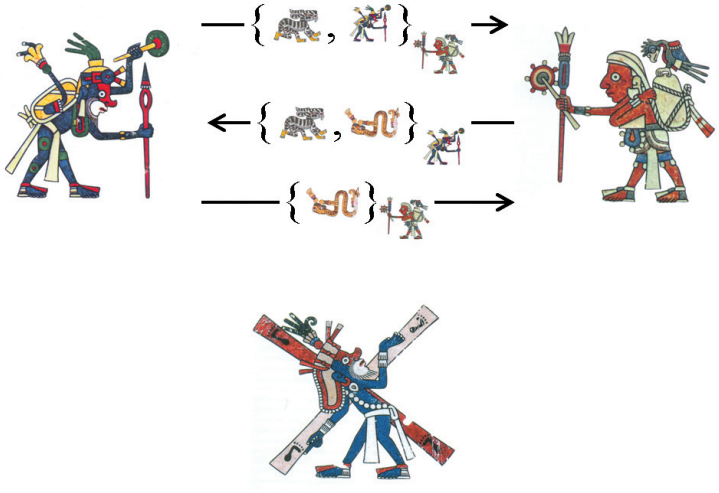
Analysis Models for Security Protocols



Ricardo Corin

Analysis Models for Security Protocols

Ricardo Corin

Graduation committee:

Prof. Dr. P. H. Hartel (promotor)	University of Twente, The Netherlands
Dr. S. Etalle (assistant promotor)	University of Twente, The Netherlands
Prof. Dr. H. Brinksma	University of Twente, The Netherlands
Dr. C. Fournet	Microsoft Research Cambridge, United Kingdom
Prof. Dr. B. P. F. Jacobs	Radboud University Nijmegen, The Netherlands
Prof. Dr. Y. Lakhnech	Verimag, University Joseph Fourier, France
Dr. F. Martinelli	IIT, National Research Council - C.N.R., Italy

Cover: Needham Schroeder protocol [154] with principals and attacker illustrated by Aztec merchants, *pochtecs*. Back cover: Lowe's attack [131] over the Needham Schroeder protocol.

The work in this thesis has been carried out under the auspices of the Institute for Programming Research and Algorithmics (IPA) research school and within the context of the Centre for Telematics and Information Technology (CTIT).

IPA Dissertation Series No. 2006–02
Series title: CTIT Ph.D.-thesis Series
Series number: 1381-3617
CTIT number: 05-76

Typeset by LaTeX, edited with Emacs, and printed by Febodruk b.v.
Keywords: security protocol, analysis model, formal methods.

ISBN: 90-365-2279-X

ANALYSIS MODELS FOR SECURITY PROTOCOLS

DISSERTATION

to obtain
the doctor's degree at the University of Twente,
on the authority of the rector magnificus,
prof. dr. W.H.M. Zijm,
on account of the decision of the graduation committee,
to be publicly defended
on Thursday, January 12, 2006 at 13:15

by

Ricardo Javier Corin

born on 14 August 1977

in Córdoba, Argentina

This dissertation is approved by:

Prof. Dr. P. H. Hartel (promotor)
Dr. S. Etalle (assistant promotor)

A la memoria de mi abuelo León Jaimovich.

Summary

The communication via a shared medium, like the Internet, is inherently insecure: anyone may access en route messages and could potentially eavesdrop or even manipulate the ongoing communication. Security protocols are distributed programs specifically designed to achieve secure communication over such media, typically exchanging messages built using cryptographic operations (e.g. message encryption).

Security protocols are difficult to design correctly, hence their analysis is critical. A particularly successful model to analyze security protocols is the Dolev Yao model, in which the attacker is assumed to have complete control over the network. Also, the model assumes ideal cryptography, where cryptographic operations are assumed to be perfect. The Dolev Yao model is attractive because it can be easily formalized using languages and tools based on formal methods. Moreover, the model has an appropriate level of abstraction, as many attacks are independent of the underlying details of the cryptographic operations and are based only on combinations of message exchanges plus knowledge gathered by the attacker during the execution.

In this thesis, we present five significant, orthogonal extensions to the Dolev Yao model. Each extension considers a more realistic setting, closer to the real world, thus providing a stronger security guarantee. We provide examples both from the literature and from industrial case studies to show the practical applicability of each extension. The extensions are:

1. We propose an efficient constraint solving procedure, which improves the work of Millen and Shmatikov. We then introduce a property language based on linear temporal logic and also present a procedure to search guessing attacks. We use the procedures both as a teaching tool and to analyze case studies.

2. We develop a model which considers explicitly the passage of time during the execution of protocols, using timed automata. This allows the study of timing issues like timeouts and retransmissions in security protocol implementations.

3. We use the Applied Pi Calculus to study guessing attacks under an extended attacker that can exploit cryptographic relations in eavesdropped messages.

4. We use the language TulaFale to study session based web service protocols, by formalizing and providing semantics to two industrial specifications. We build a realistic model to analyze protocols relying on these specifications.

5. We relate a Dolev Yao model with a realistic computational model. In particular, we extend the original work of Abadi and Rogaway by considering encryption with composed keys as opposed to atomic keys.

Acknowledgements

I am grateful to many people for their support and, more importantly, for letting me have lots of fun with them during these years. Besides the large amounts of caffeine concentrated in uncountable espressos, having fun was an indispensable fuel to make this document a reality!

I thank my promotor Pieter Hartel, who not only taught me how to write papers but also showed me the joy of doing research. I also wish to thank my daily advisor Sandro Etalle, who was the first person (including myself) that believed I could do a PhD. Moreover, Sandro spent much time teaching me how to fully develop, organize and present my ideas.

Both Pieter and Sandro had the infinite patience necessary to deal with my chaotic lifestyle. Besides them, I am grateful to the group's secretaries Marlous, Nicol and Paula for suffering my (non-existent) organizational skills without complaining (at least not too much!).

I thank Ed Brinksma, Cédric Fournet, Bart Jacobs, Yassine Lakhnech and Fabio Martinelli for assessing my manuscript as members of my graduation committee. Besides them, I received useful comments to early drafts of this thesis from Laura, Jerry den Hartog (who also kindly translated the Summary to Dutch), Angelika Mader and Ari Saptawijaya.

I had the pleasure of being an intern at Microsoft Research in Cambridge during the summer of 2004, working within the Samoa project with Cédric Fournet, Karthik Bhargavan and Andy Gordon. I thank them also for inviting me (twice) to visit MSR afterwards, with the excuse of finishing some work (and, in the process, of course to have some more fun).

I also wish to thank my undergraduate professor Diego Vaggione, whom exemplary behaviour put me in the right track for mature research. Even if at that moment I could not understand it, now I do and can say *gracias!*

Thanks to Jordan, Law, Nikolay, Stefan, Supriyo, Tim and Vasughi for being such nice fellow PhDs. Besides them, I had enjoyable time with many people, far more than I could thank properly here; I mention thus only some: Antonio, Ari, Bill, Carlos, Conrado, Diego, Flavio, Gwenael, Jan, Jerry, Jeroen, José, Marcos and family, Mohammad, Pedro, Philippe and Sergio. I also thank the new PhD batch for the nice parties and sportive gatherings. Finally, thanks to Cas Cremers and Martijn Warnier for the pleasant SPAN meetings.

Although I could seldom meet my Argentinian friends, they still remained close to me. They are Adrián, Alfredo, Diego, Matías, Nico S. and K., Lucas, Tamara, Tino and Valeria.

Moreover, I am lucky to have been able to make new friends along the way. Sincere thanks to Gabriele, Karthik and Marnix for all the great time we spent together.

Muchísimas gracias a mi familia, que siempre estuvo y estarán conmigo a pesar de la distancia: Mami, Papi, Gaby, Diego, Vale, y por supuesto a todos mis otros familiares, tíos, primos, cuñados, suegros, sobrinos, y en especial a mis abuelas Juana y Frida (cuyos abrigos fueron indispensables para pasar el invierno!).

I thank finally my wife Laura, who unconditionally shines every day of my life.

Enschede, December 2005.

Table of Contents

CHAPTER 1

Introduction

An *analysis model* is a description specifically created to examine and evaluate an object. Such description is realized using a formal language and it reflects properties or behaviours of the original object, while it abstracts from other aspects. In general, the objects of interest may be anything, like complex physical phenomena or computer systems composed of parallel processes executing together.

In this thesis, the objects we consider are *security protocols*, also known as cryptographic protocols. Security protocol *principals*, such as human beings or computers, execute a security protocol by exchanging messages over a medium to achieve some particular *goal*. We assume that such a medium (the *network*) is not private to the principals, but shared between all other participants. This implies that other participants not taking part in the execution may still see messages passing by, and potentially play an active role in the communication. Moreover, participants do not necessarily trust each other, and thus principals executing the security protocol can not rely on other participants to simply avert their eyes and behave honestly. A prominent example of such a medium is the Internet.

The correct design of security protocols is a difficult task. Furthermore, ensuring that the execution of a security protocol indeed achieves the intended goals is critical to provide confidence on the security of the communication. To establish the correctness of a security protocol, we first need to define a model in which such protocol is going to be analyzed. Our models of security protocols follow the structure depicted in Figure 1.1. An *analysis model* $\mathcal{M}$ (or just model for short) consists of three submodels: a *property model*, an *attacker model*, and an *environment model*. The environment model encloses the attacker model, while the property model is separate. In the following, we elaborate on each of these models.

The property model allows the formalization of the *goals* of the protocol, that is the security guarantees it is supposed to provide. The security goals are also known as the protocol *requirements* or the *security properties*.

The attacker model describes a participant, called the attacker (or *intruder*) which does not necessarily follow the rules of the protocol. Actually, its main interest is in *breaking* the protocol, by subverting the intended goal (specified using the property model described above). In the attacker model, we detail which *abilities* are available to the attacker, that is, which operations the attacker is able to perform when trying to accomplish its goal. The attacker model is also sometimes called the *threat model*.

The environment model is a representation of all the surrounding world of the attacker (described above in the attacker model). The environment model includes *honest* principals which faithfully follow the steps prescribed by the security protocol. By modelling these principals, the environment model also encodes the security protocol under consideration. Furthermore, the environment model describes the communication mechanisms available between participants. Alternatively, the environment model may also describe any quality of interest from the real world which may influence the behaviour (or security assurances) of the protocol. Examples include modelling explicitly the passage of time, or modelling intrinsic network characteristics such as noise or routing details.

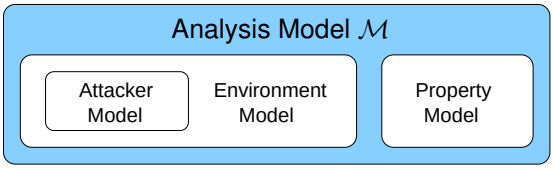


Figure 1.1: Analysis model for security protocols

Both the environment model and the property model, as illustrated in Figure 1.1, should be as general as possible to allow the specification of several protocols and properties, so that we do not need to modify the model whenever we want to analyze a different protocol or state a different property. On the other hand, the attacker model need not be so flexible, since its capabilities are not assumed to change with respect to a particular protocol or goal. Thus a model $\mathcal{M}$ is *parametric* in both protocol P and its security property ϕ, and we can denote this as $\mathcal{M}(P, \phi)$, highlighting the fact that $\mathcal{M}(P, \phi)$ is a function of P and ϕ. $\mathcal{M}(P, \phi)$ is an instance of $\mathcal{M}$ in which P is formalized by the environment model and ϕ is formalized by the property model.

Given a model $\mathcal{M}$, the problem of security protocol analysis can be seen as a *model checking* problem (which shares the same goal as the general problem of software model checking [48]). More precisely, $\mathcal{M}$ should allow to solve the following problem:

Security Decision:	Given a protocol P and a desired security property ϕ, determine whether ϕ holds in $\mathcal{M}(P, \phi)$.

Useful analysis models provide feasible strategies to solve the security decision problem. In general, the choice of the attacker model sets the overall precision of the model. The stronger an attacker is, the stronger the security results given by the model are, but also the more difficult it is to solve the security decision problem.

In one extreme, one could consider an attacker model to consist of *any*, arbitrary computer program restricted only in its (space and time) resources. Such attacker model is realistic but provides no general (nor simple) method to establish the security decision of an arbitrary protocol. A more manageable attacker model is presented within the analysis model we introduce in the next section.

1.1 The Dolev Yao Analysis Model

We now describe the so-called Dolev Yao model, denoted by $\mathcal{M}_{DY}$. The name comes from the attacker model proposed by Dolev and Yao [85], following earlier work of Needham and Schroeder [154]. We describe this model by elaborating on each of its submodels.

Property model The typical property that is considered is *secrecy*, which states that some exchanged value between honest participants remains secret to the attacker. This means that the attacker is not able to reach a state in which it has the secret as part of its knowledge. Other (more involved) notions of secrecy exist [18, 56]. Another classical property is *entity authentication*, which states that principals are correctly identified by their corresponding communicating parties. A protocol meets the goal of authentication when an attacker is not able to impersonate some honest principal successfully. Again, this definition is the simplest, and many others exist [133].

Attacker model The attacker is assumed to have complete network control. Thus, the attacker can intercept, block or redirect any communication action executed by an honest principal. The attacker can also synthesize new messages from the knowledge it has, and communicate these messages to honest participants. This synthesis, which is the ability to create new messages, is precisely defined. For example, the attacker is allowed to pair (i.e. concatenate) and split messages it has in its knowledge, and also to *encrypt* and *decrypt* with known keys. However, if the attacker does not know the correct decryption key of a given *ciphertext* (i.e. an encrypted message), then it can not gain any information from the ciphertext. This assumption is crucial, and it is known as *perfect* or *ideal* encryption. Hence, the attacker is not assumed to be able to cryptanalyse the underlying encryption scheme, but simply treat it as perfect. As with encryption, every cryptographic primitive available to the attacker (e.g. hashing or signature) is similarly idealized, arriving at *perfect cryptography*.

Environment model This model consists of honest participants that execute the protocol faithfully, performing each protocol step exactly as prescribed. Protocols are split into protocol *roles*, which are sequences of actions that prescribe what each principal should perform (the roles typically receive self-explanatory names like *initiator* or *responder*). The actions we consider are basic asynchronous communication actions, i.e. either *sending* or *receiving* messages. The exchanged messages between protocol participants are modelled as terms from a term algebra built up using constructors that represent different data operators (e.g. message pairing or encryption).

There are two main motivations to adopt the $\mathcal{M}_{DY}$ model:

- *Appropriate level of abstraction.* As mentioned in the Summary, many security protocol attacks are based on combinations of the message exchanges and knowledge gathered by the attacker during the execution. Moreover, the attacks are completely independent of the underlying details of the cryptographic primitives employed by the protocol. This justifies the assumption of perfect cryptography. Thus, analyzing security protocols at this level of abstraction is useful since many protocol errors are likely to be uncovered. As a prominent (and extensively used) example, for the Needham-Schroeder protocol [154] (shown in the cover of this thesis), Lowe [131] found an

attack (shown in the back cover of this thesis) seventeen years after its conception. The attack is based purely on the protocol's message flows, and can be perpetrated even if we assume perfect cryptography. Besides Lowe's attack, many other attacks have been found over proposed protocols from the literature (see [71, 15] for surveys).

- *Ease of manageability with formal methods.* Another reason for which $\mathcal{M}_{DY}$ is attractive is that each of its submodels can be described with approaches and languages available within formal methods.

  - The environment model can be seen as a collection of sequential processes running in parallel. Extensive research within the formal methods community has been carried out on concurrent languages that provide both modelling and analysis capabilities (e.g. CCS [152] and CSP [111]).
  - The attacker model can be seen as another process running in parallel with the principals from the environment model, controlling the communication between the principals. The data manipulation abilities of the attacker can be modelled in many ways with formal methods. For example, an attacker can be a process that applies term rewriting over the messages it has in its knowledge. Alternatively, we can think of the attacker as an arbitrary process where the relations between different data are modelled by equational theories over the assumed term algebra.
  - Finally, the property model can also be formalized using well-studied expressive logics (e.g. linear temporal logic) or using process equivalences in concurrent languages.

  Furthermore, the relative simplicity of the Dolev Yao model allows the development of security proof strategies that follow a general form regardless of the actual protocol under analysis. In fact, the security decision problem stated above turns out to be *decidable* [31, 157] when assuming a finite number of participants in the environment model. Still, participants may communicate arbitrarily long messages, thus possibly generating an infinite state space. Decidability means that the question of whether some protocol satisfies a security property can be answered *automatically* by some (smart) computer program, without any human assistance (e.g. [134, 149]). In this context, the efficiency of such a verifier is important in practice, since having decidability in theory alone does not help if the verifier is overly slow. Relaxing the assumption of having a bounded number of participants to allow an unbounded number makes the problem undecidable [87]. Nevertheless, formal techniques using manual theorem provers or automatic *semi*-decision procedures have also been developed successfully (e.g. [155, 55]).

Due to these reasons, during the last decade a large number of formal approaches to analyze security protocols have been proposed in the literature. For the sake of brevity, we do not list these approaches here but refer the interested reader to the survey of Comon and Shmatikov [73] (and the references therein) for an overview. For a survey focused on property models we refer the reader to the recent survey by Meadows [143] (there, property models are called *requirement languages*). Still, we provide extensive related work and references for each of our developments in Chapters 2, 3, 4, 5 and 6.

1.2 Research Question

A model $\mathcal{M}$ can be *extended* to another model $\mathcal{M}'$, where some new quality is taken into account in $\mathcal{M}'$ which is not addressed in $\mathcal{M}$. In principle, $\mathcal{M}'$ may extend $\mathcal{M}$ in any of its submodels alone (i.e. its attacker, environment or property model), or a combination of the submodels. In this thesis we consider extensions of the Dolev Yao model $\mathcal{M}_{DY}$ presented above to answer the following question:

Research Question: In which *useful* ways can we extend the Dolev Yao model $\mathcal{M}_{DY}$ efficiently and effectively ?

Each extended model $\mathcal{M}$ of $\mathcal{M}_{DY}$ enables us to study protocols under new light, thus providing further confidence in the protocol security or uncovering additional threats. This is because $\mathcal{M}$ analyzes a different aspect not covered by $\mathcal{M}_{DY}$. As we already mentioned, models can be refined by extending any of the submodels of Figure 1.1, or a combination of them. The simplest cases involve the extension of one submodel, leaving the other models untouched. However, more complex changes involve multiple submodel modifications. We now illustrate several possible extensions, which describe a range of useful developments that can be built on top of the basic Dolev Yao model. (These are not the actual thesis' contributions, which are listed below in Section 1.3.)

Extending the environment model alone In Section 1.1, the environment model is described as consisting of participants executing the protocol, which are asynchronous communication actions like send and receive. This description is somewhat loose, in the sense that it does not specify (a) how many principals are running the protocol, nor (b) how many steps the protocol has. It also does not say anything about (c) the possibility of principals executing other actions besides just send and receive actions. For (a), we already hinted that one can choose a bounded number of participants (which makes the security decision problem decidable) or extend the environment to allow an unbounded number of participants. For (b), although protocols are typically designed to consist of a finite, a-priori fixed number of exchanges, nothing prevents us from considering cases in which the protocol involves an unbounded exchange of messages (thus modelling *open-ended sessions*). Furthermore, for (c), it is known that some protocols may need other actions than communication to be implemented properly. For example, some protocols may require the ability to check particular conditions (e.g. checking that two values match) before proceeding in their execution. Each of these possible extensions regards only the environment model, and is independent of the attacker or property model.

