Ishtiaq, Markulf Kohlweiss, Rustan Leino, Jay R. Lorch, Kenji Maillard, Jianyang Pan, Bryan Parno, Jonathan Protzenko, Tahina Ramananandro, Ashay Rane, Aseem Rastogi, Nikhil Swamy, Laure Thompson, Peng Wang, Santiago Zanella Béguelin, and Jean Karim Zinzindohoue. "Everest: Towards a Verified, Drop-in Replacement of HTTPS". In: *SNAPL*. Vol. 71. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 1:1–1:12.

[Bha+14] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Santiago Zanella-Béguelin. "Proving the TLS handshake secure (as it is)". In: *Annual Cryptology Conference*. Springer. 2014, pp. 235–255.

[Bla06] Bruno Blanchet. "A Computationally Sound Mechanized Prover for Security Protocols". In: *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2006, pp. 140–154.

[Bla01] Bruno Blanchet. "An Efficient Cryptographic Protocol Verifier Based on Prolog Rules". In: *CSFW*. IEEE Computer Society, 2001, pp. 82–96.

[Bla04] Bruno Blanchet. "Automatic proof of strong secrecy for security protocols". In: *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*. IEEE. 2004, pp. 86–100.

[Bla09] Bruno Blanchet. "Automatic Verification of Correspondences for Security Protocols". In: *Journal of Computer Security* 17.4 (July 2009), pp. 363–434.

[Bla16] Bruno Blanchet. "Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif". In: *Foundations and Trends in Privacy and Security* 1.1-2 (Oct. 2016), pp. 1–135. DOI: 10.1561/3300000004. URL: https://hal.inria.fr/hal-01423760.

[BAF08] Bruno Blanchet, Martín Abadi, and Cédric Fournet. "Automated verification of selected equivalences for security protocols". In: *The Journal of Logic and Algebraic Programming* 75.1 (2008), pp. 3–51.

[Bla+01] [SW] Bruno Blanchet, Xavier Allamigeon, Vincent Cheval, Ben Smyth, and Marc Sylvestre, *ProVerif*, 2001. URL: https://prosecco.gforge.inria.fr/personal/bblanche/proverif/.

[Boj+09] Mikołaj Bojańczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. "Two-variable logic on data trees and XML reasoning". In: *J. ACM* 56.3, 13 (2009). DOI: 10.1145/1516512.1516515.

[Bor+17] Ravishankar Borgaonkar, Lucca Hirshi, Shinjo Park, Altaf Shaik, Andrew Martin, and Jean-Pierre Seifert. "New Adventures in Spying 3G & 4G Users: Locate, Track, Monitor". In: *Blackhat Las Vegas Conference*. 2017.

[Bre+11] Emmanuel Bresson, Yassine Lakhnech, Laurent Mazaré, and Bogdan Warinschi. "Computational Soundness - The Case of Diffie-Hellman Keys". In: *Formal Models and Techniques for Analyzing Security Protocols*. Ed. by Véronique Cortier and Steve Kremer. Vol. 5. Cryptology and Information Security Series. IOS Press, 2011, pp. 277–302. DOI: 10.3233/978-1-60750-714-7-277. URL: https://doi.org/10.3233/978-1-60750-714-7-277.

[BS11] James Brotherston and Alex Simpson. "Sequent calculi for induction and infinite descent". In: *Journal of Logic and Computation* 21.6 (Dec. 2011), pp. 1177–1216.

[BCH10] Mayla Brusó, K. Chatzikokolakis, and J. den Hartog. "Formal verification of privacy for RFID systems". In: *Proceedings of CSF'10*. 2010.

[Bru+12] Mayla Brusó, Konstantinos Chatzikokolakis, Sandro Etalle, and Jerry Den Hartog. "Linking unlinkability". In: *Trustworthy Global Computing*. Springer, 2012, pp. 129–144.

[CSV19] Ran Canetti, Alley Stoughton, and Mayank Varia. "EasyUC: Using easycrypt to mechanize proofs of universally composable security". In: *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*. IEEE. 2019, pp. 167–16716.

[CL09] Balder ten Cate and Carsten Lutz. "The complexity of query containment in expressive fragments of XPath 2.0". In: *J. ACM* 56.6, 31 (2009). DOI: 10.1145/1568318.1568321.