Extending the environment and attacker model Some other extensions to the environment model also trigger extensions to the attacker model. For example, sometimes assuming that our term algebra used to represent messages is in reality a *free* algebra (i.e. an algebra in which terms are only equal to themselves) is unrealistic, since it provides no relation between messages which represent real-world bit strings known to be related. One example occurs when we use XOR (exclusive-or) as a term constructor. Here, we would like to equate terms built with XOR up to commutativity and associativity.

Furthermore, messages equate to themselves when XOR'ed with the neutral element, and equate the neutral element when XOR'ed twice. Some protocols are designed specifically to use such underlying properties (e.g. Bull's recursive authentication protocol [64]), and thus to model those protocols our environment model has to support such term equalities. However, to be realistic, the attacker should also be aware of such term equalities, and have the ability to exploit them. For a recent survey of algebraic properties used in term algebras for the analysis of security protocols we refer the reader to the work of Cortier et al. [74].

Other extensions to the environment model include refinements to make the model more realistic by modelling real-world qualities, e.g. modelling the passage of time explicitly. This extension provides the facility to model protocols and their participants in a more detailed fashion, including explicit timing issues like timeouts and retransmissions. The attacker model may also be endowed with timing capabilities that help the attacker to deduce facts which are assumed to be kept secret (such attacks are usually called *timing* attacks [120]).

Extending the environment and property model Another useful extension to the property model consists of developing a full language that allows the formulation of sophisticated security properties. This contrasts to using fixed versions of properties, in that a full language provides more flexibility and fine-grained precision of specification of the desired security property. Such languages typically build the properties as assertions about the execution trace and the principals' status at different points of their execution. Thus, to provide a useful property language we need to provide another action (besides communication) available to the principals in the environment model, a so called *status* assertion action, in which principals state details about their current execution state. Then the property language can use the status assertions as atomic predicates to build complex security properties.

Extending the attacker and property model As a last example, suppose we extend the attacker model and let the attacker mount offline dictionary attacks [137] over weak keys (e.g. passwords chosen by humans). Some protocols are designed specifically to avoid such attacks [47], by providing no means to an attacker to verify its guesses. Apart from giving the attacker the possibility to guess keys, we need to extend the property model adequately to allow to specify the property that expresses the inability of the attacker to verify its guesses.

Relating Security Models

Besides extending models, another fruitful research area can be found in *relating* models. Here, we do not create a new model $\mathcal{M}'$ out of $\mathcal{M}$ but take two models $\mathcal{M}'$ and $\mathcal{M}$ and compare them. Typically, informal relations can be expressed in natural language between two different analysis models that achieve roughly the same goal (e.g. our constraint solving approach of Chapter 2 is compared to other existing approaches in Section 2.11, Related Work).

However, in some cases, we can relate two models in a much more formal and rigorous manner. In fact, it has been shown that it is possible to translate formal security proofs from one model to another, and vice versa. Within methods based upon the Dolev Yao model

$\mathcal{M}_{DY}$, one example is in the work of Abadi and Blanchet [20] that relates a typed process calculus and untyped logic programs for secrecy properties. As another example, Bistarelli et al. [53] also relates a process calculus with a multiset rewriting framework.

An interesting recent trend of relating models has been started by the work of Abadi and Rogaway [27]. There, the Dolev Yao model is related to a radically different model, based upon complexity theoretic definitions. Messages are not terms from a term algebra, but strings of bits, and the security properties are based on predicates which state that the attacker advantage in breaking the protocol is probabilistically negligible. Active research is being done in the area to show that security results can be translated between these models.

1.3 Contributions

In this thesis we propose five original orthogonal contributions that build on the Dolev Yao model $\mathcal{M}_{DY}$ of the previous section, as shown in Figure 1.2. In the following we elaborate on each contribution.

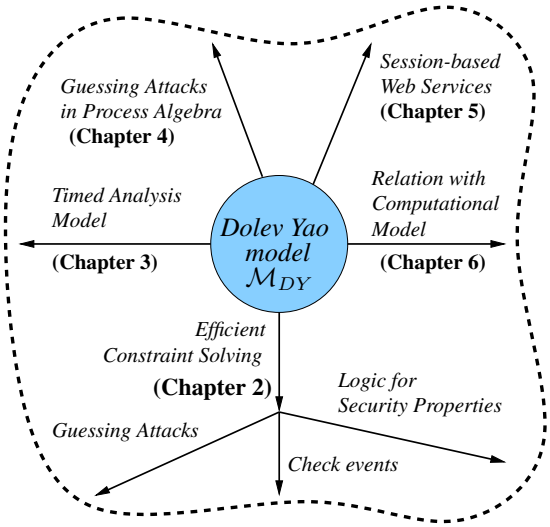


Figure 1.2: Extending and relating the Dolev Yao model $\mathcal{M}_{DY}$

A Constraint Solving Analysis Model (Chapter 2) Our starting point is the work on constraint based analysis of security protocols proposed by Millen and Shmatikov [149]. The model is similar to $\mathcal{M}_{DY}$, although richer since it allows encryption with non-atomic keys, i.e. regular (constructed) terms may be used as encryption keys, something that was not possible in previous approaches. Non-atomic keys are important since many real-life protocols use constructed keys, e.g. establish a session key by computing the hash of two secret *nonces* (i.e. random numbers used as challenges). Millen and Shmatikov present a terminating, sound and complete algorithm to solve

constraint sets derived from protocol executions of a bounded number of participants. We review briefly the preliminaries to constraint solving in Section 2.2. We then develop several extensions:

1. We present an improved decision procedure which works more efficiently than Millen and Shmatikov's in Section 2.3. This improved procedure appears in the paper [5] which is joint work with S. Etalle.
2. In Section 2.5, we extend the property model and allow a full language to specify properties. This language, called $\mathcal{PS}$-LTL, is based on linear temporal logic and it allows great flexibility when specifying security properties. We present an associated decision algorithm. This section contains novel work done in cooperation with A. Saptawijaya and S. Etalle, building on the basic idea of a trace logic presented in previous work [4], which is joint work with A. Durante and S. Etalle.
3. In Section 2.6, we refine the environment model by allowing principals to check equalities between different parts of received messages. As we shall see, merely having communication actions for principals in the environment model does not allow the correct specification of a class of protocols. This refinement appears in the above mentioned paper [5].
4. In Section 2.7, we extend the model to study *guessing attacks* (also known as dictionary attacks). Our refined attacker model is allowed to mount offline dictionary attacks, and obtain guesses for weak keys (e.g. passwords chosen by humans). The property model specifies the property of verifying a guess, thus representing a successful attack. We present a procedure (that appears in the paper [8], which is joint work with S. Etalle, S. Malladi and J. Alves-Foss) to find guessing attacks in constraint solving.
5. We apply our analysis technique to several case studies. The essence of the case studies is described in Section 2.9, while the details can be found in the papers [11, 12, 10] and in the PhD theses of Chong [69], Law [127] and Lenzini [130]. In [10] (joint work with C.N. Chong, B. Ren, J.M. Doumen, S. Etalle and P. Hartel), protocols for license management (used for digital rights management) are developed and analyzed. In [11] (which is joint work with G. Di Caprio, S. Etalle, S. Gnesi, G. Lenzini and C. Moiso), authentication of an (industrial) protocol for telecommunications is analyzed. In [12] (joint work with Y.W. Law, S. Etalle and P. Hartel), the security of proposed wireless sensor network protocols is studied. Finally, we report our experience on the use of the analysis technique as a teaching tool to computer scientist undergraduates at the University of Twente.

A Timed Automata Analysis Model (Chapter 3) We propose a model in which the passage of time is modelled explicitly. We use timed automata to model both the environment and the attacker model, and use the UPPAAL [33] model checker to decide security of simple reachability properties for secrecy and authentication. We illustrate the technique by studying the Yahalom protocol. We present further issues that appear when an attacker can use timing as its abilities, by exposing a timing attack over an authentication protocol. On the positive side, we develop a protocol in which principals use timing to achieve authentication by sending messages at appropriate moments in

time. This section builds on the work presented in the paper [6], which is joint work with S. Etalle, P. Hartel and A. Mader.

A Process Algebraic Model for Guessing Attacks (Chapter 4) We revisit guessing attacks over security protocols (studied already in Section 2.7 under the constraint based framework). This time we consider a model in which the perfect cryptography assumption is relaxed, and an attacker can distinguish certain relations between messages. For example, an attacker can detect whether a ciphertext is really an encrypted message and not just a random value. These relations are modelled as equational theories over a message term algebra in the Applied Pi Calculus [23]. In this calculus, the attacker is represented as an arbitrary context, and security against dictionary is stated as an equivalence between processes. As illustration, we study the EPT and EKE protocol. This work appears in the paper [3], which is joint work with J. Doumen and S. Etalle.

A Process Algebraic Model for Session-based Web Services (Chapter 5) We consider the analysis of web service security protocols, building upon earlier work of Bhargavan et al. [50]. In particular, we use the language TulaFale [52] (a process algebraic language supporting XML encoded data) to study session based web service security protocols. We formalize and provide semantics to two industrial specifications, namely WS-Trust [117] and WS-SecureConversation [116]. Furthermore, we build a realistic analysis model that models insider attacks and use it to analyze typical protocols relying on these specifications. Finally, we apply our models to analyze a concrete protocol used recently in the Interoperability workshop [166] by several industrial companies to test their web service implementations. This chapter builds on previous work [1], which is joint work with K. Bhargavan, C. Fournet and A.D. Gordon.

Relating Analysis Models (Chapter 6) Abadi and Rogaway [27] relate the Dolev Yao model $\mathcal{M}_{DY}$ to a computational counterpart model based on complexity theoretic definitions. In their seminal work, Abadi and Rogaway consider a term algebra in which messages may be encrypted with atomic keys only. To merge that result with the formal method based on constraint solving of Chapter 2, we consider a term algebra that allows non-atomic encryption with constructed (also called composed) keys, and reestablish Abadi and Rogaway's result. This work appears in the paper [9], which is joint work with P. Laud.

1.4 Conclusion and Outlook

In this work we answer the Research Question of Section 1.2 positively, by proposing several useful extensions to the Dolev Yao model $\mathcal{M}_{DY}$ of Section 1.1. While some extensions emphasize efficiency, others provide effective means to model certain security qualities. We also illustrate a relation between a Dolev Yao model and a computational model.

Typically, the process of protocol specification in a given formalism already uncovers many potential issues about the protocol, and provides a good understanding of the protocol behaviour. Thus, the whole process of modelling is an interactive process during which the protocol is already under analysis. The same applies to modelling the protocols in the extensions provided in this thesis.

Our extensions are of course neither definitive nor complete; there is always room for further improvement in our proposed methods. In general, we observe historically that formal models are always behind new developments in cryptography. For example, only recently the formal approaches considered probabilistic encryption (e.g. [23]) while the original cryptographic development was introduced more than twenty years ago [99]. This lag in time can be explained partially by the fact that recasting the cryptographic problems into the (formal) analysis models takes a non trivial amount of time.

Still, the reader may (fairly) ask whether these extensions (and relations) are only coming from technical improvements, which in such case would imply that the research community could reach a 'fix point' in which the ultimate solution for analyzing security protocols has been developed.

Our answer is negative: even if we assume that improvements on the analysis techniques could eventually reach such a 'fix point', we argue that new analysis models will *always* be needed, mainly because new protocols are developed continuously, arising from the evolution of communication methods and cryptographic developments:

New environments New protocols are needed to cope with new situations. For example, Perrig et al. [156] proposed a new protocol called TESLA, to deal with authentication in a wireless network environment, which is a particularly lossy channel. To be able to analyze TESLA new methods were required (e.g. [139, 63]), that model the particular communication details.

New cryptographic primitives New primitives are developed constantly by the cryptographic community. For example, active research is being done on identity based encryption (IBE) [58], which means that several security protocols using IBE are going to be derived from existing ones.

In general, the field of security advances by a process of trial and error: existing security constructions (e.g. protocols) are continuously reviewed, and replaced by new proposals as soon as the construction is found to be unsatisfactory (e.g. an attack or a more efficient construction is found), and the process is repeated henceforth. In this thesis we contribute to the research community by proposing analysis models that help in the reviewing process of security protocols, which are constructions particularly difficult to design correctly.

CHAPTER 2

A Constraint-based Analysis Model

2.1 Introduction

In this chapter we instantiate the general model described in the Introduction (shown in Figure 1.1) and develop an analysis model based on constraint solving, as illustrated in Figure 2.1.

In this model, the analysis is performed by first instantiating the environment model with a particular (finite) set of principals. Then, this instance (the *system scenario*) is fed to a constraint solving procedure, whose goal is to find a protocol execution that leads to an attack. Protocol executions are represented by constraints the attacker has to meet in order to perform the executions. Thus, reaching a particular protocol execution (which represents an attack on the protocol under study) is equivalent to solving a set of constraints, hence the name of the technique.

Millen and Shmatikov proposed an original constraint solving procedure [149], based upon a sound and complete reduction procedure (see Section 2.2). Building on top of this work, we propose an analysis procedure in Section 2.3. Our procedure is more efficient than the procedure of [149] since it considers less protocol executions when searching for an attack, without losing precision.

As we mentioned, the constraint solving procedure looks for protocol executions that represent attacks. However, what specifies an attack in a given execution depends on which security property we are considering, as modelled by the property model. In our model, the analysis procedure queries a *termination condition* predicate that decides whether a given protocol execution represents an attack or not. If the termination condition holds, the procedure can stop its search and report the protocol execution that represents the attack. Otherwise, the procedure continues searching for an appropriate protocol execution. In this chapter, we consider three different termination conditions:

1. A simple role termination condition that can be used to check basic secrecy and authentication properties, as detailed in Section 2.2;

2. A more involved termination condition predicate that allows the use of a specialized temporal language to specify security properties (called $\mathcal{PS}$-LTL), developed in Section 2.5; and

3. A particular termination condition that can be used to test the security against guessing attacks (developed in Section 2.7).

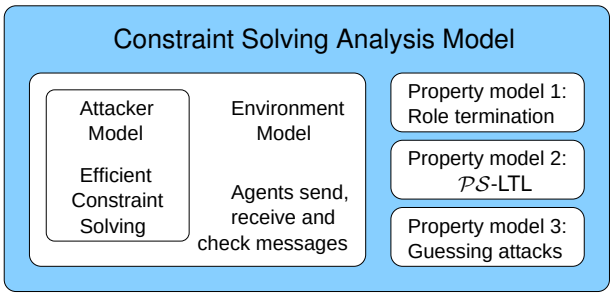


Figure 2.1: Analysis Model based on Constraint Solving

In Section 2.2 we introduce preliminary concepts needed in the rest of the chapter, presenting the technique of constraint solving as seminally introduced by Millen and Shmatikov. After presenting the improved verification algorithm in Section 2.3, we introduce negated constraints in Section 2.4, which are used subsequently in Sections 2.5 and 2.7. In Section 2.6, we extend the environment model and allow principals to send, receive and also *check* messages. The complete analysis model is presented in Section 2.8. Then, we report on case studies in Section 2.9 and our experience on using our methods as a teaching tool in Section 2.10. We discuss related work on Section 2.11 and finally conclude in Section 2.12.

2.2 Preliminaries

In this section we review the terminology we need in the rest of the chapter. We first introduce our term algebra and intruder deduction rules, then the security protocol model, and finally the technique of constraint solving for security protocol analysis.

2.2.1 Terms and Rules

Term Algebra Messages are represented as terms in a free algebra $\mathcal{T}$ generated by the operators in Table 2.1 (left), from an enumerable set of variables $\mathcal{V}$ (denoted by uppercase letters $A, B, Na, K, \dots$), and an enumerable set of constants $\mathcal{C}$ (denoted by lowercase $a, b, na, k, \dots$), representing the principal identities, nonces and keys. A special constant $e \in \mathcal{C}$ is distinguished to denote the intruder's identity. We have constructors for representing public keys, pairing, message hashing, symmetric and asymmetric encryption, and signature. Private keys in asymmetric encryption are not modelled in the term algebra since we assume that these keys are never part of messages in protocols. This assumption is realistic, as every protocol we consider does not send its private key in messages. Finally, we sometimes denote the syntactic equality of two terms t_1 and t_2 as $t_1 \equiv t_2$, which holds iff t_1 is exactly the same term as t_2.

Further definitions and terminology For $i \leq j$, the integer interval $\{i, i+1, \ldots, j-1, j\}$ is denoted as $[i \ldots j]$. The set of *ground* terms, denoted by $\mathcal{T}^+$, is generated like $\mathcal{T}$ above but only from constants $\mathcal{C}$ and excluding $\mathcal{V}$. When $t \in \mathcal{T}^+$, we say that t is *ground*, otherwise it is *non-ground*. The variables of a term t are denoted as $var(t)$.

Substitutions (denoted by $\sigma, \rho, \gamma, \ldots$) are finite mappings from $\mathcal{V}$ to $\mathcal{T}$. Ground substitutions map $\mathcal{V}$ to $\mathcal{T}^+$; the domain of σ is denoted as $dom(\sigma)$. The empty substitution is denoted as ε. Given $v \in \mathcal{V}$ and $t \in \mathcal{T}$, $[^t/_v]$ denotes the singleton substitution that maps v to t. Given a term $t \in \mathcal{T}$ and a substitution σ, $t\sigma$ denotes the term resulting from substituting each occurrence of $v \in dom(\sigma)$ in t by $\sigma(v)$. A term t' is an *instance* of another term t if there is a substitution σ s.t. $t' = t\sigma$. The same terminology is used for the (later introduced) events, protocol roles and traces.

$t_1, t_2 ::=$	c	constant in $\mathcal{C}$	$\{t_1, t_2\}$	$\rightarrow_{pair}$	(t_1, t_2)
	v	variable in $\mathcal{V}$	$\{(t_1, t_2)\}$	$\rightarrow_{first}$	t_1
	$pk(t_1)$	public key	$\{(t_1, t_2)\}$	$\rightarrow_{second}$	t_2
	(t_1, t_2)	pair	$\{t\}$	$\rightarrow_{hash}$	$h(t)$
	$h(t_1)$	hash	$\{t_1, t_2\}$	$\rightarrow_{senc}$	$\{t_1\}_{t_2}$
	$\{t_1\}_{t_2}$	symmetric encryption	$\{\{t_1\}_{t_2}, t_2\}$	$\rightarrow_{sdec}$	t_1
	$\{t_1\}^{\rightarrow}_{t_2}$	asymmetric encryption	$\{t_1, t_2\}$	$\rightarrow_{penc}$	$\{t_1\}^{\rightarrow}_{t_2}$
	$sig_{t_1}(t_2)$	signature	$\{\{t_1\}^{\rightarrow}_{pk(e)}\}$	$\rightarrow_{pdec}$	t_1
			$\{t_1\}$	$\rightarrow_{sig}$	$sig_{pk(e)}(t_1)$

Table 2.1: Grammar for terms (left) and DY rules (right)

DY Rules Rules are used to represent the abilities of the intruder. Let A be a set of terms and t a term, and let ℓ be a rule label, stating the name of the rule. A *rule* is denoted by $A \rightarrow_\ell t$. We work with the set of rules given in Table 2.1 (right), where t_1 and t_2 are terms in $\mathcal{T}$. As usual, the attacker is allowed to pair and split terms, hash, symmetrically encrypt terms with any (possibly non-atomic) key and decrypt symmetrically if the key is known to the attacker. Public-key encryption ($penc$) is modelled by allowing to encrypt with any key. However, rule *pdec* only allows the asymmetric decryption of a term encrypted with the attacker's public key. The attacker cannot decrypt any term encrypted with a different public key than his own, since we assume that private keys are never leaked (as they do not take part of any message). Moreover, the attacker can only sign terms using his private key, represented in rule *sig* (here $pk(e)$ is the public key that is needed to verify the signature).