[Cha+16] Rohit Chadha, Vincent Cheval, Ştefan Ciobâcă, and Steve Kremer. "Automated verification of equivalence properties of cryptographic protocols". In: *ACM Transactions on Computational Logic (TOCL)* 17.4 (2016), pp. 1–32.

[CCK12] Rohit Chadha, Ştefan Ciobâcă, and Steve Kremer. "Automated Verification of Equivalence Properties of Cryptographic Protocols". In: *Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*. Ed. by Helmut Seidl. Vol. 7211. Lecture Notes in Computer Science. Springer, 2012, pp. 108–127. DOI: 10.1007/978-3-642-28869-2_6. URL: https://doi.org/10.1007/978-3-642-28869-2%5C_6.

[CMS08] Kaustuv Chaudhuri, Dale Miller, and Alexis Saurin. "Canonical sequent proofs via multi-focusing". In: *Fifth IFIP International Conference On Theoretical Computer Science–TCS 2008*. Springer. 2008, pp. 383–396.

[CP05] Kaustuv Chaudhuri and Frank Pfenning. "A Focusing Inverse Method Theorem Prover for First-Order Linear Logic". In: *Automated Deduction – CADE-20*. Ed. by Robert Nieuwenhuis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 69–83. ISBN: 978-3-540-31864-4.

[CPP06] Kaustuv Chaudhuri, Frank Pfenning, and Greg Price. "A Logical Characterization of Forward and Backward Chaining in the Inverse Method". In: *Automated Reasoning*. Ed. by Ulrich Furbach and Natarajan Shankar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 97–111. ISBN: 978-3-540-37188-5.

[Che13] [SW] Vincent Cheval, *APTE*, 2013. URL: http://projects.lsv.ens-cachan.fr/APTE/.

[Che14] Vincent Cheval. "APTE: An Algorithm for Proving Trace Equivalence". In: *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*. Ed. by Erika Ábrahám and Klaus Havelund. Vol. 8413. Lecture Notes in Computer Science. Springer, 2014, pp. 587–592.

[CCD11] Vincent Cheval, Hubert Comon-Lundh, and Stéphanie Delaune. "Trace equivalence decision: negative tests and non-determinism". In: *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011*. Ed. by Yan Chen, George Danezis, and Vitaly Shmatikov. ACM, 2011, pp. 321–330. DOI: 10.1145/2046707.2046744. URL: https://doi.org/10.1145/2046707.2046744.

[CCD13] Vincent Cheval, Véronique Cortier, and Stéphanie Delaune. "Deciding equivalence-based properties using constraint solving". In: *Theoretical Computer Science* 492 (2013), pp. 1–39.

[CKR19] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. "Exploiting Symmetries When Proving Equivalence Properties for Security Protocols". In: *CCS'19 - 26th ACM Conference on Computer and Communications Security*. London, United Kingdom, Nov. 2019.

[CKR18] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. "The DEEPSEC Prover". In: *CAV (2)*. Vol. 10982. Lecture Notes in Computer Science. Springer, 2018, pp. 28–36.

[Che+18] [SW] Vincent Cheval, Steve Kremer, Itsaka Rakotonirina, and Victor Yon, *Deepsec*, 2018. URL: https://deepsec-prover.github.io/.

[Cio+12] [SW] Ștefan Ciobâcă, David Baelde, Stéphane Glondu, Steve Kremer, and Ivan Gazeau, *Akiss*, 2012. URL: https://github.com/akiss/akiss.

[CJM00] Edmund Clarke, Somesh Jha, and Will Marrero. "Partial order reductions for security protocol verification". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2000, pp. 503–518.

[CJM03] Edmund M. Clarke, Somesh Jha, and Wilfredo R. Marrero. "Efficient verification of security protocols using partial-order reductions". In: *International Journal on Software Tools for Technology Transfer* 4.2 (2003).

[CJS20] Hubert Comon, Charlie Jacomme, and Guillaume Scerri. "Oracle simulation: a technique for protocol composition with long term shared secrets". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020, pp. 1427–1444.

[CK17] Hubert Comon and Adrien Koutsos. "Formal Computational Unlinkability Proofs of RFID Protocols". In: *CSF*. IEEE Computer Society, 2017, pp. 100–114.