We now define $\mathcal{F}(T)$ (the *fake* operation), representing the terms the intruder can generate from the set of ground terms T:

Definition 2.2.1. *Let T be a ground term set, i.e. $T \subseteq \mathcal{T}^+$. Then, $\mathcal{F}(T)$ is defined as $\cup_{n \geq 0} \mathcal{F}^n(T)$, where $\mathcal{F}^n(T)$ is the set defined inductively as follows:*

$$\begin{aligned} \mathcal{F}^0(T) &= T \\ \mathcal{F}^n(T) &= \mathcal{F}^{n-1}(T) \cup \{t \mid A \rightarrow_\ell t \text{ is a DY rule and } A \subseteq \mathcal{F}^{n-1}(T)\} \end{aligned}$$

Intuitively, when the attacker knows (e.g. has eavesdropped) a set of messages T, then he can compute the set of terms $\mathcal{F}(T)$.

Example 2.2.2. *Let $T = \{e, \{na\}_k, \{(k,e)\}_{h(e)}\}$. We now show that $\{k, na\} \subseteq \mathcal{F}(T)$. Since $e \in T = \mathcal{F}^0(T)$, applying rule $hash$ we deduce that $h(e) \in \mathcal{F}^1(T)$. Then using rule $sdec$, we obtain that $(k,e) \in \mathcal{F}^2(T)$. Using rule $first$ we find that $k \in \mathcal{F}^3(T)$, and finally using rule $sdec$ again we deduce that $na \in \mathcal{F}^4(T)$.*

2.2.2 Protocol Model

Our protocol model is related to the strand-space formalism [162], although we sometimes use a different terminology, e.g. we call system scenario what in strand spaces is called a *semibundle*. In the following, we introduce events, traces, protocol roles, and system scenarios.

Definition 2.2.3. *A* communication *event is a pair $\langle a : m \diamond b \rangle$ where a, b are variables or principal constants, $\diamond \in \{\triangleleft, \triangleright\}$ and m is a term. a is called the* active *party, and b is the* passive *party.*

Since we will let the attacker intercept and forge communication messages, the event $\langle a : m \triangleright b \rangle$ reads as "principal a sends message m with *intended destination* b". Symmetrically, $\langle a : m \triangleleft b \rangle$ stands for "principal a receives message m *apparently from* b".

Communication events are not the only events we consider in this chapter. Later we use *status* events in Section 2.5 and *check* events in Section 2.6.

Example 2.2.4. *The following are examples of communication events:*

- *$\langle a : (a, na) \triangleright B \rangle$ is a communication event in which principal a sends pair (a, na) to another principal B. Since B is a variable (rather than a constant), this event represents any concrete communication event $\langle a : (a, na) \triangleright c \rangle$ for any principal c. In other words, B is still undetermined. (In fact, B can take any term value, and not just an principal identity, due to the untyped behaviour of the analysis. Still, attacks in which principal variables (like B above) are instantiated to non principal constants are unrealistic and will be ignored.)*

- *$\langle b : (A, N_A) \triangleleft A \rangle$ is a communication event in which principal b receives the pair (A, N_A) from another principal A. Similarly, because of the presence of variables A and N_A, this stands for any concrete event $\langle b : (c, n) \triangleleft c \rangle$ for any principal c and any term n.*

Definition 2.2.5. *A* protocol role *is a finite sequence of events in which all events share the same active principal.*

Given a protocol written in standard '$A \rightarrow B : M$' notation (where the intended meaning is that A sends message M to B), it is straightforward to obtain its parametric protocol roles, as shown in the next example.

Consider the BAN Concrete Andrew Secure RPC protocol [65] (with the last message 4 stripped out since it is not necessary for security) shown as Protocol 2.1.

First a sends a message with her identity and a fresh nonce n_a. Upon receipt, b generates a short term session key k_{st}, encrypts it along with a's nonce n_a using the long term key k_{lt}, shared previously with a. Finally, a replies with her nonce n_a encrypted with the newly established key k_{st}.

Message 1.	$a \rightarrow b : (a, n_a)$
Message 2.	$b \rightarrow a : \{(n_a, k_{st})\}_{k_{lt}}$
Message 3.	$a \rightarrow b : \{n_a\}_{k_{st}}$

Protocol 2.1: BAN Concrete Andrew Secure RPC protocol (Messages 1-3)

Protocol notation caveat While describing protocols throughout this chapter, like in Protocol 2.1 above, we denote the principal identities and exchanged elements in lowercase (to prevent confusion with the variables denoted with uppercase). Those elements are to be understood as meta variables used to describe the protocol behaviour, and should *not* to be understood as constants from our term algebra (e.g. n_a in Protocol 2.1 is not *the* nonce $n_a \in \mathcal{C}$, but a generic nonce). In the subsequent chapters, however, protocols are denoted using standard uppercase letters, as the notation of variables does not clash anymore.

Example 2.2.6. *In the following, we name protocol roles such as* init*, and* resp*, denoting an initiator and a responder respectively. The parametric protocol roles of Protocol 2.1 are then:*

$$\begin{aligned}
\text{init}(A, B, N_A, K_{lt}, K_{st}) &= \langle\, \langle A : (A, N_A) \triangleright B\rangle\, \langle A : \{(N_A, K_{st})\}_{K_{lt}} \triangleleft B\rangle \\
&\qquad \langle A : \{N_A\}_{K_{st}} \triangleright B\rangle\, \rangle \\
\text{resp}(A, B, N_A, K_{lt}, K_{st}) &= \langle\, \langle B : (A, N_A) \triangleleft A\rangle\, \langle B : \{(N_A, K_{st})\}_{K_{lt}} \triangleright A\rangle \\
&\qquad \langle B : \{N_A\}_{K_{st}} \triangleleft A\rangle\, \rangle
\end{aligned}$$

The next step consists of gathering several protocol roles together, which provides a particular system instance.

Definition 2.2.7. *A* system scenario *is a multiset of protocol roles.*

A system scenario determines which sessions are present, and which principals play which roles.

Example 2.2.8. *Consider the following simple system scenario, where* init *and* resp *are the roles defined in Example 2.2.6:*

$$\begin{aligned}
Sc &= \{\text{init}(a, B, na, k_{lt}, K_{st}), \text{resp}(a, b, N_A, k_{lt}, k_{st})\} \\
&= \{\langle\langle a : (a, n_a) \triangleright B\rangle\langle a : \{(n_a, K_{st})\}_{k_{lt}} \triangleleft B\rangle\langle a : \{n_a\}_{K_{st}} \triangleright B\rangle\, \rangle, \\
&\qquad \langle\langle b : (a, N_A) \triangleleft a\rangle\langle b : \{(N_A, k_{st})\}_{k_{lt}} \triangleright a\rangle\langle b : \{N_A\}_{k_{st}} \triangleleft a\rangle\, \rangle\}
\end{aligned}$$

This scenario is obtained by partially instantiating the roles of Example 2.2.6. The initiator is played by a, using fresh nonce na and shared key k_{lt}, while the responder is b, using the shared key k_{lt} and the (freshly created) session key k_{st}. B denotes the fact that initially a does not know to which participant she will be talking to, and what session key (K_{st}) she is going to use. Similarly, initially the Responder b does not know the nonce N_A it will receive.

We further require that each protocol role in a system scenario satisfies the *origination assumption*: all uninstantiated variables need to occur in a receive event before they occur in a send event (see [149] for details). This assumption does not result in a loss of generality

since protocols can be easily specified to satisfy the assumption; the assumption is needed to show *monotonicity*, which states that the attacker never forgets messages it knows, and also prohibits from learning messages it never knew. Monotonicity helps in the proof of soundness of completeness (Theorem 2.2.15 below)

2.2.3 Security Property Specification

We now show how Millen and Shmatikov [149] in their seminal work encode *secrecy* and *authentication* inside system scenarios. The encoding of these properties is ad-hoc, and does not follow the modularity advocated in the Introduction. In Section 2.5 we propose a different, more powerful language to express security properties.

Secrecy Secrecy of some term x can be checked by adding an artificial role

$$secret(x) = \langle e : x \triangleleft e \rangle$$

to a scenario, and ensuring that there is no corresponding role with event $\langle e : x \triangleright e \rangle$. Hence, if role $secret(x)$ can finish its execution, then we know that the intruder provided x, and thus it can generate x, breaking secrecy.

Example 2.2.9. *A scenario Sc' with a secrecy check of k_{st} in Protocol 2.1 is $Sc' = Sc \cup \{secret(k_{st})\}$, where Sc is the scenario defined in Example 2.2.8.*

Authentication Checking authentication is done by crafting system scenarios in which certain principals are missing, while the present ones finish their execution believing the missing principal is present.

Example 2.2.10. *Consider scenario $Sc'' = \{\text{resp}(a, b, N_A, k_{lt}, k_{st})\}$ from Example 2.2.6, of Protocol 2.1. Here, if the responder* resp *ever finishes it is execution believing it is talking to principal a, Protocol 2.1 does not provide authentication of a to b, since the intruder could impersonate a without b noticing.*

This way of checking authentication follows the way presented in [149]. Later, Millen implemented a more refined way of checking authentication [147] in his Prolog implementation, see Section 2.5.

2.2.4 Constraint Solving

Central to this chapter is the notion of *constraint* and *constraint set*, as defined below.

Definition 2.2.11. *A* constraint *is a pair $m : K$, of a term m and a term set K (standing for* knowledge*). A constraint is called* simple *if m is a variable, i.e. $m \in \mathcal{V}$. A* constraint set *CS is a finite set of constraints; CS is simple if each constraint in the set is simple.*

From scenarios to interleavings Given a scenario Sc (as in Example 2.2.8 above), we can obtain *interleavings* of events. An interleaving ν of $Sc = \{r_1, \ldots, r_n\}$ is obtained by concatenating events from r_i, $i \in [1..n]$. Formally, ν is an interleaving of Sc if $\nu \in ||Sc$, for $||Sc$ defined as: $||Sc = \cup_{\langle a\ r\rangle \in Sc}\ a.(||(Sc \setminus \{\langle a\ r\rangle\} \cup \{r\})$, with $s.Sc = \{\langle s\ r\rangle \mid r \in Sc\}$.

Example 2.2.12. *A possible interleaving ν of scenario Sc from Example 2.2.8 is:*

$$\begin{array}{rcl} \nu & = & \langle\ \langle a : (a, na) \triangleright B\rangle\ \langle b : (a, N_A) \triangleleft a\rangle\ \langle b : \{(N_A, k_{st})\}_{k_{lt}} \triangleright a\rangle \\ & & \langle a : \{(na, K_{st})\}_{k_{lt}} \triangleleft b\rangle \langle a : \{na\}_{K_{st}} \triangleright B\rangle\ \ \langle b : \{N_A\}_{k_{st}} \triangleleft a\rangle\ \rangle \end{array}$$

Given a scenario Sc, the number of possible interleavings (i.e. the cardinality of $||Sc$) grows exponentially w.r.t. the number of roles in Sc (i.e. the cardinality of Sc). Since the security analysis searches through possible interleavings which may represent potential attacks, the optimization of the number of interleavings that need to be considered is an important task. In Section 2.3 we develop such an optimization.

From interleavings to constraint sets Each interleaving is transformed to a constraint set, as follows. Each *send* communication event is added to an accumulating knowledge set, while each *receive* communication event is translated to a constraint. Formally, given an interleaving ν and a knowledge set K we let the mapping $cs(\nu, K)$ be defined as:

$$\begin{array}{lcl} cs(\langle\rangle, K) & = & \emptyset \\ cs(\langle\langle x : m \triangleright y\rangle\ \nu'\rangle, K) & = & cs(\nu', K \cup \{m\}) \\ cs(\langle\langle x : m \triangleleft y\rangle\ \nu'\rangle, K) & = & \{m : K\} \cup cs(\nu', K) \end{array}$$

The *initial intruder knowledge* set, denoted IK, is a ground set of terms representing what the intruder knows before starting the analysis of a specific scenario. This set includes the intruder identity e, and may include other public or previously known information, like principal identities or public keys. Given an interleaving ν, we calculate its corresponding constraint set as $cs(\nu, IK)$.

Example 2.2.13. *The constraint set $cs(\nu, IK)$ corresponding to interleaving ν from Example 2.2.12 and initial intruder knowledge IK is:*

$$\begin{array}{rcl} cs(\nu, IK) & = & \{(a, N_A) : IK \cup \{(a, na)\}, \\ & & \ \{(na, K_{st})\}_{k_{lt}} : IK \cup \{(a, na), \{(N_A, k_{st})\}_{k_{lt}}\}, \\ & & \ \{N_A\}_{k_{st}} : IK \cup \{(a, na), \{(N_A, k_{st})\}_{k_{lt}}, \{na\}_{K_{st}} \\ & & \} \end{array}$$

The generated constraint set omits information regarding the actual sender and intended receiver and vice versa; this models the fact that an attacker can impersonate the originators simply if the attacker is able to compose the necessary messages. Still, the recorded information regarding senders and receivers is useful to state the security properties of the protocol, as done in Section 2.5.

The mapping $cs(\cdot, IK)$ never 'forgets' IK. More formally, for any ν, IK, and $m : K \in cs(\nu, IK)$,

$$IK \subseteq K \tag{2.1}$$

Solving a constraint set A *solution* of a constraint set is a substitution that makes every constraint in the set to be *solvable*. We formalize this notion in the next definition, using the fake operator $\mathcal{F}(\cdot)$ introduced in Definition 2.2.1.

Definition 2.2.14. *Let $CS = \{m_i : K_i \mid i \in [1 \dots n]\}$ be a constraint set, and let σ be a ground substitution for all the variables in CS. We say that σ is a* solution *of CS if for each $i \in [1 \dots n]$, $m_i\sigma \in \mathcal{F}(K_i\sigma)$. We also say that CS is* solvable *if there exists at least one solution σ of CS. A* partial *solution γ of CS is a substitution s.t. $dom(\gamma) \subseteq var(CS)$, and $CS\gamma$ is solvable.*

Let CS be a simple constraint set (see Definition 2.2.11) s.t. each constraint $X : T_X \in CS$ satisfies that T_X includes the initial intruder knowledge IK (recall that IK is ground), i.e. $IK \subseteq T_X$. Using the assumption that $e \in IK$, we now show that CS is solvable. Let σ be the substitution that assigns e to each variable $X \in \mathcal{V}$. Then σ is a solution of each constraint $X : T_X \in CS$, since $X\sigma = e \in IK = IK\sigma \subseteq T_X\sigma \subseteq \mathcal{F}(T_X\sigma)$.

We anticipate that every simple constraint set considered in this chapter is derived from translation $cs(\cdot, \cdot)$ from interleavings to constraint sets, thus including IK by (2.1). This implies that simple constraint sets are always solvable.

Millen and Shmatikov's reduction algorithm (called $\mathbf{P}$ in the following) [149] reduces a constraint set CS to (a possibly empty set of) pairs of simple constraint sets CS' and substitutions γ. We do not explicitly define $\mathbf{P}$ here but rather use it as a black box (since our extensions do not concern its details), relying on $\mathbf{P}$'s properties:

Theorem 2.2.15 ([149][1])**.** *(a)* $\mathbf{P}$ *always* terminates*. (b)* Soundness: *If* $\mathbf{P}$ *applied to CS outputs (CS', γ), then γ is a partial solution of CS, and every solution of CS' is also a solution of $CS\gamma$. (c)* Completeness: *If CS is solvable with solution σ, then applying* $\mathbf{P}$ *to CS returns some (CS', γ) such that, for some solution σ' of CS', $\sigma = \gamma\sigma'$.*

Completeness is the most difficult to achieve in the above result, due to the existence of non-atomic keys for symmetric encryption. Millen and Shmatikov [149] use an extra deduction rule and terms called *encryption hiding*, used to flag symmetric decryptions so that procedure $\mathbf{P}$ does not loop uselessly. We do not include encryption hiding in our description since it is not required in our improvement, shown below; however, it should be noted that indeed encryption hiding is assumed, to be able to guarantee Theorem 2.2.15. This means that our term algebra provides such a constructor, and a corresponding rule allows the deduction of such terms.

Example 2.2.16. *Let $cs(\nu, IK)$ be the constraint set of Example 2.2.13. Applying* $\mathbf{P}$ *returns $(\emptyset, \sigma)$, for $\sigma(Na) = na$ and $\sigma(K_{st}) = k_{st}$. It is straightforward to check that σ is a solution (see Definition 2.2.14) of $cs(\nu, IK)$.*

We are ready to summarize the protocol analysis procedure as presented in [149]:

Procedure 2.2.17 ([149][2])**.** *The analysis of a scenario Sc w.r.t. an initial intruder knowledge IK consists of:*

[1] We actually reformulated the result [149] using the terminology of partial solutions as given in Definition 2.2.14.

[2] The actual procedure formulation as we give it here is implicitly described in [149]; Our step 1 is detailed in Section 3.1 of [149], while step 2 is described in Section 3.2 of [149]. Step 3 concerns Section 4 of [149].

1. *Pick an* interleaving ν *of* Sc *(i.e.* $\nu \in ||Sc$*);*

2. *Obtain its* constraint set $cs(\nu, IK)$*;*

3. Solve $cs(\nu, IK)$*, i.e. apply* $\mathbf{P}$*: If success, stop and report an attack. Otherwise, go back to Step 1.*

There are many interleavings to consider in Step 1 of Procedure 2.2.17 above, given by the cardinality of $||\cdot$. This is calculated precisely in Section 2.3.1, although now we illustrate the case of $|Sc| = 2$. For such a scenario Sc with two roles, one of n events and another of m events, then we have $\binom{m+n}{n}$ possible interleavings, which gives an exponential number of cases. Some interleavings can avoided, as shown in [149]: An interleaving ν *dominates* another interleaving ν' if for each constraint $m : K' \in cs(\nu', IK)$ there is another constraint $m : K \in cs(\nu, IK)$ with $K' \subseteq K$. It can be seen that every solution of $cs(\nu', IK)$ is also a solution of $cs(\nu, IK)$, so there is no need to consider interleaving ν' if we already considered ν. Typically, dominating interleavings can be chosen by always selecting a *send* event *before* a *receive* event, when generating interleavings for Step 1 in Procedure 2.2.17.

2.3 Improving Constraint Solving

In this section we propose a new verification procedure that considers fewer interleavings, without losing attacks. Let Sc be a system scenario. We say that ν is a *prefix interleaving* of Sc if there is an interleaving $\nu' \in ||Sc$ s.t. ν is a prefix of ν'. Also, we say that a constraint set CS is *unsolvable* if CS has no solution, as in Definition 2.2.14. Our new procedure is motivated by the following simple observations.

Observation 2.3.1. *1. Let* ν*,*ν' *be two interleavings of* Sc *s.t.* ν *is a prefix of* ν'*. Then* $cs(\nu, IK) \subseteq cs(\nu', IK)$*.*

2. Let CS_1 *and* CS_2 *be two constraint sets s.t.* $CS_1 \subseteq CS_2$ *and* CS_1 *is unsolvable. Then* CS_2 *is also unsolvable.*

Combining 1. and 2. from Observation 2.3.1 we obtain the following fact:

Remark 2.3.2. *If* ν *is a prefix of* ν' *and* $cs(\nu, IK)$ *is unsolvable, then* $cs(\nu', IK)$ *is also unsolvable.*

So, whenever we find that an interleaving ν has its corresponding constraint set $cs(\nu, IK)$ unsolvable, we immediately know that any interleaving ν' that extends ν (i.e. ν is a prefix of ν') is *also* unsolvable, and therefore does not have to be considered.