[CC08] Hubert Comon-Lundh and Véronique Cortier. "Computational soundness of observational equivalence". In: *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*. Ed. by Peng Ning, Paul F. Syverson, and Somesh Jha. ACM, 2008, pp. 109–118. DOI: 10.1145/1455770.1455786. URL: https://doi.org/10.1145/1455770.1455786.

[CD05] Hubert Comon-Lundh and Stéphanie Delaune. "The finite variant property: How to get rid of some algebraic properties". In: *International Conference on Rewriting Techniques and Applications*. Springer. 2005, pp. 294–307.

[CDE05] Ricardo Corin, Jeroen Doumen, and Sandro Etalle. "Analysing password protocol security against off-line dictionary attacks". In: *Electronic Notes in Theoretical Computer Science* 121 (2005), pp. 47–63.

[CDD18] Véronique Cortier, Antoine Dallon, and Stéphanie Delaune. "Efficiently deciding equivalence for standard primitives and phases". In: *European Symposium on Research in Computer Security*. Springer. 2018, pp. 491–511.

[Cor+17] Véronique Cortier, Niklas Grimm, Joseph Lallemand, and Matteo Maffei. "A type system for privacy properties". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 409–423.

[Cor+18] Véronique Cortier, Niklas Grimm, Joseph Lallemand, and Matteo Maffei. "Equivalence properties by typing in cryptographic branching protocols". In: *International Conference on Principles of Security and Trust*. Springer, Cham. 2018, pp. 160–187.

[CK14] Véronique Cortier and Steve Kremer. "Formal Models and Techniques for Analyzing Security Protocols: A Tutorial". In: *Foundations and Trends in Programming Languages* 1.3 (Sept. 2014), p. 117. DOI: 10.1561/2500000001. URL: https://hal.inria.fr/hal-01090874.

[CKW11] Véronique Cortier, Steve Kremer, and Bogdan Warinschi. "A Survey of Symbolic Methods in Computational Analysis of Cryptographic Systems". In: *J. Autom. Reasoning* 46.3-4 (2011), pp. 225–259.

[CW11] Véronique Cortier and Bogdan Warinschi. "A composable computational soundness notion". In: *ACM Conference on Computer and Communications Security*. ACM, 2011, pp. 63–74.

[Cre08] Cas JF Cremers. "The Scyther Tool: Verification, falsification, and analysis of security protocols". In: *International conference on computer aided verification*. Springer. 2008, pp. 414–418.

[CM05] Cas JF Cremers and Sjouke Mauw. "Checking secrecy by means of partial order reduction". In: *System Analysis and Modeling*. Springer, 2005.

[Cze+18] Wojciech Czerwiński, Claire David, Filip Murlak, and Paweł Parys. "Reasoning about integrity constraints for tree-structured data". In: *Theor. Comput. Syst.* 62.4 (2018), pp. 941–976. DOI: 10.1007/s00224-017-9771-z.

[DCD17] [SW] Antoine Dallon, Véronique Cortier, and Stéphane Delaune, *SAT-Equiv*, 2017. URL: https://projects.lsv.fr/satequiv/.

[DJS95] Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. "LKQ and LKT: sequent calculi for second order logic based upon dual linear decompositions of classical implication". In: *Advances in Linear Logic* 222 (1995), pp. 211–224.

[DR08] Ton van Deursen and Sasa Radomirovic. "Attacks on RFID Protocols". In: *IACR Cryptology ePrint Archive* 2008 (2008), p. 310.

[DY81] Danny Dolev and Andrew Chi-Chih Yao. "On the Security of Public Key Protocols (Extended Abstract)". In: *FOCS*. IEEE Computer Society, 1981, pp. 350–357.

[Dou17] Amina Doumane. "Constructive completeness for the linear-time μ-calculus". In: *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. IEEE Computer Society, 2017, pp. 1–12.

[FDW04] Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. "Strong Authentication for RFID Systems Using the AES Algorithm". In: *CHES*. Vol. 3156. Lecture Notes in Computer Science. Springer, 2004, pp. 357–370.

[FS17] Diego Figueira and Luc Segoufin. "Bottom-up automata on data trees and vertical XPath". In: *Logical Methods in Computer Science* 13.4, 5 (2017). DOI: 10.23638/LMCS-13(4:5)2017.