Prefix interleavings from a scenario Sc may also be seen as execution *traces*, usually denoted as tr. Appending an event ev to trace tr is written $\langle tr\ ev \rangle$. Functions $last$ and $length$ have the usual meaning: $last(\langle tr\ ev \rangle) = ev$ (*last* is undefined for the empty trace), $length(\langle\rangle) = 0$ and $length(\langle tr\ ev \rangle) = length(tr) + 1$. The prefix trace consisting of the first i events is denoted as tr_i, with $tr_0 = \langle\rangle$ and $tr_m = tr$ for $m \geq length(tr)$.

We say that a trace tr is *derived* from Sc if there exists an instance Sc' of Sc s.t. tr is a prefix interleaving of Sc'.

Definition 2.3.3. *The knowledge of a trace tr is given by* $K(tr) = \{m \mid last(tr_i) = \langle a : m \triangleright b\rangle, i \in [1 \dots length(tr)]\}$

We are now ready to describe an algorithm which, given a system scenario Sc and an initial intruder knowledge IK, non-deterministically produces a set of traces. This procedure differs from Procedure 2.2.17 in that we do not consider different interleavings and then try to solve the corresponding constraint set, but directly execute the scenario, by incrementally adding events during an execution and checking that the constraint set remains solvable. This ensures that we never consider unsolvable interleavings.

Procedure 2.3.4. *A* state *is a 4-tuple $\langle Sc, IK, CS, tr\rangle$, where Sc is a system scenario, IK is the initial intruder knowledge, CS is a simple constraint set and tr is a (possibly non-ground) trace. An* execution step *from state $\langle Sc, IK, CS, tr\rangle$ to $\langle Sc', IK, CS', tr'\rangle$ is obtained by performing the following:*

1. *Choose non-deterministically a non-empty role $r \in Sc$. Let $r = \langle ev\ r'\rangle$. Consider the following cases for ev:*
   (a) *If ev is a* send *communication event, let γ be the empty substitution and CS'' be CS.*
   (b) *If ev is a* receive *communication event, i.e. $ev = \langle a : m \triangleleft b\rangle$, check that the intruder can generate m using the knowledge $K(tr) \cup IK$, by applying procedure* **P** *to $CS \cup \{m : (K(tr) \cup IK)\}$, obtaining a new simple constraint set CS'' and a partial solution γ (Note that there may be many possible CS'' and γ).*
2. *Let $Sc' := (Sc \setminus \{r\} \cup \{r'\})\gamma$, $CS' := CS''$ and $tr' := \langle\ tr\gamma\ ev\gamma\ \rangle$.*

A run for Sc_0 with initial intruder knowledge IK *is a sequence of execution steps starting from state $\langle Sc_0, IK, \emptyset, \langle\rangle\rangle$.*

Given a run $s_1 \cdot s_2 \cdots s_n$ for Sc with IK from Procedure 2.3.4, where s_i ($i \in [1 \dots n]$) are states and $s_n = \langle Sc_n, IK, CS_n, tr_n\rangle$, we usually say that tr_n is an output trace of Procedure 2.3.4, instead of referring to the full run. We also say that s_n is a resulting state (or just a state) from Procedure 2.3.4.

Soundness and completeness of Procedure 2.3.4 Suppose we have a ground trace $tr = \langle tr'\ ev\rangle$, with $ev = \langle a : m \triangleleft b\rangle$. We say that the event ev in tr is *valid* if the intruder could produce m using $IK \cup K(tr')$. A whole trace is valid when all its receive communication events are valid, as shown in the next definition.

Definition 2.3.5. *A ground trace tr is* valid *w.r.t. IK if for each $i \in [0 \dots length(tr) - 1]$, $last(tr_{i+1}) = \langle a : m \triangleleft b\rangle$ implies that $m \in \mathcal{F}(K(tr_i) \cup IK)$.*

Every (and only!) valid traces should be output in states of Procedure 2.3.4. From Theorem 2.2.15 it follows that this procedure is sound and complete, as shown next.

Theorem 2.3.6. *Let Sc_0 be a system scenario and let IK be an initial intruder knowledge. For Procedure 2.3.4, it holds:*

1. *Soundness: Let $\langle Sc, IK, CS, tr\rangle$ be a resulting state in a run for Sc_0 with initial intruder knowledge IK. Then for every solution σ of CS, (i) $tr\sigma$ is derived from Sc_0 and (ii) $tr\sigma$ is valid w.r.t. IK.*

2. *Completeness: Let tr be a valid trace w.r.t. IK derived from Sc_0. Then there exists a resulting state $\langle Sc, IK, CS, tr'\rangle$ in a run for Sc_0 w.r.t. IK and a substitution σ s.t. σ is a solution of CS and $tr = tr'\sigma$.*

Proof. See Appendix A.1. □

2.3.1 Attacks, Partial Runs and the Termination condition

Procedure 2.2.17 considers complete interleavings where all events from a scenario Sc_0 are included. It is shown [149] that if any interleaving is solvable, then an attack is found.

Differently from Procedure 2.2.17, our Procedure 2.3.4 outputs execution traces which are instances of prefix interleavings representing *partial runs*, in which not every participant necessarily finishes its execution. For example, the empty trace $\langle\rangle$ is always output regardless of the input scenario Sc_0 or IK in Procedure 2.3.4. Clearly, the empty trace does *not* represent an attack on the protocol in question. We thus need to include a *termination condition* $TC(s)$ which is evaluated in each resulting state s of a run from Procedure 2.3.4. When $TC(s)$ holds, we know that an attack has happened and we can then stop execution and report the attack trace. Otherwise, execution should continue in search for another attack.

In the following we instantiate $TC(\cdot)$ with a basic termination condition to check basic secrecy and authentication properties. In the next sections we propose other instantiations to $TC(\cdot)$, in which the termination condition allows to cover complex properties expressed in a temporal language in Section 2.5 and security against guessing attacks in Section 2.7.

Definition 2.3.7. *Let Sc_0 be a scenario and let A be a set of protocol roles s.t. $A \subseteq Sc_0$. Let $\langle Sc, IK, CS, tr\rangle$ be a state in Procedure 2.3.4. We define the predicate $TC_A(\langle Sc, IK, CS, tr\rangle)$ to hold when for every role $r \in Sc$ with $r \neq \langle\rangle$, r is not a suffix of any instance of some role $r' \in A$.*

Intuitively, when $TC_A(s)$ holds then it means that every role $r' \in A$ has finished its execution in state $s = \langle Sc, IK, CS, tr\rangle$. As we see now, this definition of $TC_A(\cdot)$ allows to specify suitable termination conditions for both secrecy and authentication.

Example 2.3.8. *Consider scenario Sc' from Example 2.5.2. We set $A = \{secret(ks)\}$. Then $TC_A(\cdot)$ holds when role $secret(ks)$ is finished, which means that event $\langle e : x \triangleleft e\rangle$ is executed and hence ks is not secret anymore.*

Example 2.3.9. *Consider scenario Sc'' from Example 2.5.2. We set $A = Sc''$. Then $TC_A(\cdot)$ holds when the responder role is finished, and since no corresponding initiator is present then this means that an authentication attack took place.*

Furthermore, using this termination condition we can detect attacks involving partial runs. Suppose we have a scenario Sc with two roles $r, r' \in Sc$, with $TC_{\{r\}}(\cdot)$. One possible attack over r may require r' to execute partially with r' remaining stuck and thus unfinished (e.g., expecting a message that the intruder cannot generate). Still, since r finishes, Procedure 2.3.4 would find the attack. However, Procedure 2.2.17 would fail to find the attack since only

complete interleavings are considered, in which both r and r' are required to finish their executions.

Attacks regarding partial runs occur commonly in security protocols, as illustrated below in the Woo and Lam mutual authentication protocol [165], shown here as Protocol 2.2. The protocol aims at establishing a session key and provides mutual authentication between two principals a and b, with the help of a trusted server s. a starts by sending her identity and a nonce n_a to b (Message 1). b responds by sending b and n_b (Message 2). Then a sends an encrypted message intended for s, using k_{as}, a long term shared key between a and s (Message 3), which b forwards along with a message of his own using the shared key k_{bs} between b and s (Message 4). s checks that the principal identities and nonces match on both encrypted messages, then selects a new key k_{ab} to be used by a and b. Subsequently, the server creates two encrypted components, one for each of a and b, and returns both of them to b (Message 5). b decrypts his component to obtain the key, and then forwards the other component to a, along with a component encrypted with k_{ab} (Message 6). Finally, a decrypts her component to learn the key, and returns a message to b (Message 7).

Message 1.	$a \rightarrow b : (a, n_a)$
Message 2.	$b \rightarrow a : (b, n_b)$
Message 3.	$a \rightarrow b : \{a, b, n_a, n_b\}_{k_{as}}$
Message 4.	$b \rightarrow s : \{a, b, n_a, n_b\}_{k_{as}}, \{a, b, n_a, n_b\}_{k_{bs}}$
Message 5.	$s \rightarrow b : \{b, n_a, n_b, k_{ab}\}_{k_{as}}, \{a, n_a, n_b, k_{ab}\}_{k_{bs}}$
Message 6.	$b \rightarrow a : \{b, n_a, n_b, k_{ab}\}_{k_{as}}, \{n_a, n_b\}_{k_{ab}}$
Message 7.	$a \rightarrow b : \{n_b\}_{k_{ab}}$

Protocol 2.2: Woo-Lam mutual authentication protocol

Example 2.3.10. *A possible attack over Protocol 2.2 is described by Lowe [132]. The attack uses two sessions, α and β:*

Message α.1.	$e_a \rightarrow b : (a, b)$
Message α.2.	$b \rightarrow e_a : (b, n_b)$
Message α.3.	$e_a \rightarrow b : X$
Message α.4.	$b \rightarrow e_s : X, \{a, b, b, n_b\}_{k_{bs}}$
Message β.1.	$e_a \rightarrow b : (a, n_b)$
Message β.2.	$b \rightarrow e_a : (b, n'_b)$
Message β.3.	$e_a \rightarrow b : Y$
Message β.4.	$b \rightarrow e_s : Y, \{a, b, n_b, n'_b\}_{k_{bs}}$
Message α.5.	$e_s \rightarrow b : Z, \{a, b, n_b, n'_b\}_{k_{bs}}$
Message α.6.	$b \rightarrow e_a : Z, \{(b, n_b)\}_{n'_b}$
Message α.7.	$e_a \rightarrow b : \{n_b\}_{n'_b}$

In message α.1, the intruder starts to impersonate a (denoted as e_a) to attack b, choosing b as the nonce *(This is known as a* type-flaw *confusion, which may occur in implementations where principal identities and nonces have the same bit length, for example.) b responds in Message α.2 by returning b and n_b. Then, the intruder sends any bit-string X (Message α.3).*

b forwards an encrypted component of the appropriate form to s (Message $\alpha.4$). The intruder intercepts this message, and starts a second run, β, again impersonating a. In message $\beta.1$, e_a sends n_b, and b returns n'_b (Message $\beta.2$) and the intruder again responds some bit-string Y (Message $\beta.3$). Then b sends Message $\beta.4$, which is precisely of the form expected by b in message $\alpha.5$. Hence e_s forwards this message to b, and b accepts n'_b as a session key. Finally, b sends an appropriate Message $\alpha.6$ and the intruder returns the component $\{n_b\}_{n'_b}$ (Message $\alpha.7$).

The attack involves a partial run, since session β above is left unfinished and stuck, since the attacker cannot provide the required message $\beta.5$. Still, the attack is successful against session α, and it is found by Procedure 2.3.4 but not by Procedure 2.2.17. However, it is possible to 'patch' Procedure 2.2.17, by modifying Step 3. In a scenario Sc_0, we take a role $r \in Sc_0$ and add a special *send* communication event $\langle x : stop \triangleright y \rangle$ where $stop$ is a special fresh constant. Then we can modify the solving step (i.e. Step 3 in Procedure 2.2.17) so that as soon as $stop$ is seen in a constraint, the solving process is stopped and declared successful. For example, the artificial role for checking secrecy is now $secrecy(X) = \langle e : X \triangleleft e \rangle \langle e : stop \triangleright e \rangle$. This modification allows Procedure 2.2.17 to also cope with partial runs. The modification was later implemented by Millen [147], thus regarding both procedures 2.2.17 and 2.3.4 capable of supporting partial runs.

2.3.2 Comparison and Benchmarks

Let $Sc = \{r_1, \ldots, r_n\}$ be a scenario. Recall that $||Sc$ is the set of all possible interleavings from Sc. The cardinality of $||Sc$, i.e. $| \, ||Sc \, |$, can be calculated by counting the possible permutations of length $\sigma = \sum_{i=1}^{n} |r_i|$, i.e. $\sigma!$, divided by the number of permutations of each sequence of $|r_i|$ to count only one possible ordering. This is known as a multinomial combinatorial [72]:

$$| \, ||Sc \, | = \frac{\sigma!}{\prod_{i=1}^{n} |r_i|!}$$

For $Sc = \{r_1, r_2\}$, we have $|Sc| = n = 2$ and we get $| \, ||Sc \, | = \frac{(|r_1|+|r_2|)!}{|r_1|!|r_2|!} = \binom{|r_1|+|r_2|}{|r_2|}$ as anticipated at the end of Section 2.2.11.

Given a scenario Sc, Procedure 2.2.17 makes $| \, ||Sc \, |$ calls to the reduction procedure $\mathbf{P}$ in the worst case, when no interleaving $\nu \in ||Sc$ is solvable. In general, $| \, ||Sc \, |$ is huge. For example, for the three message protocol used in the benchmarks in the next section, a scenario with four initiators, four responders and one secrecy check generates more than 10^{19} possible interleavings.

We now consider different scenarios in which we compare Procedure 2.2.17 to our Procedure 2.3.4.

A Linear Best Case for Procedure 2.3.4 Let $IK = \{e\}$ be the initial intruder knowledge and let $Sc = \{r_1, \ldots, r_n\}$ be a scenario in which every event in r_i, $i \in [1 \ldots n]$ is a receive event (We do not consider send events so we can avoid the send optimization as described right after introducing Procedure 2.2.17. This still provides a fair comparison since the send optimization applies to both procedures). Also assume that for each $i \in [1 \ldots n]$, $r_i = \langle ev_i \; r'_i \rangle$ s.t. its associated constraint is not solvable (e.g. $ev_i = \langle a :$

# roles	description	Procedure 2.3.4 (time (s))	Procedure 2.2.17 (time (s))
3	1A1B	0.01	0.01
5	2A2B	0.05	0.34
7	3A3B	10	206
8	3A4B	27	2289
8	4A3B	21	9531
9	4A4B	10380	230042

Table 2.2: Comparison between Procedure 2.2.17 and Procedure 2.3.4

$secret \triangleleft b\rangle$ for $secret \in \mathcal{C}$ and $secret \neq e$). This implies that for every $\nu \in ||Sc$, $cs(\nu, IK)$ is unsolvable.

As we already mentioned, Procedure 2.2.17 makes $\mid ||Sc \mid$ calls to $\mathbf{P}$. On the other hand, Procedure 2.3.4 considers just n calls to procedure $\mathbf{P}$ with each corresponding constraint which is unsolvable by assumption. This case shows a significant efficiency gain of Procedure 2.3.4 (which is linear in n) over Procedure 2.2.17 (which is of exponential order of n).

A Worst Case for Procedure 2.3.4 Let $IK = \{e\}$ be the initial intruder knowledge and let ev be the event $ev = \langle e : e \triangleleft e\rangle$. Let $r = \langle ev_1 \dots ev_m\rangle$ be a role consisting of m copies of event ev, so that $ev_i = ev$ for $i \in [1 \dots m]$. Let $Sc' = \{r_1, \dots, r_n\}$ be a scenario in which every event in $r_i = r$, $i \in [1 \dots n-1]$, and $r_n = \langle ev'\rangle$ with $ev' = \langle a : secret \triangleleft b\rangle$.

Here, Procedure 2.2.17 makes $\mid ||Sc' \mid$ calls to $\mathbf{P}$. However, now Procedure 2.3.4 makes $\frac{(|r_1|+\dots+|r_{n-1}|)!}{\prod_{i=1}^{n-1}|r_i|!}$ calls to $\mathbf{P}$, which is better than the number of calls made by Procedure 2.2.17 but still of the same (exponential) order.

Benchmarks As benchmarking protocol, we use the Bilateral Key Exchange with Public Key protocol (BKEPK) (described in the Clark and Jacob library [71]), which appears here as Protocol 2.3. We checked secrecy of nonce n_b.

Message 1.	$b \rightarrow a : b, \{(n_b, b)\}^{\rightarrow}_{pk(a)}$
Message 2.	$a \rightarrow b : \{h(n_b), n_a, a, k_{ab}\}^{\rightarrow}_{pk(b)}$
Message 3.	$b \rightarrow a : \{h(n_a)\}_{k_{ab}}$

Protocol 2.3: Bilateral Key Exchange protocol

Our test-bed is a Pentium Xeon 2.8GHz running Linux 2.6.5-7 and XSB Prolog 2.2 (Tsingtao). The comparison is shown in Table 2.2. We always add one secrecy check role (checking secrecy of nonce n_b). The description field gives an idea of the scenario tested. For example, 2A2B means that we are including 2 A roles and 2 B roles (thus plus the secrecy check makes 5 roles in total).

Further Experiments with protocols from Clark and Jacob library We ran Procedures 2.3.4 and 2.2.17 and confirmed several known attacks over protocols from the Clark and Jacob library [71]. We analyse eleven (representative for their different vulnerabilities) protocols, out of the total forty two (including variants of the same protocols) listed in the library. Table 2.3 lists the performance and the attacks found. The timings are similar for both procedures, since we are checking scenarios that are too small for the optimization of Procedure 2.3.4 over Procedure 2.2.17 to be highlighted.

Protocol	Attack	Proc. 2.3.4	Proc. 2.2.17
Encrypted Key Exchange	Parallel Session	0.01	0.02
ISO sym key 1-pass unilat auth	Replay	0.01	0.02
ISO pub key 2-pass mutual auth	Replay	0.01	0.01
Needham-Schroeder	Man-in-the-middle	0.01	0.07
Needham-Schroeder PK	Man-in-the-middle	0.01	0.02
Needham-Schroeder-Lowe PK	Type Flaw	0.01	0.02
Neuman Stubblebine	Type Flaw	0.02	0.01
Otway-Rees	Type Flaw	0.01	0.01
SPLICE	Replay	0.01	0.02
Woo-Lam Mutual Authentication	Parallel Session	0.02[1]	n/a[2]
Yahalom with Lowe's alteration	Type Flaw	0.02	0.02

Table 2.3: Benchmarking time to find known attacks using Procedure 2.3.4

Implementation We implemented Procedure 2.3.4 using Prolog, modifying Millen and Shmatikov's implementation [149]. The above benchmarks and experiments can be executed from an available online demo website [7] (which also includes more protocol examples not included here).

2.4 Negated Constraint Solving

Up to now we have considered *positive* constraints $m : T$. They are called positive since their solution decides whether there exists a substitution σ s.t. $m\sigma$ can be built from $T\sigma$. In the following we consider *negated* constraints, whose solution decides whether there exists a substitution σ s.t. $m\sigma$ is *not* derivable from $T\sigma$. Negated constraints are going to be useful in the following sections (both Section 2.5 and 2.7), so we introduce and develop negated constraints in this section.