[FS09] Diego Figueira and Luc Segoufin. "Future-looking logics on data words and trees". In: *MFCS '09*. Vol. 5734. LNCS. Novy Smokovec, High Tatras, Slovakia: Springer, 2009, pp. 331–343. DOI: 10.1007/978-3-642-03816-7_29.

[Fil+19] Ihor Filimonov, Ross Horne, Sjouke Mauw, and Zach Smith. "Breaking unlinkability of the icao 9303 standard for e-passports using bisimilarity". In: *European Symposium on Research in Computer Security*. Springer. 2019, pp. 577–594.

[FG05] Cormac Flanagan and Patrice Godefroid. "Dynamic partial-order reduction for model checking software". In: *ACM Sigplan Notices*. Vol. 40. 1. ACM. 2005, pp. 110–121.

[FDW10] Wan Fokkink, Mohammad Torabi Dashti, and Anton Wijs. "Partial order reduction for branching security protocols". In: *Proceedings of ACSD'10*. IEEE. 2010.

[FS13] Jérôme Fortier and Luigi Santocanale. "Cuts for circular proofs: semantics and cut-elimination". In: *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*. Ed. by Simona Ronchi Della Rocca. Vol. 23. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013, pp. 248–262. ISBN: 978-3-939897-60-6.

[GK17] Ivan Gazeau and Steve Kremer. "Automated Analysis of Equivalence Properties for Security Protocols Using Else Branches". In: *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II*. Ed. by Simon N. Foley, Dieter Gollmann, and Einar Snekkenes. Vol. 10493. Lecture Notes in Computer Science. Springer, 2017, pp. 1–20.

[GF05] Floris Geerts and Wenfei Fan. "Satisfiability of XPath queries with sibling axes". In: *DBPL '05*. Trondheim, Norway: Springer, 2005, pp. 122–137. DOI: 10.1007/11601524_8.

[Gir01] Jean-Yves Girard. "Locus Solum: From the rules of logic to the logic of rules". In: *Math. Struct. Comput. Sci.* 11.3 (2001), pp. 301–506. DOI: 10.1017/S096012950100336X. URL: https://doi.org/10.1017/S096012950100336X.

[God95] Patrice Godefroid. "Partial-Order Methods for the Verification of Concurrent Systems". PhD thesis. Université de Liège, 1995.

[God96] Patrice Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems - An Approach to the State-Explosion Problem*. Vol. 1032. Lecture Notes in Computer Science. Springer, 1996. ISBN: 3-540-60761-7. DOI: 10.1007/3-540-60761-7. URL: http://dx.doi.org/10.1007/3-540-60761-7.

[God90] Patrice Godefroid. "Using partial orders to improve automatic verification methods". In: *International Conference on Computer Aided Verification*. Springer. 1990, pp. 176–185.

[GK02] Georg Gottlob and Christoph Koch. "Monadic queries over tree-structured data". In: *LICS '02*. IEEE, 2002, pp. 189–202. DOI: 10.1109/LICS.2002.1029828.

[Her95] Hugo Herbelin. "Séquents qu'on calcule: de l'interprétation du calcul des séquents comme calcul de lambda-termes et comme calcul de stratégies gagnantes". PhD thesis. 1995.

[Hid04] Jan Hidders. "Satisfiability of XPath expressions". In: *DBPL 2003*. Ed. by Georg Lausen and Dan Suciu. Vol. 2921. LNCS. Potsdam, Germany: Springer, 2004, pp. 21–36. DOI: 10.1007/978-3-540-24607-7_3.

[Hir17] Lucca Hirschi. "Automated Verification of Privacy in Security Protocols : Back and Forth Between Theory & Practice". Theses. Université Paris-Saclay, Apr. 2017. URL: https://tel.archives-ouvertes.fr/tel-01534145.

[Hir16] [SW] Lucca Hirschi, *UKAno*, 2016. URL: http://projects.lsv.fr/ukano/.

[HC19] Lucca Hirschi and Cas Cremers. "Improving automated symbolic analysis of ballot secrecy for e-voting protocols: A method based on sufficient conditions". In: *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2019, pp. 635–650.

[HS96] Hans Hüttel and Sandeep Shukla. "On the Complexity of Deciding Behavioural Equivalences and Preorders. A Survey". In: *BRICS Report Series* 39 (1996).

[09] *ISO 15408-2: Common Criteria for Information Technology Security Evaluation - Part 2: Security functional components*. ISO, July 2009.

[II19] International Organization for Standardization and International Electrotechnical Commission. *ISO/IEC 9798, IT Security techniques – Entity authentication – Part 3: Mechanisms using digital signature techniques*. Standard. 2019.

[Jac+19] 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*. 2019, pp. 2165–2180.

[JNS05] Radha Jagadeesan, Gopalan Nadathur, and Vijay Saraswat. "Testing concurrent systems: An interpretation of intuitionistic logic". In: *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer. 2005, pp. 517–528.

[JW09] Ari Juels and Stephen A Weis. "Defining strong privacy for RFID". In: *ACM Transactions on Information and System Security (TISSEC)* 13.1 (2009), p. 7.

[JL11] M. Jurdziński and R. Lazić. "Alternating automata on data trees and XPath satisfiability". In: *ACM Trans. on Computational Logic* 12.3, 19 (2011). DOI: 10.1145/1929954.1929956.

[Kou19a] Adrien Koutsos. "Decidability of a Sound Set of Inference Rules for Computational Indistinguishability". In: *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*. IEEE, 2019, pp. 48–61. DOI: 10.1109/CSF.2019.00011. URL: https://doi.org/10.1109/CSF.2019.00011.

[Kou19b] Adrien Koutsos. "The 5G-AKA authentication protocol privacy". In: *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2019, pp. 464–479.

[KK16] Steve Kremer and Robert Künnemann. "Automated analysis of security protocols with global state". In: *Journal of Computer Security* 24.5 (2016), pp. 583–616.

[Lau02] Olivier Laurent. "Etude de la polarisation en logique". PhD thesis. Université de la Méditerranée-Aix-Marseille II, 2002.

[LAK+06] Sangshin Lee, Tomoyuki Asano, Kwangjo Kim, et al. "RFID mutual authentication scheme based on synchronized secret information". In: *Symposium on cryptography and information security*. 2006, pp. 17–20.

[LM09] Chuck Liang and Dale Miller. "Focusing and polarization in linear, intuitionistic, and classical logics". In: *Theoretical Computer Science* 410.46 (2009). Abstract Interpretation and Logic Programming: In honor of professor Giorgio Levi, pp. 4747–4768. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2009.07.041. URL: http://www.sciencedirect.com/science/article/pii/S0304397509005301.

[Lic19] Anthony Lick. "A Hypersequent Calculus with Clusters for Data Logic over Ordinals". In: *Automated Reasoning with Analytic Tableaux and Related Methods - 28th International Conference, TABLEAUX 2019, London, UK, September 3-5, 2019, Proceedings*. Ed. by Serenella Cerrito and Andrei Popescu. Vol. 11714. Lecture Notes in Computer Science. Springer, 2019, pp. 166–184.

[Low96] Gavin Lowe. "Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR". In: *Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS '96, Passau, Germany, March 27-29, 1996, Proceedings*. Ed. by Tiziana Margaria and Bernhard Steffen. Vol. 1055. Lecture Notes in Computer Science. Springer, 1996, pp. 147–166. DOI: 10.1007/3-540-61042-1_43. URL: https://doi.org/10.1007/3-540-61042-1%5C_43.

[Mei+13] Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. "The TAMARIN Prover for the Symbolic Analysis of Security Protocols". In: *CAV*. Vol. 8044. Lecture Notes in Computer Science. Springer, 2013, pp. 696–701.

[MW04a] Daniele Micciancio and Bogdan Warinschi. "Soundness of formal encryption in the presence of active adversaries". In: *Theory of Cryptography Conference*. Springer. 2004, pp. 133–151.

[Mil03] Dale Miller. "Encryption as an abstract data-type: An extended abstract". In: *Electronic Notes in Theoretical Computer Science* 84 (2003). WoLLIC'2003, 10th Workshop on Logic, Language, Information and Computation, pp. 18–29. ISSN: 1571-0661. DOI: https://doi.org/10.1016/S1571-0661(04)80841-7. URL: http://www.sciencedirect.com/science/article/pii/S1571066104808417.