Definition 2.4.1. *A* negated *constraint is denoted by $\neg(m : T)$, where m is a term and T is a set of terms. σ is a* solution *of $\neg(m : T)$ if $m\sigma \notin \mathcal{F}(T\sigma)$, in which case we say that $\neg(m : T)$ is solvable.*

If both m and T are ground, then procedure $\mathbf{P}$ (see Theorem 2.2.15) can be used to solve $\neg(m : T)$:

[1] Originally in [5] the reported time was 30.96. Thanks to Cas Cremers and Ari Saptawijaya for pointing out that the protocol can be modelled more efficiently.

[2] Attack not found due to the requirement of partial runs.

Corollary 2.4.2. *Let m be a ground term and let T be a set of ground terms. Then $\neg(m : T)$ is solvable iff* $\mathbf{P}$ *applied to $m : T$ fails.*

Proof. By Theorem 2.2.15, $\mathbf{P}$ fails iff for all σ, $m\sigma \notin \mathcal{F}(T\sigma)$. Since $m : T$ is ground, we obtain that $\mathbf{P}$ fails iff $m \notin \mathcal{F}(T)$, establishing the property. □

When $\mathbf{P}$ succeeds, we know that there exists one substitution σ s.t. $m\sigma \in \mathcal{F}(T\sigma)$. So if $\mathbf{P}$ fails, we have that *for all* substitutions σ, $m\sigma \notin \mathcal{F}(T\sigma)$; However, what we are trying to establish is whether *there exists one* substitution σ s.t. $m\sigma \notin \mathcal{F}(T\sigma)$. In the case that $m : T$ is ground, then the two cases collapse and hence we can use Corollary 2.4.2. However, when m or T is non ground, we cannot use $\mathbf{P}$ straightforwardly.

Example 2.4.3. *Consider the negated constraint $\neg(\{X\}_Y : \{\{secret_1\}_{secret_2}, e\})$. Applying procedure* $\mathbf{P}$ *to $\{\{X\}_Y : \{\{secret_1\}_{secret_2}, e\}\}$ succeeds, assigning $secret_1$ to X and $secret_2$ to Y. However, the negated constraint is solvable, e.g. by assigning e to X and $secret_2$ to Y.*

Given a state $\langle Sc, IK, CS, tr\rangle$ from Procedure 2.3.4 for a run of input scenario Sc_0 and IK and given a negated constraint $\neg(m : K(tr') \cup IK)$[3] for some term m and some tr' prefix trace of tr, we are interested on finding a solution σ of both CS and $\neg(m : K(tr') \cup IK)$.

We now present a simple strategy to solve $CS \cup \neg(m : K(tr') \cup IK)$ when $K(tr')$ is possibly non ground, although m has to be ground. This solution is enough for our current purposes, as all our security properties are covered; a solution for the general case is still a matter of current research.

Initially, we include a set of fresh constants to the attacker knowledge, one for each variable occurring in the input scenario Sc_0. More formally, we assume that the initial intruder knowledge IK includes a set of constants $C_V = \{c_X \mid X \in \mathcal{V}$ and X occurs in $Sc_0\}$ (recall that $\mathcal{V}$ is the set of variables) thus $C_V \subseteq IK$. C_V contains *intruder generated constants* which do not occur in the input scenario, and hence are never needed to to solve the positive constraint solving phase of CS (the use of these constants is inspired by the work of Kähler and Küsters [115]).

Let σ_V be the substitution that maps every variable X to the corresponding constant c_X. Our solving strategy consists on checking whether σ_V is a solution of $CS \cup \neg(m : K(tr') \cup IK)$. Intuitively, using a fresh constant for each variable gives the best chances for the negated constraint $\neg(m : (K(tr') \cup IK)\sigma_V)$ to hold; any other arbitrary solution σ could assign variables to terms which are more related to each other than the (completely unrelated) fresh constants used by σ_V, thus giving more chances that the attacker can derive m from $(K(tr') \cup IK)\sigma$. This result is formalized below, where $T \not\preceq S$ means that no term $t \in T$ occurs in any term $s \in S$ and $T \not\preceq s$ means that no term $t \in T$ occurs in s (see Appendix A.1 for the formal definition of $\preceq$).

Theorem 2.4.4. *Let $\langle Sc, \mathrm{IK}, CS, tr\rangle$ be a state from Procedure 2.3.4 where for each $X : T_X \in CS$, $C_V \subseteq T_X$ and $C_V \not\preceq T_X \setminus C_V$. Let tr' be a prefix of tr and $\neg(m : K(tr') \cup \mathrm{IK})$ be a negated constraint,* where m is ground and $C_V \not\preceq m$. *Let σ be a substitution with $X : T_X \in CS$, $C_V \not\preceq X\sigma$. Then σ is a solution of both CS and $\neg(m : K(tr') \cup \mathrm{IK})$ iff σ_V is a solution of both CS and $\neg(m : K(tr') \cup \mathrm{IK})$.*

[3] For readability we only consider one negated constraint; the extension to the general case is straightforward.

Proof. See Appendix A.1. □

The proof of this theorem is not easy, and involves e.g. extending the fake function $\mathcal{F}(\cdot)$. The result relies on a series of lemmas, as reported in the Appendix A.1.

Now, the problem of deciding whether a negated constraint $\neg(m : K)$ is solvable (where m is ground) is solved by Theorem 2.4.4, which tells us that $\neg(m : K)$ is solvable iff $m\sigma_V : K\sigma_V$ is solvable, something that can be easily checked using $\mathbf{P}$ as described in Corollary 2.4.2.

2.5 $\mathcal{PS}$-LTL

As we already observed in Section 2.2.3, constraint-based verification procedures are poor when it comes to specifying the properties one wants to check. For example, checking authentication is done in an ad-hoc manner, by e.g. adding a protocol participant but not its corresponding party, and observing whether the participant can still finish its run. This is coarse-grained, and cumbersome to implement (besides this, only a built-in notion of authentication is implemented by Millen [147] in his Prolog implementation). Checking secrecy is also ad-hoc, by adding an artificial protocol role which expects a secret no other participant would send. As we shall see, when secrecy is an atomic predicate being part of a language, more interesting properties can be stated about secrecy.

In this section we propose a new language to specify security properties, based on linear temporal logic (LTL) with pure-past operators, called $\mathcal{PS}$-LTL (pronounced as *pastel*). $\mathcal{PS}$-LTL provides adequate flexibility, allowing one to specify several security properties like authentication ([133, 75]) (including *aliveness*, *weak agreement* and *non-injective agreement*), secrecy (*standard secrecy* [56] and *perfect forward secrecy* [84]) and also data freshness. We also present a (preliminary) study of specification of denial of service (DoS) [142] within our language. While the semantics of $\mathcal{PS}$-LTL is defined as usual on concrete (variable-free) traces, constraint-based protocol verifiers generate symbolic traces which contain *constrained* variables (i.e. variables which may be instantiated only with values the attacker can compute). We present a decision procedure which allows to check a relevant subset (that covers all the properties of interest) of $\mathcal{PS}$-LTL on the symbolic traces produced by constraint solving systems. Moreover, we show the soundness and completeness of our decision procedure.

We start by extending our events, to allow not only communication events but also *status* events, issued by a participant to denote their current state in the execution of their protocol roles.

Definition 2.5.1. *An* event *is one of the following:*

- *A* communication *event as in Definition 2.2.3;*
- *A* status *event $p(d_1, \cdots, d_n)$, where $n \geq 0$, d_i is a term for $i \in [1 \ldots n]$ and p is a function symbol.*

  We consider three different, self-explanatory status events: start, run *and* end *(see Example 2.5.2).*

Typically, status events include information about the state of an principal. Examples include events that indicate that the principal has started an execution, that it is currently running an execution or that it has ended an execution. Moreover, a status event may also include state information, for example to indicate which principal is believed to be the corresponding party in an execution, or what session key is believed to be shared at the end of an execution. Here, *belief* of a fact does not mean that the fact holds in reality; indeed, we are going to look for situations in which such beliefs are indeed false. Other approaches, like the BAN logic [65], instead try to (manually) prove that the beliefs hold.

Example 2.5.2. *Using status events, we can decorate the roles from Example 2.2.6, of Protocol 2.1.*

We show the status events in bold typeface.

$$\begin{aligned}\text{init}(A, B, N_A, K_{lt}, K_{st}) \quad = \quad & \langle\, \mathbf{start}(\mathbf{A}, \mathbf{B}, \mathbf{initiator}) \\ & \langle A : (A, N_A) \triangleright B \rangle \\ & \langle A : \{(N_A, K_{st})\}_{K_{lt}} \triangleleft B \rangle \\ & \mathbf{run}(\mathbf{A}, \mathbf{B}, \mathbf{initiator}, \mathbf{N_A}, \mathbf{K_{lt}}, \mathbf{K_{st}}) \\ & \langle A : \{N_A\}_{K_{st}} \triangleright B \rangle \\ & \mathbf{end}(\mathbf{A}, \mathbf{B}, \mathbf{initiator}, \mathbf{N_A}, \mathbf{K_{lt}}, \mathbf{K_{st}}) \,\rangle \end{aligned}$$

$$\begin{aligned}\text{resp}(A, B, N_A, K_{lt}, K_{st}) \quad = \quad & \langle\, \mathbf{start}(\mathbf{B}, \mathbf{A}, \mathbf{responder}) \\ & \langle B : (A, N_A) \triangleleft A \rangle \\ & \mathbf{run}(\mathbf{B}, \mathbf{A}, \mathbf{responder}, \mathbf{N_A}, \mathbf{K_{lt}}, \mathbf{K_{st}}) \\ & \langle B : \{(N_A, K_{st})\}_{K_{lt}} \triangleright A \rangle \\ & \langle B : \{N_A\}_{K_{st}} \triangleleft A \rangle \\ & \mathbf{end}(\mathbf{B}, \mathbf{A}, \mathbf{responder}, \mathbf{N_A}, \mathbf{K_{lt}}, \mathbf{K_{st}}) \,\rangle \end{aligned}$$

While start and end status events are located in the obvious places, the position where run status events are located is more subtle. Our rule is that run events are located as soon as the protocol role has received every piece of data relevant to the protocol run (this becomes relevant in Section 2.5.2).

2.5.1 Syntax and Semantics

We now introduce our language for writing security properties. Then we provide a semantics, in the form of *concrete* and *symbolic* validity.

Definition 2.5.3. *A $\mathcal{PS}$-LTL formula is defined by the following grammar:*

$$\begin{aligned}\phi ::= \quad & \texttt{true} \mid \texttt{false} \mid p(d_1, \ldots, d_n) \mid \texttt{learn}(m) \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \\ & \mid \texttt{Y}\phi \mid \phi \texttt{S} \phi \mid \exists v.\phi \mid \forall v.\phi\end{aligned}$$

where each d_i ($i \in [1 \ldots n]$) and m is either a variable in $\mathcal{V}$ or a ground term in $\mathcal{T}^+$.

Standard formulas $\texttt{true}$, $\texttt{false}$, $\neg\phi$, $\phi \wedge \phi$, $\phi \vee \phi$ carry the usual meaning. Formula $p(d_1, \ldots, d_n)$ is a status event. $\texttt{learn}(m)$ is a predicate stating that the intruder knows term m (we borrow the name from NPATRL [161]). $\texttt{Y}\phi$ means '*yesterday* ϕ held', while $\phi_1 \texttt{S} \phi_2$ means that 'ϕ_1 held *ever since* a moment in which ϕ_2 held'. When $v \in \mathcal{V}$, we write $\exists v.\phi$ and $\forall v.\phi$ to bind v in ϕ, with the quantifiers carrying the usual meaning with v

ranging over terms. Other operators can be represented using the above defined operators: $\phi_1 \rightarrow \phi_2$ is defined as $\neg\phi_1 \vee \phi_2$; $\mathtt{O}\phi$ (*once* ϕ) is a shorthand for $\mathtt{true}\ \mathtt{S}\ \phi$ and finally $\mathtt{H}\phi$ (*historically* ϕ) is a shorthand for $\neg\mathtt{O}\neg\phi$. For clarity, we impose a precedence hierarchy for operators, where unary operators bind stronger than binary operators. Operators $\mathtt{Y}$, $\mathtt{O}$, and $\mathtt{H}$ bind equally strong and bind stronger than $\neg$. The precedence hierarchy for binary operators is $\mathtt{S} > \wedge > \vee > \rightarrow$, where $op_1 > op_2$ means "op_1 binds stronger than op_2". In the sequel, we assume that $\mathcal{PS}$-LTL formulas are *closed* (i.e. they contain no free variables), and that each variable is quantified at most once; Also, we assume that the variables occurring in a formula ϕ are disjoint from the variables occurring in execution traces tr from the considered system scenario (this can always be achieved by alpha conversion).

Our semantics $\langle tr, IK\rangle \models \phi$ is defined for two different cases: First, we define it when tr is a ground trace, which we call *concrete* validity. Then, we extend the semantics to the general case, in which tr may be symbolic. This establishes *symbolic* validity. Given a trace tr, we recall that tr_i denotes its prefix trace consisting of the first i events (see Section 2.3).

Definition 2.5.4 (Concrete validity)**.** *Let ϕ be a closed $\mathcal{PS}$-LTL formula, tr be a ground trace and IK be an initial intruder knowledge. We then define $\langle tr, IK\rangle \models \phi$ as:*

$$
\begin{array}{lll}
\langle tr, IK\rangle \models \mathtt{true} & & \\
\langle tr, IK\rangle \not\models \mathtt{false} & & \\
\langle tr, IK\rangle \models p(d_1, \ldots, d_n) & \textit{iff} & tr = \langle tr'\ q(e_1, \ldots, e_m)\rangle \\
 & & \quad \textit{and}\ p(d_1, \ldots, d_n) = q(e_1, \ldots, e_m) \\
\langle tr, IK\rangle \models \mathtt{learn}(m) & \textit{iff} & m \in \mathcal{F}(K(tr) \cup IK) \\
\langle tr, IK\rangle \models \neg\varphi & \textit{iff} & \langle tr, IK\rangle \not\models \varphi \\
\langle tr, IK\rangle \models \exists v.\varphi & \textit{iff} & \exists t \in \mathcal{T}^+ : \langle tr, IK\rangle \models \varphi[^t/_v] \\
\langle tr, IK\rangle \models \forall v.\varphi & \textit{iff} & \forall t \in \mathcal{T}^+ : \langle tr, IK\rangle \models \varphi[^t/_v] \\
\langle tr, IK\rangle \models \varphi_1 \wedge \varphi_2 & \textit{iff} & \langle tr, IK\rangle \models \varphi_1\ \textit{and}\ \langle tr, IK\rangle \models \varphi_2 \\
\langle tr, IK\rangle \models \varphi_1 \vee \varphi_2 & \textit{iff} & \langle tr, IK\rangle \models \varphi_1\ \textit{or}\ \langle tr, IK\rangle \models \varphi_2 \\
\langle tr, IK\rangle \models \mathtt{Y}\varphi & \textit{iff} & tr = \langle tr'\ ev\rangle\ \textit{and}\ \langle tr', IK\rangle \models \varphi \\
\langle tr, IK\rangle \models \varphi_1 \mathtt{S} \varphi_2 & \textit{iff} & \exists i \in [0, length(tr)] : (\langle tr_i, IK\rangle \models \varphi_2 \wedge \\
 & & \quad \forall j \in [i+1, length(tr)] : \langle tr_j, IK\rangle \models \varphi_1)
\end{array}
$$

The following proposition shows that the semantics of $\mathtt{O}$ and $\mathtt{H}$ coincide with the intuitive ones:

Proposition 2.5.5. *Let tr be a ground trace, IK the initial intruder knowledge and ϕ a closed $\mathcal{PS}$-LTL formula. Then:*

(i) $\langle tr, IK\rangle \models \mathtt{O}\phi$ *iff* $\exists i \in [0, length(tr)] : \langle tr_i, IK\rangle \models \phi$*, and*

(ii) $\langle tr, IK\rangle \models \mathtt{H}\phi$ *iff* $\forall i \in [0, length(tr)] : \langle tr_i, IK\rangle \models \phi$*.*

Proof. Straightforward from unfolding Definition 2.5.4. □

In fact, we can also state and prove other standard results for LTL and infinite traces, like the tautology $\mathtt{H}\phi \rightarrow \mathtt{O}\phi$. Furthermore, we can state some particular relations of $\mathcal{PS}$-LTL:

Proposition 2.5.6. *For every trace ground tr, IK and message m:*

(i) $\langle tr, IK\rangle \models \mathtt{learn}(m)$ *iff* $\langle tr, IK\rangle \models \mathtt{O}\ \mathtt{learn}(m)$ *iff* $\langle tr, IK\rangle \models \mathtt{learn}(m)\ \mathtt{S}\ \mathtt{learn}(m)$*, and*

(ii) $\langle tr, IK\rangle \models \mathtt{Y}\ \mathtt{learn}(m)$ *implies* $\langle tr, IK\rangle \models \mathtt{learn}(m)$.

Proof. Straightforward from unfolding Definition 2.5.4 and the monotonicity of $\mathcal{F}(\cdot)$, i.e. $\mathcal{F}(K(tr_i) \cup IK) \subseteq \mathcal{F}(K(tr_j) \cup IK)$ for each $i \leq j$. □

The proposition intuitively shows that the intruder never forgets information.

2.5.2 Writing Security Properties with $\mathcal{PS}$-LTL

In this section we show how to specify several security properties in $\mathcal{PS}$-LTL for the BAN Concrete Andrew Secure RPC protocol [65], shown in Example 2.2.6. We report that in addition to this protocol, we also have successfully used our tool for several other protocols (we already analysed over twenty protocols from the Clark Jacob library [71]).

Authentication

First we specify various forms of authentication as defined in [133]. We cover all the variants except injective agreement, which would require counting events in a trace. (In principle, as future work we could extend our system to cover injective agreement, which would result on the ability to detect some replay attacks on which injective agreement is violated but non-injective agreement is satisfied.)

We detail the case of authentication of the initiator to a responder (the converse is similar).

Aliveness The aliveness property is the weakest form of authentication in Lowe's hierarchy:

A protocol guarantees to a responder A aliveness *of another principal* B *if, whenever* A *(acting as responder) completes a run of the protocol, apparently with initiator* B*, then* B *has previously been running the protocol.*

Notice that B may have run the protocol with a principal other than A. The aliveness of principal B to responder A can be specified in $\mathcal{PS}$-LTL as follows:

$$\forall A, B, D1, D2, D3.\ \exists A', R'.\ end(A, B, responder, D1, D2, D3) \rightarrow \mathtt{O}\ start(B, A', R')$$

The aliveness property is violated for our protocol of Example 2.2.6 on a scenario containing at least two protocol sessions (i.e. two initiators and two responders). We ran our tool on the scenario

$$\{init(a, b, n_a, k_{lt}, K_{st_1}), init(b, a, n_b, k_{lt}, K_{st_2}), resp(b, a, N_B, k_{lt}, k_{st_2})\} \tag{2.2}$$

and found a similar attack to the one found by Lowe [132]. Furthermore, we also checked the aliveness property for Lowe's fixed version of BAN concrete Andrew Secure RPC protocol [132], and found no attacks thus confirming the validity of Lowe's fix.

Weak Agreement Weak agreement is slightly stronger than aliveness:

A protocol guarantees to a responder A weak agreement *with another principal B if, whenever A (acting as responder) completes a run of the protocol, apparently with initiator B, then B has previously been running the protocol,* ***apparently with*** *A.*

For this property, B may not necessarily have been acting as initiator. The weak agreement property can be expressed in $\mathcal{PS}$-LTL as follows:

$$\forall A, B, D1, D2, D3.\ \exists R'.\ end(A, B, responder, D1, D2, D3) \rightarrow \mathtt{O}\ start(B, A, R')$$

Since weak agreement is stronger than aliveness, the attack mentioned above also applies (e.g. can be found using scenario (2.2)).