[Mil93] Dale Miller. "The π-calculus as a theory in linear logic: Preliminary results". In: *Extensions of Logic Programming*. Ed. by E. Lamma and P. Mello. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 242–264. ISBN: 978-3-540-47562-0.

[Mil+91] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. "Uniform proofs as a foundation for logic programming". In: *Annals of Pure and Applied logic* 51.1-2 (1991), pp. 125–157.

[MS07] Dale Miller and Alexis Saurin. "From proofs to focused proofs: a modular proof of focalization in linear logic". In: *International Workshop on Computer Science Logic*. Springer. 2007, pp. 405–419.

[Mil99] Robin Milner. *Communicating and mobile systems: the pi calculus*. Cambridge university press, 1999.

[MVB10] S. Mödersheim, L. Viganò, and D. Basin. "Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols". In: *JCS* 18.4 (2010).

[MW04b] David Molnar and David Wagner. "Privacy and security in library RFID: Issues, practices, and architectures". In: *Proceedings of the 11th ACM conference on Computer and communications security*. 2004, pp. 210–219.

[OSK+03] Miyako Ohkubo, Koutarou Suzuki, Shingo Kinoshita, et al. "Cryptographic approach to "privacy-friendly" tags". In: *RFID privacy workshop*. Vol. 82. Cambridge, USA. 2003.

[Pel98] D. Peled. "Ten Years of Partial Order Reduction". In: *Proc. of CAV'98*. Springer, 1998.

[Pel93] Doron Peled. "All from one, one for all: on model checking using representatives". In: *International Conference on Computer Aided Verification*. Springer. 1993, pp. 409–423.

[San02] Luigi Santocanale. "A Calculus of Circular Proofs and Its Categorical Semantics". In: *Foundations of Software Science and Computation Structures*. Ed. by Mogens Nielsen and Uffe Engberg. Vol. 2303. Lecture Notes in Computer Science. Springer, 2002, pp. 357–371. ISBN: 3-540-43366-X.

[SRC15] Ben Smyth, Mark D Ryan, and Liqun Chen. "Formal analysis of privacy in Direct Anonymous Attestation schemes". In: *Science of Computer Programming* 111 (2015), pp. 300–317.

[TD10] Alwen Tiu and Jeremy E. Dawson. "Automating Open Bisimulation Checking for the Spi Calculus". In: *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010*. IEEE Computer Society, 2010, pp. 307–321. DOI: 10.1109/CSF.2010.28. URL: https://doi.org/10.1109/CSF.2010.28.

[TNH10] [SW] Alwen Tiu, Nguyen Thanh Nam, and Ross Horne, *SPEC*, 2010. URL: http://users.cecs.anu.edu.au/~tiu/spec-prover/.

[TNH16] Alwen Tiu, Nam Nguyen, and Ross Horne. "SPEC: An Equivalence Checker for Security Protocols". In: *Programming Languages and Systems - 14th Asian Symposium, APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings*. Ed. by Atsushi Igarashi. Vol. 10017. Lecture Notes in Computer Science. 2016, pp. 87–95.

[Unr10] Dominique Unruh. "The impossibility of computationally sound XOR." In: *IACR Cryptol. ePrint Arch.* 2010 (2010), p. 389.

[Wei+04] Stephen A Weis, Sanjay E Sarma, Ronald L Rivest, and Daniel W Engels. "Security and privacy aspects of low-cost radio frequency identification systems". In: *Security in pervasive computing*. Springer, 2004, pp. 201–212.

[YL] Tatu Ylonen and Chris Lonvick. *The Secure Shell (SSH) Transport Layer Protocol*. en. URL: https://tools.ietf.org/html/rfc4253 (visited on 05/17/2019).

Publications

Below is a complete list of my publications, sorted by year.

Edited volumes

[Bae+20] David Baelde, Amy P. Felty, Gopalan Nadathur, and Alexis Saurin. "A special issue on structural proof theory, automated reasoning and computation in celebration of Dale Miller's 60th birthday". In: *Mathematical Structures in Computer Science* 29.8 (2020), pp. 1007–1008.

[BA15] David Baelde and Jade Alglave, eds. *Actes des Vingt-Sixi'emes Journ'ees Francophones des Langages Applicatifs (JFLA)*. Collection HAL. 2015.

[BC13] David Baelde and Arnaud Carayol, eds. *Proceedings of the Workshop on Fixed Points in Computer Science, FICS 2013*. EPTCS. 2013.

Journals

[HBD19] Lucca Hirschi, David Baelde, and Stéphanie Delaune. "A method for unbounded verification of privacy-type properties". In: *Journal of Computer Security* 27.3 (2019), pp. 277–342.

[BDH17] David Baelde, Stéphanie Delaune, and Lucca Hirschi. "A Reduced Semantics for Deciding Trace Equivalence". In: *Logical Methods in Computer Science* 13.2:8 (2017), pp. 1–48.

[Bae+14] David Baelde, Kaustuv Chaudhuri, Andrew Gacek, Dale Miller, Gopalan Nadathur, Alwen Tiu, and Yuting Wang. "Abella: A System for Reasoning about Relational Specifications". In: *Journal of Formalized Reasoning* (2014).

[Bae12] David Baelde. "Least and greatest fixed points in linear logic". In: *ACM Trans. on Computational Logic* (2012).

Conferences

[Bae+22a] David Baelde, Stéphanie Delaune, Adrien Koutsos, and Solène Moreau. "Cracking the Stateful Nut: Computational Proofs of Stateful Security Protocols using the Squirrel Proof Assistant". In: *Proceedings of the 35th IEEE Computer Security Foundations Symposium (CSF'22)*. To appear. IEEE Computer Society Press, 2022.

[Bae+22b] David Baelde, Amina Doumane, Denis Kuperberg, and Alexis Saurin. "Bouncing threads for circular and non-wellfounded proofs". In: *Proceedings on the 37th Symposium on Logic in Computer Science (LICS'22)"*. To appear. IEEE Computer Society Press, 2022.

[Bae+21] David Baelde, Stéphanie Delaune, Charlie Jacomme, Adrien Koutsos, and Solène Moreau. "An Interactive Prover for Protocol Verification in the Computational Model". In: *Proceedings of IEEE Symposium on Security and Privacy (S&P'21)*. IEEE Computer Society Press, 2021.

[BDM20] David Baelde, Stéphanie Delaune, and Solène Moreau. "A Method for Proving Unlinkability of Stateful Protocols". In: *Proceedings of the 33rd IEEE Computer Security Foundations Symposium (CSF'20)*. Boston, MA, USA: IEEE Computer Society Press, July 2020.

[BLS19] David Baelde, Anthony Lick, and Sylvain Schmitz. "Decidable XPath Fragments in the Real World". In: *Proceedings of the 38th Annual ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'19)*. Amsterdam, Netherlands: ACM Press, 2019, pp. 285–302.

[BDH18] David Baelde, Stéphanie Delaune, and Lucca Hirschi. "POR for Security Protocol Equivalences - Beyond Action-Determinism". In: *Proceedings of the 23rd European Symposium on Research in Computer Security (ESORICS'18)*. Vol. 11098. Lecture Notes in Computer Science. Barcelona, Spain: Springer, Sept. 2018, pp. 385–405.

[BLS18a] David Baelde, Anthony Lick, and Sylvain Schmitz. "A Hypersequent Calculus with Clusters for Linear Frames". In: *Proceedings of the 10th Conference on Advances in Modal Logics (AiML'18)*. Bern, Switzerland: College Publications, Aug. 2018, pp. 36–55.

[BLS18b] David Baelde, Anthony Lick, and Sylvain Schmitz. "A Hypersequent Calculus with Clusters for Tense Logic over Ordinals". In: *Proceedings of the 38th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'18)*. Vol. 122. Leibniz International Proceedings in Informatics. Ahmedabad, India: Leibniz-Zentrum für Informatik, Dec. 2018, 15:1–15:19.

[Bae+17] David Baelde, Stéphanie Delaune, Ivan Gazeau, and Steve Kremer. "Symbolic Verification of Privacy-Type Properties for Security Protocols with XOR". In: *Proceedings of the 30th IEEE Computer Security Foundations Symposium (CSF'17)*. Santa Barbara, California, USA: IEEE Computer Society Press, Aug. 2017, pp. 234–248.

[BDS16] David Baelde, Amina Doumane, and Alexis Saurin. "On the proof theory of infinitary proofs". In: *Proceedings of the 25th EACSL Annual Conference on Computer Science Logic (CSL'16)*. LIPIcs. 2016.