Non-injective Agreement Non-injective agreement is slightly stronger than weak agreement:

A protocol guarantees to a responder A non-injective agreement *with another principal B on a set of data items $\mathbf{D}$ if, whenever A (acting as responder) completes a run of the protocol, apparently with initiator B, then B has previously been running the protocol, apparently with A, and B was acting as initiator in his run, and the two principals agreed on the data values corresponding to all the variables in $\mathbf{D}$.*

The property is formalized in $\mathcal{PS}$-LTL as follows (our tool also discovers the attack to this property, using e.g. scenario (2.2)):

$$\forall A, B, D1, D2, D3. end(A, B, responder, D1, D2, D3) \rightarrow \\ \mathtt{O}\ run(B, A, initiator, D1, D2, D3)$$

Secrecy

We now turn to study secrecy. In particular, we focus on two particular notions of secrecy, viz. standard secrecy and perfect forward secrecy.

Standard secrecy We define first the simple case of standard secrecy, which is the inability of an attacker to obtain the value of the secret [56]. Recall scenario Sc' of Example 2.5.2. The secrecy of the session key k_{st}, once the initiator a started a protocol run with the responder b, can be checked by the $\mathcal{PS}$-LTL formula $\neg\mathtt{learn}(k_{st})$. We checked this property with our tool and found no secrecy attack on Sc'.

Perfect Forward Secrecy We now consider the more challenging case of perfect forward secrecy (PFS). We follow the definition given by Diffie, et.al [84]:

An (authenticated key exchange) protocol provides perfect forward secrecy if disclosure of long-term secret keying material does not compromise the secrecy of the exchanged keys from earlier runs.

In Diffie et.al [84], the proposed Authenticated Diffie-Hellman key exchange protocol preserves PFS, since long term keys are only used to sign messages and are never related to the session key derivation. This is not the case for the RPC Andrew protocol and its variants,

since the short term key is directly encrypted by the long term key (see below for an example of a secure protocol).

We model the disclosure of a long term secret, like k_{lt} in the RPC Andrew protocol, by providing an additional protocol role which contains only one send event that leaks k_{lt} to the intruder. A violation of PFS happens in a situation in which (i) a protocol run has been executed in the past and the short term session key used in the run (k_{st} in our example) remained secret; (ii) the long term key k_{lt} is learnt by the intruder; and finally (iii) the short term key k_{st} that was secret during the run is now compromised, i.e. learnt by the attacker. We express PFS in $\mathcal{PS}$-LTL as follows:

$$\forall A, B, N, K_{lt}.\ \texttt{learn}(K_{lt}) \wedge \\ \texttt{Y}(\texttt{O}\ (end(B, A, responder, N, K_{lt}, k_{st}) \wedge \texttt{H}\ \neg\texttt{learn}(k_{st}))) \rightarrow \texttt{H}\ \neg\texttt{learn}(\texttt{k}_{\texttt{st}})$$

The negation of this formula captures our desired violation. Thanks to Proposition 2.5.6, we can rewrite the property more efficiently:

$$\forall A, B, N, K_{lt}.\ \texttt{learn}(K_{lt}) \wedge \\ \texttt{Y}(\texttt{O}\ (end(B, A, responder, N, K_{lt}, k_{st}) \wedge\ \neg\texttt{learn}(k_{st}))) \rightarrow\ \neg\texttt{learn}(k_{st})$$

Our tool finds the straightforward attack quickly in the following appropriate scenario:

$$\{init(a, b, na, k_{lt}, K_{st}), resp(a, b, N_A, k_{lt}, k_{st}), \langle\ \langle leaker : k_{lt} \triangleright e \rangle\ \rangle\}$$

A secure protocol w.r.t. PFS Consider the following protocol shown as Protocol 2.4.

Message 1.	$a \rightarrow b : \{(pk(r_a), b)\}_{k_{lt}}$
Message 2.	$b \rightarrow a : (\{k_{st}\}^{\rightarrow}_{pk(r_a)}, \{(h(k_{st}), a)\}_{k_{lt}})$

Protocol 2.4: A protocol satisfying perfect forward secrecy (PFS)

Protocol 2.4 is a modified version of the protocol due to Boyd and Mathuria [61] that aims to meet perfect forward secrecy.[4] Agents a and b share a long term key k_{lt}. Agent a generates a *fresh* asymmetric key pair in every protocol run (indicated by a *fresh* r_a) and *discards* it after the run is completed (thus the key pair is *ephemeral*). In the first message, a encrypts the public part $pk(r_a)$ together with b's identity with k_{lt} and sends it to b. Upon receipt, b obtains $pk(r_a)$ and then replies by encrypting the freshly generated short term session key k_{st} with $pk(r_a)$ and encrypting the hash of k_{st} and a's identity with k_{lt}.

Although the disclosure of k_{lt} after a completed protocol run allows an attacker to impersonate a or b in the subsequent runs, it does not provide the attacker with the ability to recover the session key k_{st} from the previous run. This session key can only be recovered using a's private key from the completed run (which has been discarded as soon as the protocol run completes).

In a scenario in which the short term key chosen by the responder is k_{st}, we can check PFS as follows:

[4] Note that the protocol provides only one-way authentication, viz. the authentication of b to a.

$$\forall A, B, N, K_{lt}.\ \texttt{learn}(K_{lt}) \wedge \\ \texttt{Y}(\texttt{O}\ (end(A, B, initiator, N, K_{lt}, k_{st}) \wedge\ \neg\texttt{learn}(k_{st}))) \rightarrow\ \neg\texttt{learn}(k_{st})$$

Our tool found no attack in appropriate scenarios, thus confirming the analysis of a similar protocol by Delicata and Schneider [81].

Data Freshness

We state the data freshness property as follows:

Data D is fresh whenever a principal A (either as an initiator or as a responder) never completes a run with another principal agreeing on D, if in the past A (either as an initiator or as a responder) has already completed a protocol run with another principal agreeing on the same data D.

The freshness $\mathcal{PS}$-LTL property of a short term session key K for the protocol of Example 2.2.6 is:

$$\forall A, B_1, R_1, N_1, K_1, K, B_2, R_2, N_2, K_2. \\ \texttt{Y}(\texttt{O}\ end(A, B_1, R_1, N_1, K_1, K)) \rightarrow \neg end(A, B_2, R_2, N_2, K_2, K)$$

We run our tool to check the freshness of the session key k_{st} on an appropriate scenario, and obtained an attack similar to the previous aliveness attack. In this attack, the session key k_{st} is used twice, i.e. when a was acting as an initiator in one session and as a responder in the other session. Thus, it violates the freshness of k_{st}. For Lowe's fixed version of BAN concrete Andrew Secure RPC protocol [132], no attack was found as expected.

Towards analysing Denial of Service attacks

We now sketch how to specify a property which can unveil potential vulnerabilities of denial of service (DoS) attacks [142] In the protocol of Example 2.2.6, the first message (a, n_a) can be generated cheaply by anyone (not necessarily by a). On the other hand, upon receiving (a, n_a), b commits to perform several expensive operations (e.g., generating the session key k_{st} and allocating the state for the running session). This is a typical situation in which one can mount a DoS attack against b. To do this, an attacker simply needs to start several sessions which cause b to reach its *run* event: i.e., the point at which b commits to carrying out a computationally expensive part of the protocol. The attack is possible because starting such a session is computationally unexpensive, and many such sessions can be launched in a short time, thereby exhausting b's resources.

To spot this vulnerability using $\mathcal{PS}$-LTL we use the following formula, specifying that if a responder b reaches the computationally expensive part of a session (indicated by b emitting the *run* event), then the honest initiator a has once started a session with b:

$$\forall A, B, D1, D2, D3.\ run(A, B, responder, D1, D2, D3) \rightarrow \texttt{O}\ start(B, A, initiator).$$

A violation to this formula would indicate that an attacker could easily impersonate a to mount a DoS. Note that the attack does not require that b finishes his execution (and reaching the *end* event): thus this property differs from the authentication ones reported above.

We successfully used our tool to check this formula on a single session and obtain a trace that thus indicates the vulnerability to DoS attacks. When we modified the protocol by encrypting the first message using the long term key k_{lt} (i.e. Message 1 becomes $\{(a, n_a)\}_{k_{lt}}$) the trace indicating the weakness disappeared. Intuitively, this extra encryption allows only *honest* participants, who share k_{lt}, to generate the first message. (Still, the attacker could replay Message 1 and continue mounting the DoS attack; this would however be detected by an injective analysis, as discussed in the beginning of Section 2.5.2.)

2.5.3 Deciding $\mathcal{PS}$-LTL Validity in Constraint Solving

The constraint-based Procedure 2.3.4 presented in Section 2.2.4 outputs *symbolic* traces containing constrained variables. In this section we show how to decide validity of a $\mathcal{PS}$-LTL formula against such a symbolic trace.

Since in the previous section we defined validity (called concrete) only w.r.t. ground traces, the first thing we need to do is to extend the notion of validity for symbolic traces.

Definition 2.5.7 (Symbolic validity)**.** *Given a trace tr derived from a system scenario Sc and the initial intruder knowledge I*K*, we say that $\langle tr, I\text{K}\rangle \models \phi$ when for every valid instance tr' of tr, $\langle tr', I\text{K}\rangle \models \phi$.*

Let φ be a closed $\mathcal{PS}$-LTL formula representing a security property. We let $A_\varphi = \neg\varphi$ be its corresponding *attack* property. Given a symbolic trace tr and the initial intruder knowledge *I*K, we now define a procedure $\mathbf{D}$ that tries to find a valid ground instance tr' of tr s.t. $\langle tr', IK\rangle \models A_\varphi$. If $\mathbf{D}$ succeeds, tr' represents a violation of φ (hence an attack), since $\langle tr', IK\rangle \models A_\varphi$ iff $\langle tr', IK\rangle \not\models \varphi$, and thus $\langle tr, IK\rangle \not\models \varphi$. On the other hand, if $\mathbf{D}$ fails, then we know that there is no tr' s.t. $\langle tr', IK\rangle \models A_\varphi$. In other words, for every ground instance tr' of tr, $\langle tr', IK\rangle \models \varphi$, i.e. $\langle tr, IK\rangle \models \varphi$. Thus $\mathbf{D}$ decides symbolic validity.

Let φ be a closed $\mathcal{PS}$-LTL formula representing a security property. We let $A_\varphi = \neg\varphi$ be its corresponding *attack* property. Given a symbolic trace tr and IK, we define a procedure $\mathbf{D}$ (in Section 2.5.3) that tries to find a ground instance tr' of tr s.t. $\langle tr', IK\rangle \models A_\varphi$. If $\mathbf{D}$ succeeds, tr' represents a violation of φ (hence an attack), since $\langle tr', IK\rangle \models A_\varphi$ iff $\langle tr', IK\rangle \not\models \varphi$, and thus $\langle tr, IK\rangle \not\models \varphi$. On the other hand, if $\mathbf{D}$ fails, then we know that there is no tr' s.t. $\langle tr', IK\rangle \models A_\varphi$. In other words, for every ground instance tr' of tr, $\langle tr', IK\rangle \models \varphi$, i.e., $\langle tr, IK\rangle \models \varphi$. Thus $\mathbf{D}$ decides symbolic validity.

Our approach consists of two stages. We first translate a closed $\mathcal{PS}$-LTL formula ϕ into a (shown equivalent) *elementary formula* EF, as defined below, using the transformation $\mathbf{T}$ developed next in Section 2.5.3. Then, we input the translated formula to the decision procedure $\mathbf{D}$, presented in Section 2.5.3.

Definition 2.5.8. *Elementary formulas EF (ranged over by π) are defined by the grammar:*

$$\pi ::= \texttt{true} \mid \texttt{false} \mid t_1 = t_2 \mid m : K \mid \neg\pi \mid \pi \wedge \pi \mid \pi \vee \pi \mid \exists v.\pi \mid \forall v.\pi$$

Here each t_1, t_2 and m is either a variable or a ground term, K is a set of terms and v is a variable.

Let π be an EF formula. We define its *left* free variables $free_l(\pi)$ and its *right* free variables $free_r(\pi)$, as follows:

$$
\begin{array}{lcl}
free_l(\texttt{true}) = free_l(\texttt{false}) & = & \emptyset \\
free_l(t_1 = t_2) & = & var(t_1) \\
free_l(m : K) & = & var(m) \\
free_l(\neg\pi) & = & free_l(\pi) \\
free_l(\pi_1 \wedge \pi_2) = free_l(\pi_1 \vee \pi_2) & = & free_l(\pi_1) \cup free_l(\pi_2) \\
free_l(\exists v.\pi) = free_l(\forall v.\pi) & = & free_l(\pi) \setminus \{v\}
\end{array}
$$

$free_r(\pi)$ is similar, but defined with: $free_r(t_1 = t_2) = var(t_2)$ and $free_r(m : K) = var(K)$. We now give a semantics of an EF formula π w.r.t. a ground substitution σ.

Definition 2.5.9. *Let π be an EF formula and σ a ground substitution s.t. $free_l(\pi) = \emptyset$ and $free_r(\pi) = dom(\sigma)$. Then $\sigma \models' \pi$ is defined by:*

$$
\begin{array}{lll}
\sigma \models' \texttt{true} & & \\
\sigma \not\models' \texttt{false} & & \\
\sigma \models' t_1 = t_2 & \textit{iff} & t_1 = t_2\sigma \\
\sigma \models' m : K & \textit{iff} & m \in \mathcal{F}(K\sigma) \\
\sigma \models' \neg\pi & \textit{iff} & \sigma \not\models' \pi \\
\sigma \models' \pi_1 \wedge \pi_2 & \textit{iff} & \sigma \models' \pi_1 \textit{ and } \sigma \models' \pi_2 \\
\sigma \models' \pi_1 \vee \pi_2 & \textit{iff} & \sigma \models' \pi_1 \textit{ or } \sigma \models' \pi_2 \\
\sigma \models' \exists v.\pi & \textit{iff} & \exists t \in \mathcal{T}^+ : \sigma \models' \pi[^t/_v] \\
\sigma \models' \forall v.\pi & \textit{iff} & \forall t \in \mathcal{T}^+ : \sigma \models' \pi[^t/_v]
\end{array}
$$

First Stage: Translating $\mathcal{PS}$-LTL

We define a translation $\mathbf{T}(\phi, tr, IK)$ from a $\mathcal{PS}$-LTL formula ϕ, a trace tr and an initial intruder knowledge IK into an EF formula:

Definition 2.5.10. *Let ϕ be a $\mathcal{PS}$-LTL formula, tr an execution trace and IK an initial intruder knowledge. Then $\mathbf{T}(\phi, tr, IK)$ is the EF formula resulting from applying the following three steps:*

1. *First, we repeatedly apply transformation $\lfloor\cdot\rfloor\cdot$, defined below, until none of the rules can be applied:*

$$
\begin{array}{rcl}
\lfloor\exists v.\phi\rfloor tr & \Rightarrow & \exists v.\lfloor\phi\rfloor tr \\
\lfloor\forall v.\phi\rfloor tr & \Rightarrow & \forall v.\lfloor\phi\rfloor tr \\
\lfloor\neg\phi\rfloor tr & \Rightarrow & \neg\lfloor\phi\rfloor tr \\
\lfloor\phi_1 \wedge \phi_2\rfloor tr & \Rightarrow & \lfloor\phi_1\rfloor tr \ \wedge \lfloor\phi_2\rfloor tr \\
\lfloor\phi_1 \vee \phi_2\rfloor tr & \Rightarrow & \lfloor\phi_1\rfloor tr \ \vee \lfloor\phi_2\rfloor tr \\
\lfloor \mathtt{Y}\phi\rfloor\langle\rangle & \Rightarrow & \texttt{false} \\
\lfloor \mathtt{Y}\phi\rfloor\langle tr\ e\rangle & \Rightarrow & \lfloor\phi\rfloor tr \\
\lfloor\phi_1 \mathtt{S} \phi_2\rfloor\langle\rangle & \Rightarrow & \lfloor\phi_2\rfloor\langle\rangle \\
\lfloor\phi_1 \mathtt{S} \phi_2\rfloor\langle tr\ e\rangle & \Rightarrow & \lfloor\phi_2\rfloor\langle tr\ e\rangle \vee \\
 & & (\lfloor\phi_1\rfloor\langle tr\ e\rangle \ \wedge \lfloor\phi_1 \mathtt{S} \phi_2\rfloor tr)
\end{array}
$$

$$
\begin{array}{rcl}
\lfloor \texttt{true} \rfloor tr & \Rightarrow & \texttt{true} \\
\lfloor \texttt{false} \rfloor tr & \Rightarrow & \texttt{false} \\
\lfloor \texttt{learn}(m) \rfloor tr & \Rightarrow & m : (K(tr) \cup IK) \\
\lfloor p(d_1, \ldots, d_n) \rfloor \langle\rangle & \Rightarrow & \texttt{false} \\
\lfloor p(d_1, \ldots, d_n) \rfloor \langle tr\ q(e_1, \ldots, e_m) \rangle & \Rightarrow & \texttt{false} \textit{ if } p \neq q \textit{ or } n \neq m \\
\lfloor p(d_1, \ldots, d_n) \rfloor \langle tr\ p(e_1, \ldots, e_n) \rangle & \Rightarrow & d_1 = e_1 \wedge \cdots \wedge d_n = e_n
\end{array}
$$

(Note interestingly how a `learn` *translates directly into a constraint. Also, notice that in each equality $d_i = e_i$, $var(d_i) \cap var(e_i) = \emptyset$, as we require that variables from the formula and from the trace do not clash (see last paragraph of Section 2.5.1).)*

2. *Repeatedly rewrite atoms $\neg\neg\phi$ to ϕ, and move $\neg$ inside conjunctions and disjunctions using DeMorgan distributive laws.*

3. *Move universal quantifiers ($\forall$) as far as possible to the right, and simplify universally quantified formulas over (possibly negated) equalities and constraints, according to the following rules:*

$$
\begin{array}{rcl}
\forall v.(\phi_1 \wedge \phi_2) & \Rightarrow & \forall v.\phi_1 \wedge \forall v.\phi_2 \\
\forall v.(\phi_1 \vee \phi_2) & \Rightarrow & \forall v.\phi_1 \vee \forall v.\phi_2 \textit{ if } v \textit{ is not free in } \phi_1 \\
 & & \qquad \textit{or } v \textit{ is not free in } \phi_2 \\
\forall v.\phi & \Rightarrow & \phi \textit{ if } v \textit{ is not free in } \phi \\
\forall v.(v : K) & \Rightarrow & \texttt{false} \textit{ (where } K \textit{ is a term set)} \\
\forall v.\neg(v : K) & \Rightarrow & \texttt{false} \textit{ (where } K \textit{ is a term set)} \\
\forall v.\neg(v = t) & \Rightarrow & \texttt{false} \\
\forall v.(v = t) & \Rightarrow & \texttt{false}
\end{array}
$$

(In the last two rules, $t \neq v$ since we require that $var(v) = \{v\} \cap var(t) = \emptyset$.)

It can be shown that the transformation $\mathbf{T}$ terminates and is confluent, given a finite trace, although we do not prove that here. The last step (3) removes cases which are known not to hold (their correctness is proven in Lemma 2.5.12). Transformation $\mathbf{T}$ provides the necessary input to the decision algorithm $\mathbf{D}$ of the next section, to decide validity of the original formula.

We call an EF formula *existential* if it is of the form $\exists v_1 \ldots \exists v_n.\varphi$, and φ does not contain any quantifiers ($\forall$ nor $\exists$). In addition, we say that an EF formula ϕ is *negation ground* if every occurrence of a negated constraint $\neg(m : T)$ in ϕ satisfies that m is ground (we will need this requirement to be able to decide negated constraints, see Section 2.4).

We now define the subset Φ of $\mathcal{PS}$-LTL over which we are going to decide symbolic validity:

Definition 2.5.11. Φ *is the set of* well-behaving *EF formulas:*

$$\Phi \stackrel{\triangle}{=} \{\ \phi \mid \phi \textit{ closed and } \mathbf{T}(\phi, tr, IK) \textit{ is existential and negation ground for all } tr, IK\ \}$$

Φ is expressive enough for several interesting security properties. In particular, every property φ considered in Section 2.5.2 satisfies $A_\varphi \in \Phi$. Examples of ϕ that are not well-behaving are:

- $\phi_1 = \forall x.\exists y.(run(x) \wedge run(y))$, since for $tr_1 = \langle\langle run(z)\rangle\rangle$ we have $\mathbf{T}(\phi_1, tr_1, IK) = \forall x.\exists y.(x = z) \wedge (y = x)$, and
- $\phi_2 = \exists x.\forall y.(run(x, y) \vee run(y, x))$, since for trace $tr_2 = \langle\langle run(z, w)\rangle\rangle$ we have that $\mathbf{T}(\phi_2, tr_2, IK) = \exists x.\forall y.((x = z \wedge y = w) \vee (y = z \wedge x = w))$ for any IK.

The following lemma states that the translation $\mathbf{T}$ is correct, i.e. that it preserves the semantics of $\mathcal{PS}$-LTL w.r.t. semantics of EF.

Lemma 2.5.12. *Let ϕ be a closed $\mathcal{PS}$-LTL formula, tr be a trace and IK be an initial intruder knowledge, and let σ be a ground substitution such that $var(tr) \subseteq dom(\sigma)$. Then $\langle tr\sigma, IK\rangle \models \phi$ iff $\sigma \models' \mathbf{T}(\phi, tr, IK)$.*

Proof. See Appendix A.1. □

Second Stage: Deciding Validity

Given a well-behaving EF formula $\pi = \exists v_1 \ldots \exists v_n.\varphi$, we transform φ into its *disjunctive normal form* $\varphi = \bigvee_j \psi_j$, with $\psi_j = \bigwedge_i \pi_{j,i}$. Given also a simple constraint set CS, the procedure $\mathbf{D}(\pi, CS)$ we are about to define either fails and returns `false` or succeeds and returns a substitution σ that makes φ true. For simplicity, in the sequel we assume that each ψ_j contains just one *positive equality* $L_j = R_j$ (where L_j and R_j denote the left and right term in the equality, respectively), one *negated equality* $\neg(L_j^\neg = R_j^\neg)$ (where $L_j^\neg$ and $R_j^\neg$ denote the left and right term in the negated equality, respectively), one *positive constraint* $m_j : K_j$ and one *negated constraint* $\neg(m_j^\neg : K_j^\neg)$. The generalization to the case with several atomic formulas and with (possibly negated) `true` and `false` atoms is straightforward.

Procedure 2.5.13. *Let $\varphi = \bigvee_j \psi_j$, with $\psi_j = (L_j = R_j) \wedge \neg(L_j^\neg = R_j^\neg) \wedge (m_j : K_j) \wedge \neg(m_j^\neg : K_j^\neg)$. Let CS be a simple constraint set. Procedure $\mathbf{D}$ succeeds if all the following steps succeed, in which case it returns $\sigma = \rho\rho_k\sigma_V$ where ρ is given by Step 2, ρ_k is given by Step 3 and σ_V is the substitution described in Section 2.4.*

1. *Pick a disjunct ψ_j while possible, otherwise exit and return* false.
2. Solve Positive Equality: *Take a relevant most general unifier ρ of L_j and R_j such that $dom(\rho) \subseteq var(L_j) \cup var(R_j)$, i.e. $L_j\rho = R_j\rho$ (If no mgu exists, go back to Step 1).*
3. Solve Positive Constraint: *Apply* $\mathbf{P}$ *to $(CS \cup \{m_j : K_j\})\rho$. Let $\rho_1, \ldots, \rho_l$ be the partial solutions.*
4. *Pick ρ_k, $k \in [1 \ldots l]$, while possible, otherwise go back to Step 1.*
5. Solve Negated Constraint: *Apply* $\mathbf{P}$ *to $(CS \cup \{m_j : K_j, m_j^\neg : K_j^\neg\})\rho\rho_k\sigma_V$ (where σ_V is the substitution given in Section 2.4). If it is solvable, go back to Step 4.*
6. Solve Negated Equality: *Check that $L_j^\neg\rho\rho_k\sigma_V$ and $R_j^\neg\rho\rho_k\sigma_V$ differ syntactically.*

Step 2 tries to solve the positive equality, finding a suitable unifier ρ. (We need a unifier and not a matching for the general case of many equalities). In case ρ is not found, then the disjunct does not hold , so we try a different one going back to Step 1. Similarly, Step 3 solves

the positive constraint. Step 5 checks that both ρ, ρ_k and σ_V cause the negated constraint to hold (this is based on Theorem 2.4.4). Finally, Step 6 checks that the negated equalities hold. Since we consider (i) relevant unifiers for Step 2, of which there are only finitely many, (ii) $\mathbf{P}$ only outputs a finite number of solutions for Step 3 and 5, and (iii) we only need to perform a syntactic check for Step 6, then we can deduce that procedure $\mathbf{D}$ terminates. The correctness of $\mathbf{D}$ is more challenging to establish.

Lemma 2.5.14. *Let ϕ be a closed $\mathcal{PS}$-LTL formula, and $\langle Sc, IK, CS, tr\rangle$ a state from Procedure 2.3.4. Assume that $\pi = \mathbf{T}(\phi, tr, IK)$ is a well-behaving EF formula, $\pi = \exists v_1 \ldots \exists v_n.\psi$ with $\psi = \bigvee_j \psi_j$ and $\psi_j = \bigwedge_i \pi_{j,i}$. Then:*

1. *$\mathbf{D}(\psi, CS)$ succeeds and returns a substitution σ implies that $\sigma \models' \pi$, with $tr\sigma$ valid w.r.t. IK; and*
2. *$\sigma \models' \pi$, with $tr\sigma$ valid w.r.t. IK implies that there exists a substitution γ s.t. $\mathbf{D}(\psi, CS)$ succeeds and returns γ.*

Proof. See Appendix A.1. □

Now we are ready to formulate the main result of this section, which states that applying the transformation $\mathbf{T}$ of Definition 2.5.10 and $\mathbf{D}$ defined in Procedure 2.5.13 is both sound and complete.

Theorem 2.5.15. *Let Sc_0 be a system scenario, IK be an initial intruder knowledge, ϕ be a closed $\mathcal{PS}$-LTL formula representing a security property, and let $A_\phi = \neg\phi$. Let $\langle Sc, IK, CS, tr\rangle$ be a state from Procedure 2.3.4. Assume $\pi = \mathbf{T}(A_\phi, tr, IK)$ is well-behaving and in disjunctive normal form, $\pi = \exists v_1 \ldots \exists v_n. \bigvee_j \psi_j$, with $\psi_j = \bigwedge_i \pi_{j,i}$. Then $\mathbf{D}(\pi, CS)$ fails iff $\langle tr, IK\rangle \models \phi$.*

Proof. $\mathbf{D}(\pi, CS)$ fails iff, by Lemma 2.5.14, $\forall\sigma : \sigma \not\models' \mathbf{T}(A_\phi, tr, IK)$. By Lemma 2.5.12, $\langle tr\sigma, IK\rangle \not\models A_\phi$. By definition, this is equivalent to $\langle tr\sigma, IK\rangle \models \phi$. So, we obtained that $\forall\sigma : \langle tr\sigma, IK\rangle \models \phi$, which by Definition 2.5.7 of symbolic validity is $\langle tr, IK\rangle \models \phi$. □

Integrating $\mathcal{PS}$-LTL to Constraint Solving We integrate the checking of a closed $\mathcal{PS}$-LTL formula ϕ (representing a security property) into the constraint-based protocol analysis approach described in Procedure 2.3.4. We need to first consider also status events (besides send and receive events). This can be handled easily since status events behave similarly to send events, so we only need to modify slightly step 1.(1a) of Procedure 2.3.4 (see Section 2.8). The next step in the integration of $\mathcal{PS}$-LTL consists in defining the appropriate termination condition.

Definition 2.5.16. *Given a state $\langle Sc, IK, CS, tr\rangle$, we define $TC_\phi(\langle Sc, IK, CS, tr\rangle)$ to be true when $\mathbf{D}(\pi, CS) = true$, for $\pi = \mathbf{T}(\neg\phi, tr, IK)$ well-behaving.*

The termination condition essentially checks whether tr in the current execution state can be instantiated to provide a solution of $\neg\phi$, that is, to *falsify* ϕ: by Theorem 2.5.15, we know that $\mathbf{D}(\mathbf{T}(\neg\phi, tr, IK), CS)$ holds iff $\langle tr, IK\rangle \not\models \phi$. In that case, the procedure terminates and outputs the trace tr that shows an attack. Otherwise, the procedure proceeds until an attack or no attack is found. Since $\mathbf{T}$ and $\mathbf{D}$ terminate, and Procedure 2.3.4 terminates, checking $TC_\phi(\cdot)$ as given in Definition 2.5.16 also terminates.

Implementation Procedure 2.5.13 and translation of Procedure 2.5.10 have been implemented in Prolog, and attached to the implementation of Procedure 2.3.4. See [7] for an online demo.

2.6 Adding a *Check* Event

Consider again the basic Procedure 2.3.4 (without $\mathcal{PS}$-LTL). Principals are allowed to perform only send and receive operations, that is communication events. In this section we show one example in which this is not enough when *specifying* security protocols, and we need an extra event. Consider now Protocol 2.5 (inspired by Zhou-Gollman [168]).

Message 1.	$a \rightarrow b : (l, c)$
Message 2.	$b \rightarrow a : sig_{pk(b)}(c)$
Message 3.	$a \rightarrow b : (sig_{pk(a)}(k), k)$
Message 4.	$b \rightarrow a : sig_{pk(b)}(l, c, k)$

Protocol 2.5: Derived Zhou-Gollman protocol

This protocol aims at exchanging some message m in a fair manner, where b gets m iff a gets *evidence* that b agreed to receive m (here we do not pursue verifying fair exchange, but merely illustrate the need of an extra operation to be able to specify this kind of protocols appropriately).

In Message 1, a starts the session by sending to b the hash $l = h(m)$ and a ciphered message $c = \{m\}_k$. At this point b cannot check that indeed l is the hash of some message m nor c is the encryption of m, since b does not have k. This is intended: when b provides its signature, a will provide the decryption key k. Thus, b replies with his signature of c in Message 2. In Message 3, a sends the signed symmetric key k to b (we need to send k also in plain since our signature operator is not invertible by the intruder). Now, *and not before*, b can decrypt c and check that $l = h(m)$. If the check does not succeed then b stops the session, otherwise he continues by sending Message 4.

A Vulnerability The above protocol presents the following vulnerability: the attacker can impersonate an honest principal (a, the initiator), and convince b that he is communicating with a (so that b will believe its exchanging m with a while in reality a is not doing so). The attack comes to light by instantiating the protocol twice (sessions α and β) as follows:

Message $\alpha.1$.	$e_a \rightarrow b : (h(m), \{m\}_k)$
Message $\alpha.2$.	$b \rightarrow e_a : sig_{pk(b)}(\{m\}_k)$
Message $\beta.1$.	$e \rightarrow a : (X, k)$
Message $\beta.2$.	$a \rightarrow e : sig_{pk(a)}(k)$
Message $\alpha.3$.	$e_a \rightarrow b : (sig_{pk(a)}(k), k)$
Message $\alpha.4$.	$b \rightarrow e_a : sig_{pk(b)}(h(m), \{m\}_k, k)$

First the intruder starts to impersonate a in Message $\alpha.1$ (e is the intruder impersonating the honest principal a). He sends $l = h(m)$ and $c = \{m\}_k$, for some m and k. b then

replies signing of $\{m\}_k$. Now the intruder must provide the message $sig_{pk(a)}(k)$. For this, he initiates a new session β with a, and composes a new message with anything as l, represented as X (it does not matter what value it takes) but with k as c (Message $\beta.1$). Then a replies by signing k, and this is exactly what the intruder needs (Message $\beta.2$). He just creates a new message concatenating this message and k, and sends it to b (Message $\alpha.3$). b receives the message, performs successfully the check and finishes his session by sending the last message $\alpha.4$. Note also that the β session is partial.

The check of $l = h(m)$ is difficult to implement using just communication events, since there is no way to *equate* l with $h(m)$. We can try to input directly at Message 1, $h(m)$ instead of l and $\{m\}_k$ instead of c, as shown in Protocol 2.6.

Message 1.	$a \rightarrow b : (h(m), \{m\}_k)$
Message 2.	$b \rightarrow a : sig_{pk(b)}(\{m\}_k)$
Message 3.	$a \rightarrow b : (sig_{pk(a)}(k), k)$
Message 4.	$b \rightarrow a : sig_{pk(b)}((h(m), \{m\}_k, k))$

Protocol 2.6: Derived Zhou-Gollman protocol without c nor l

However, even though Protocol 2.6 seems equivalent to Protocol 2.5, it is not so: here b can find out *already at step 1* if l does not comply with the definition (thus not continuing its execution), while in reality (and in the original definition) this can happen only after step 3. In fact, Protocol 2.6 is *not* vulnerable against the above attack.

Explicit Checks To implement the protocol correctly, we need to introduce a new event, the *check*. A check event has the syntax *check*$(t_1 = t_2)$, where t_1 and t_2 are terms.

By using our new event we can now correctly specify the responder role of Protocol 2.5:

$$\begin{aligned} \textit{resp}(A, B, M, L, C, K) \quad = \quad & \langle \langle B : (L, C) \triangleleft A \rangle \ \langle B : sig_{pk(B)}(C) \triangleright A \rangle \\ & \langle B : (sig_{pk(A)}(K), K) \triangleleft A \rangle \\ & \mathbf{check}(\mathbf{L} = \mathbf{h}(\mathbf{M}))\ \mathbf{check}(\mathbf{C} = \{\mathbf{M}\}_{\mathbf{K}}) \\ & \langle B : sig_{pk(B)}(L, C, K) \triangleright A \rangle \ \rangle \end{aligned}$$

Semantics of the check event are straightforward, by using unifiers. Procedure 2.3.4 is extended as follows:

Procedure 2.6.1. *Just like Procedure 2.3.4, but add another case for* ev*:*

$1.(c)$. *If* ev *is a check event,* $ev = check(t_1, t_2)$*, let* θ *be its unifier, i.e.* $t_1\theta = t_2\theta$*. Apply procedure* **P** *to* $CS\theta$*, obtaining solution* CS'', γ'*. Let* γ *be* $\theta\gamma'$*.*

Valid traces need also be extended, by simply letting a ground trace be valid if for each $i \in [0, \textit{length}(tr) - 1], last(tr_{i+1}) = check(t_1, t_2)$ implies that $t_1 = t_2$.

This new Procedure 2.6.1 can be seen to produce valid traces as defined above, thus obtaining a result similar to Proposition 2.3.6. We do not pursue that direction here (see the Related Work for the work [79] where a similar operation is studied).

2.7 Guessing Attacks

Security protocols that use weak passwords (e.g. human chosen) can be subject to *guessing attacks* [101] (also known as dictionary attacks). Guessing attacks exist in two flavours: online and offline. In *online* guessing attacks the intruder is allowed to generate fake messages and to supply them to the honest principals, for instance for checking whether a certain guess is correct. In *offline* guessing attacks, on the other hand, the intruder *first* gathers some knowledge K from the protocol execution, and *then* proceeds offline to perform a password search.

Consider the simple exchange between a and b shown in Protocol 2.7.

Message 1.	$a \rightarrow b : (a, na)$
Message 2.	$b \rightarrow a : \{na\}_{pab}$

Protocol 2.7: Simple exchange protocol

There, na is a's nonce, pab is a shared weak password between a and b. After an execution of Protocol 2.7, the attacker (who has eavesdropped all the traffic) has gathered a knowledge set $K = \{(a, na), \{na\}_{pab}\}$. With this knowledge, the intruder can mount an offline guessing attack on pab, using brute force by proceeding as follows (shown in pseudocode):

Procedure 2.7.1. *Offline Guessing Attack over Protocol 2.7:*

1. *obtain* na *by splitting* (a, na)
2. *REPEAT*
3. *Generate a guess* p *(possibly using a dictionary)*
4. *Let* na' *be the result of decrypting* $\{na\}_{pab}$ *using* p
5. *UNTIL* na' *matches* na

In this procedure, the gathered knowledge $K = \{(a, na), \{na\}_{pab}\}$ allows the intruder to check whether a given guess of pab is correct. In this case we say that pab is *guessable* wrt K. Intuitively, a password p is guessable in K if it is possible to infer a term v (the *verifier*) from $K \cup \{p\}$ in two *different ways*, along with the condition that p plays a crucial role in at least one of these ways of deriving v. In our example, the verifier was na, which is derivable from $\{na\}_{pab}$ using the guess and from (a, na) by projection.

Note that we assume that decryption always succeeds, returning a value. Some novel cryptosystems actually reject a ciphertext when it is intended to be decrypted with the wrong key, which actually simplifies a guessing attack (this assumption is adopted in the definition of guessing attacks given by Abadi and Warinschi [28], see the Related Work).

2.7.1 An Intuitive Definition

Lowe analyzes protocols for guessing attacks in [135]. His definition is:

> A guessing attack consists of the attacker guessing a value g, and then verifying that guess in some way. The verification will be by the intruder using g to produce a value v, which we call the *verifier*; the verifier will demonstrate that the guess was correct, i.e. an incorrect guess would not have led to this value. This verification can take a number of different forms: (1) the attacker knew v initially, or has seen v during the protocol run; (2) the attacker produced v in two different ways; or (3) v is an asymmetric key, and the attacker's knows the inverse of v from somewhere.

Let us leave (3) aside for the moment. We can combine (1) and (2) as follows:

The attacker produced v in two ways, and at least one of these two ways was not possible before using the guess.

Now, whether an attacker can produce v in two different ways can be determined by simply masking that occurrence of v with some fresh constant (say v') and see if he can produce v and v' again. This observation leads to the following first definition of guessing attacks.

Definition 2.7.2. *Let T be a ground term set, and let g be a ground term denoting a guess. Also, let v be a subterm of a term in T, and let v' be a fresh constant not occurring in T.*

We say that g is guessable *w.r.t. T if there exists an occurrence of v s.t. the following conditions hold (where T' is the set of terms obtained by replacing the particular occurrence of v in T with v'):*

1. *$v \in \mathcal{F}(T' \cup \{g\})$ and $v' \in \mathcal{F}(T' \cup \{g\})$; and*
2. *$v \notin \mathcal{F}(T')$ or $v' \notin \mathcal{F}(T')$.*

In this case, v is called the verifier *of g. (Recall that $\mathcal{F}(T)$ is defined in Definition 2.2.1.)*

Intuitively, by replacing v with a fresh constant v' in T we obtain a test set term T', where if we can derive both v' *and* v in T', then the original v must have been possible to derive in two different ways in T. The second condition simply checks that these two ways are not trivially derivable *before* the guess of g.