[BLS16] David Baelde, Simon Lunel, and Sylvain Schmitz. "A sequent calculus for a modal logic on finite data trees". In: *Proceedings of the 25th EACSL Annual Conference on Computer Science Logic (CSL'16)*. LIPIcs. 2016.

[Dou+16] Amina Doumane, David Baelde, Lucca Hirschi, and Alexis Saurin. "Towards completeness via proof search in the linear time mu-calculus : the case of B"uchi inclusions". In: *Proceedings of the 31st Annual Symposium on Logic in Computer Science (LICS'16)*. ACM Press. 2016.

[HBD16] Lucca Hirschi, David Baelde, and Stéphanie Delaune. "A method for verifying privacy-type properties: the unbounded case". In: *Proceedings of the 37th IEEE Symposium on Security and Privacy (S&P'16)*. San Jose, California, USA: IEEE Computer Society Press, May 2016, pp. 564–581.

[BDH15a] David Baelde, Stéphanie Delaune, and Lucca Hirschi. "Partial Order Reduction for Security Protocols". In: *Proceedings of the 26th International Conference on Concurrency Theory (CONCUR'15)*. Vol. 42. Leibniz International Proceedings in Informatics. Madrid, Spain: Leibniz-Zentrum für Informatik, Sept. 2015, pp. 497–510.

[BDS15a] David Baelde, Amina Doumane, and Alexis Saurin. "Least and Greatest Fixed Points in Ludics". In: *Proceedings of the 24th EACSL Annual Conference on Computer Science Logic (CSL'15)*. LIPcs. 2015.

[BDH14] David Baelde, St'ephanie Delaune, and Lucca Hirschi. "A reduced semantics for deciding trace equivalence using constraint systems". In: *Proceedings of the International Conference on the Principles of Security and Trust (POST'14)*. Springer, 2014.

[Bae+12] David Baelde, Pierre Courtieu, David Gross-Amblard, and Christine Paulin-Mohring. "Towards Provably Robust Watermarking". In: *Proceedings of the International Conference on Interactive Theorem Proving (ITP'12)*. Vol. 7406. LNCS. Springer, 2012.

[BN12] David Baelde and Gopalan Nadathur. "Combining Deduction Modulo and Logics of Fixed-Point Definitions". In: *Proceedings on the 27th Symposium on Logic in Computer Science (LICS'12)"*. IEEE Computer Society Press, 2012.

[BBM11] David Baelde, Romain Beauxis, and Samuel Mimram. "Liquidsoap: a High-Level Programming Language for Multimedia Streaming". In: *Proceedings of the 37th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'11)*. Vol. 6543. LNCS. Springer, 2011.

[BMS10] David Baelde, Dale Miller, and Zachary Snow. "Focused Inductive Theorem Proving". In: *Fifth International Joint Conference on Automated Reasoning (IJCAR'10)*. LNCS. 2010.

[SBN10] Zachary Snow, David Baelde, and Gopalan Nadathur. "A Meta-Programming Approach to Realizing Dependently Typed Logic Programming". In: *Proceedings of the 12th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP'10)*. ACM Press, 2010.

[Bae09] David Baelde. "On the proof theory of regular fixed points". In: *TABLEAUX'09: Automated Reasoning with Analytic Tableaux and Related Methods*. LNAI. 2009.

[Bae08] David Baelde. "On the Expressivity of Minimal Generic Quantification". In: *International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP'08)*. Elec. Notes in Theor. Comput. Sci. 2008.

[Bae+07] David Baelde, Andrew Gacek, Dale Miller, Gopalan Nadathur, and Alwen Tiu. "The Bedwyr system for model checking over syntactic expressions". In: *Proceedings of the 21st International Conference on Automated Deduction (CADE'07)*. LNAI. Springer, 2007.

[BM07] David Baelde and Dale Miller. "Least and greatest fixed points in linear logic". In: *International Conference on Logic for Programming and Automated Reasoning (LPAR'07)*. LNCS. 2007.

National conferences

[BM08] David Baelde and Samuel Mimram. "De la webradio lambda à la λ-webradio". In: *Journées Francophones des Langages Applicatifs*. INRIA, 2008.