Example 2.7.3. *We illustrate Definition 2.7.2 on some examples.*

- *Let $T = \{na, \{na\}_{pab}\}$. Let g be pab, and pick the leftmost occurrence of na as the verifier (v). Then, $T' = \{v', \{na\}_{pab}\}$. It is straightforward to check that $v' \in \mathcal{F}(T')$, $na \notin \mathcal{F}(T')$ (satisfying condition 2 of Definition 1), and $v' \in \mathcal{F}(T' \cup \{g\})$ and $na \in \mathcal{F}(T' \cup \{g\})$ (satisfying condition 1 of Definition 2.7.2). Hence, that occurrence of na is a verifier for g and there is a guessing attack.*
- *Let $T = \{\{na\}_{pab}, \{(na, nb)\}_{pab}\}$. Again make $g = pab$ and $v = na$ (in $\{na\}_{pab}$). So, $T' = \{\{v'\}_{pab}, \{(na, nb)\}_{pab}\}$. Both na and v' are not derivable from T' (satisfying condition 2), while they are both derivable from $T' \cup \{g\}$ (satisfying condition 1). Thus, that occurrence of na is a verifier for g and there is an attack.*
- *Let $T = \{\{na\}^{\rightarrow}_{pk(b)}, \{na\}_{pab}, pk(b)\}$. Let $g = pab$, and pick $v = \{na\}^{\rightarrow}_{pk(b)}$. Then, $T' = \{v', \{na\}_{pab}, pk(b)\}$. Now, $v' \in \mathcal{F}(T')$, $\{na\}_{pk(b)} \notin \mathcal{F}(T')$ (satisfying condition 2), and $v' \in \mathcal{F}(T' \cup \{g\})$ and $\{na\}^{\rightarrow}_{pk(b)} \in \mathcal{F}(T' \cup \{g\})$ (satisfying condition 1).*

Hence, $\{na\}^{\rightarrow}_{pk(b)}$ is a verifier for g, and there is a guessing attack. However, consider $v = na$ (in $\{na\}_{pab}$). $v' \in \mathcal{F}(T' \cup \{g\})$ but $v \notin \mathcal{F}(T' \cup \{g\})$ (not satisfying condition 1). Hence, the particular v cannot be a verifier and it was a wrong choice.

The definition also implicitly includes another special case of Lowe, where the protocol itself gives $\{g\}_g$ to the attacker. In this case, we select v as g (inside the encryption). After guessing, v and v' can be obtained from $\{\{v'\}_g\} \cup \{g\}$ (satisfying condition 1), but not before guessing (satisfying condition 2).

The only case in Lowe's definition not captured by our definition is his condition (3). However, we argue that this is an implementation dependent attack, which can be handled accordingly at such level. This is so since some terms are recognized by the attacker immediately (e.g. English text), even if there is no double derivation. The public/private keys fall in this category, that can be specified as part of the input of an implementation, and needs no search for double derivation as we do here.

Finding guessing attacks in Constraint Solving Up to now we only considered a set of terms T which is ground. In this section we merge the procedure with constraint solving, thus allowing potential variables to occur in T. We extend Procedure 2.3.4 as follows.

Definition 2.7.4. *Given a state $s = \langle Sc, IK, CS, tr\rangle$ from Procedure 2.3.4, we define the* termination condition *(see Section 2.3.1) for guessing attacks, $TC_g(s)$, to hold if the following steps succeed:*

- *Pick a ground subterm v in $K(tr)$. Let K be the result of replacing an occurrence of v in $K(tr)$ with a fresh constant v', not occurring in $K(tr)$ nor in CS.*
- *Applying* **P** *to $CS \cup \{v : K \cup \{g\} \cup IK, v' : K \cup \{g\} \cup IK\}$ succeeds with partial solution σ and*
- **P** *fails either on $(CS \cup \{v : K \cup IK\})\sigma\sigma_V$ or on $(CS \cup \{v' : K \cup IK\}\sigma\sigma_V$.*

This definition is appropriate, as shown in the next proposition:

Proposition 2.7.5. *Let $s = \langle Sc, IK, CS, tr\rangle$ be a state in Procedure 2.3.4, let g be a ground term representing a weak password, and let $TC_g(s)$ be the termination condition as defined in Definition 2.7.4 above. Suppose that $TC_g(s)$ holds. Then there exists γ s.t. g is verifiable w.r.t. $K(tr)\gamma \cup IK$, as defined in Definition 2.7.2.*

Proof. Let γ be $\sigma\sigma_V$ as given in Definition 2.7.4. By soundness in Theorem 2.2.15 since σ is a partial solution, we obtain that $CS\sigma$ is solvable with solution σ_V. Hence, $v \in \mathcal{F}(K\sigma\sigma_V \cup \{g\} \cup IK)$ and $v' \in \mathcal{F}(K\sigma\sigma_V \cup \{g\} \cup IK)$, obtaining Definition 2.7.2 (1). To obtain (2), we use Corollary 2.4.2 to deduce that either $v \notin \mathcal{F}(K\sigma\sigma_V \cup IK)$ or $v' \notin \mathcal{F}(K\sigma\sigma_V \cup IK)$ (since v and v' are ground by assumption). □

Note that although this proposition establishes soundness for the procedure given in Definition 2.7.4, in principle we could use Theorem 2.4.4 to use it to establish completeness. However we cannot apply the theorem as it is, since there may be non ground verifiers v of g, which we cannot decide since our negated constraint solving strategy currently only works for ground cases (see Section 2.4).

Type-flaw guessing attacks Some techniques to prevent attacks involving type-flaws and multiple protocols may actually facilitate a guessing attack. In [110], Heather et al. provide a method to prevent type-flaw attacks, by tagging the message fields with their intended types. However, type-tagging should not be implemented when the protocols are using poor passwords. To see why, consider again Message 2 of Protocol 2.7, $\{na\}_{pab}$. Now, an attacker has to decrypt it with a guess, obtain na in another way and compare to verify the guess. But if the message is typed, like $\{\mathsf{nonce},\ na\}_{pab}$, after decryption with the guess, presence of the tag 'nonce' would itself verify the guess. He does not even need to know na![5] However, in the absence of any mechanisms to detect type-flaws, protocols may be vulnerable to *type-flaw guessing attacks*. Consider Protocol 2.8.

Message 1.	$a \rightarrow b : \{\{k\}^{\rightarrow}_{pk(b)}, \{\{k\}^{\rightarrow}_{pk(b)}\}_k\}_{pab}$
Message 2.	$b \rightarrow a : \{nb, \{k_2\}_k\}_{pab}$
Message 3.	$a \rightarrow b : \{nb\}_{k_2}$

Protocol 2.8: Artificial Protocol for Type-flaw Guessing Attacks

Protocol 2.8 has an attack involving a type-flaw. During the on-line phase of the attack, the attacker performs the following communication with a (we write e_x when the attacker impersonates honest principal x):

$$\begin{array}{l}\text{Message 1. } a \rightarrow e_b : \{\{k\}^{\rightarrow}_{pk(b)}, \{\{k\}^{\rightarrow}_{pk(b)}\}_k\}_{pab}\\ \text{Message 2. } e_b \rightarrow a : \{\{k\}^{\rightarrow}_{pk(b)}, \{\{k\}^{\rightarrow}_{pk(b)}\}_k\}_{pab}\\ \text{Message 3. } a \rightarrow e_b : \{\{k\}^{\rightarrow}_{pk(b)}\}_{\{k\}^{\rightarrow}_{pk(b)}}\end{array}$$

In message 2, the attacker replays message 1 back to a. Now the attacker moves to the off-line phase. The attacker guesses pab, decrypts the first message with the guess, splits it, and takes the first part ($\{k\}^{\rightarrow}_{pk(b)}$) out of it. He can then decrypt the third message with this to obtain it $\{k\}^{\rightarrow}_{pk(b)}$ again, thereby verifying the guess.

Limitations of Definition 2.7.2.

Non- atomic keys. When considering non-atomic keys in encryption Definition 2.7.2 is not satisfactory. Consider e.g. $T = \{\{(\{k\}_k, \{k\}_{\{k\}_k})\}_{pab}\}$. A guess of pab is verified, by decrypting the only message in T, obtaining $\{k\}_k$ and $\{k\}_{\{k\}_k}$, then getting k by decrypting the latter using the former and finally then decrypting $\{k\}_k$ with k, and noticing a match of k, thus making k the verifier. Of course, this example is contrived and artificial, but it shows that sometimes the replacement of some subterm of T with a fresh v' disallows some derivations.

Protocols replaying material. Some protocols replicate material which may lead Definition 2.7.2 to find a 'fake' attack. For example, consider a protocol that receives a value $\{n\}_g$ and returns $(\{n\}_g, \{n\}_g)$. Then we would let v be n, replace n by v', and obtain

[5] This is unrealistic if nonce is short, as the change of getting it is high even if the guess is wrong. However, if we assume the attacker can gather many instances of the message, i.e. $\{\mathsf{nonce},\ na_0\}_{pab}, ..., \{\mathsf{nonce},\ na_n\}_{pab}$, then it is a valid attack.

an attack since both v and v' are derivable from $(\{v\}_g, \{v'\}_g)$ and g. However, any guess of g' of g, even if $g' \neq g$, would decrypt $\{v\}_g$ and $\{v'\}_g$ to the same value, simply because decryption is deterministic[6]. Hence, such protocols should be avoided when analysing guessing attacks with Definition 2.7.2.

2.8 Putting it All Together

In Figure 2.2 we show how the techniques developed in this chapter are integrated. On the left hand side we see Procedure 2.3.4, which maintains a state $s = \langle Sc, IK, CS, tr\rangle$ consisting of a scenario Sc, a constraint set CS, an initial intruder knowledge IK, and an execution trace (possibly symbolic) tr.

The *termination condition* (Section 2.3.1) predicate specifies when state S of Procedure 2.3.4 is insecure (i.e. there is an attack), and as such we can terminate execution and report an attack (with leading trace tr). Each extension is pure, since specifies *only* its own particular termination condition, as shown in the boxes on the right of Figure 2.2:

$TC_A(s)$ The basic way of specifying termination conditions is shown by requiring some roles A to terminate their execution. This is expressed as predicate $TC_A(s)$, in the first box (upper right of Figure 2.2).

$TC_\phi(s)$ Another way to specify a termination property is by specifying a security property ϕ in the language $\mathcal{PS}$-LTL. We then have termination condition $TC_\phi(s)$, as given by Procedure 2.5.16 (middle right in Figure 2.2).

$TC_g(s)$ Guessing attacks over a weak key g (e.g. a password chosen by a human) by setting a termination condition $TC_g(s)$ to be a predicate implementing Procedure 2.7.4 (bottom right in Figure 2.2).

The complete procedure is then:

Procedure 2.8.1. *As inputs we take a system scenario Sc, an initial intruder knowledge IK and one of the following:*

- *A set $A \subseteq Sc$ of roles required to terminate indicating that an attack has happened; in this case we let the termination condition $TC(\cdot) = TC_A(\cdot)$. $TC_A(\langle Sc', IK, CS, tr\rangle)$, holds when for every role $r \in Sc'$ with $r \neq \langle\rangle$, r is not a suffix of any instance of some role $r' \in A$.*
- *A $\mathcal{PS}$-LTL formula ϕ; in which case we let $TC(\cdot) = TC_\phi(\cdot)$. $TC_\phi(\langle Sc', IK, CS, tr\rangle)$ is true when $\mathbf{D}(\pi, CS) = true$, for $\pi = \mathbf{T}(\neg\phi, tr, IK)$ well-behaving; or*
- *A weak password g with which we are going to mount a guessing attack; in which $TC(\cdot) = TC_g(\cdot)$. $TC_g(s)$, holds if the following steps succeed:*
  - *Pick a ground subterm v in $K(tr)$. Let K be the result of replacing an occurrence of v in $K(tr)$ with a fresh constant v', not occurring in $K(tr)$ nor in CS.*

[6] We are grateful to S. Delaune and F. Jacquemard to point out the issue of Definition 2.7.2 w.r.t. protocols replying materials in their paper [80].

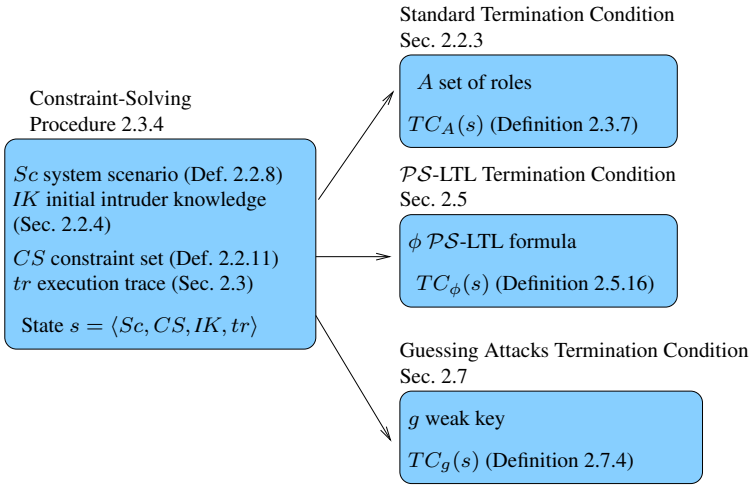


Figure 2.2: Putting it all together

– *Applying* **P** *to* $CS \cup \{v : K \cup \{g\} \cup IK, v' : K \cup \{g\} \cup IK\}$ *succeeds with partial solution* σ *and*

– **P** *fails either on* $(CS \cup \{v : K \cup IK\})\sigma\sigma_V$ *or on* $(CS \cup \{v' : K \cup IK\}\sigma\sigma_V$.

A run for scenario Sc_0 with initial intruder knowledge IK *is a sequence of execution steps starting from state* $\langle Sc_0, IK, \emptyset, \langle\rangle\rangle$*, where each step is performed as follows, and in which the last state* s *of the sequence satisfies* $TC(s)$.

A state *is a 4-tuple* $\langle Sc, IK, CS, tr\rangle$*, where* Sc *is a system scenario,* IK *is the initial intruder knowledge,* CS *is a simple constraint set and* tr *is a (possibly non-ground) trace. An* execution step *from state* $s = \langle Sc, IK, CS, tr\rangle$ *to* $s' = \langle Sc', IK, CS', tr'\rangle$ *is obtained by performing the following:*

1. *Choose non-deterministically a non-empty role* $r \in Sc$*. Let* $r = \langle ev\ r'\rangle$*. Consider the following cases for* ev*:*

   (a) *If* ev *is a* send *communication or a* status *event, let* γ *be the empty substitution and* CS'' *be* CS*.*

   (b) *If* ev *is a* receive *communication event, ie.* $ev = \langle a : m \triangleleft b\rangle$*, check that the intruder can generate* m *using the knowledge* $K(tr) \cup IK$*, by applying procedure* **P** *to* $CS \cup \{m : (K(tr) \cup IK)\}$*, obtaining a new simple constraint set* CS'' *and a partial solution* γ *(Note that there may be many possible* CS'' *and* γ*).*

   (c) *If* ev *is a* check *event*[7]*,* $ev = check(t_1, t_2)$*, let* θ *be its unifier, i.e.* $t_1\theta = t_2\theta$*. Apply procedure* **P** *to* $CS\theta$*, obtaining solution* CS'', γ'*. Let* γ *be* $\theta\gamma'$*.*

[7]Caveat: explicit checks have not been considered in the developments of $\mathcal{PS}$-LTL and guessing attacks, and should only be used with the standard termination condition $TC_A(\cdot)$.

2. *Let* $Sc' := (Sc \setminus \{r\} \cup \{r'\})\gamma$, $CS' := CS''$ *and* $tr' := \langle\, tr\gamma\ ev\gamma\ \rangle$.
   This sets the step transition from states s *to* s'*, for* $s = \langle Sc, IK, CS, tr\rangle$ *and* $s' = \langle Sc', IK, CS', tr'\rangle$.

Procedure 2.8.1 completes the description of our extensions based on constraint solving for analysing security protocols.

2.9 Case Studies

The techniques developed in this chapter have been used to analyze several case studies. These case studies were developed by fellow PhD students (C.N. Chong, Y.W. Law and G. Lenzini) at the University of Twente, and were found to benefit from a formal analysis using our implementation of Procedure 2.3.4.

We report that each formal analysis was useful to establish relevant security properties, and the collaborations resulted in published works [11, 12, 10]. Each case is described in detail in the PhD theses of Chong [69], Lenzini [130] and Law [127]; here we provide a short summary of each case.

As a contribution to the Telematica Instituut project LicenseScript [70], Chong [69] presents a content (e.g. MP3 music or movies) protection scheme, which exploits tamper-resistant cryptographic hardware. Content protection mechanisms are intended to enforce the usage rights on the content, which are carried by a *license* (which may even include the key that is used to unlock the protected content). In [69] a protocol is introduced to specify the interactions between the hardware token, the application, and the license provider. Our constraint solving procedure is used to verify the designed protocol. Indeed, several early versions were debugged after flaws were found by the procedure. We analyze secrecy (e.g. of fresh nonces as exchanged by the token and the genuine provider) and authentication (e.g. to guarantee that a malicious application and provider cannot impersonate the genuine ones and render the content even if unauthorized).

Lenzini [130] analyzes the security of the Trust and Security Management (TSM) protocol, an authentication protocol which is part of the Parlay/OSA Application Program Interfaces (APIs) [17], used by Italia Telecom and other telecommunication companies. Architectures based on Parlay/OSA APIs allow third party service providers to develop new services that can access, in a controlled and secure way, network capabilities offered by the network operator. The role of the TSM protocol, run by network gateways, is to authenticate the client applications trying to access and use the network capabilities on offer. We use our constraint solving procedure to analyze the TSM protocol, and underline some problems. Regarding secrecy, our analysis shows that some interfaces assumed to be secret can be found by the intruder, thus exposing a vulnerability. Regarding authentication, our analysis highlights a vulnerability regarding the negotiation of session parameters (e.g. encryption algorithm). In TSM, such negotiation happens *before* the actual authentication (based upon CHAP-based authentication, studied in [129]), and hence can be subverted by the intruder (e.g. by selecting the weakest encryption algorithm). This is serious, since even if this negotiation is protected with some key, our analysis shows possible replays (since the authentication phase has not happened yet).

Finally, as a contribution to the European project Eyes [128], Law [127] presents a decentralized key management architecture for wireless sensor networks, covering the aspects of

key deployment, key refreshment and key establishment. Balance between security and energy consumption is achieved by partitioning a system into two interoperable security realms: the supervised realm trades off simplicity and resources for higher security whereas the unsupervised realm targets the other extreme of the trade off. Key deployment uses minimal key storage while key refreshment is based on the well-studied scheme of Abdalla et al. [29].

The keying protocols involved use only symmetric cryptography and have all been analyzed with the constraint solving technique of this chapter, providing confidence of the protocols' correctness. Many design issues (for example when one should include the sender's ID and/or the receiver's ID in the messages of the protocols) were clarified by analyzing the protocols using the constraint solving tool. Also, the tool helped to analyze large scenarios which are beyond informal analysis. For example, the Inter-Supervised Cluster Keying protocol contains 9 steps in total, involving 4 parties: one supervisor S_A and one supervised node A of one cluster, one supervisor S_B and one supervised node B of another cluster. The purpose is to establish a shared key between A and B through the negotiation between S_A and S_B. During the protocol design process, we found replay attacks due to the similarity between two messages. We then modified one message accordingly, to avoid clashes, and rerun our analysis, to finally verify the security of the protocol against any kind of replay attacks.

2.10 Constraint Solving as a Teaching Tool

The implementation [7] of Procedure 2.3.4 has been used as a tool to teach security protocols and their analysis, in two guest lectures given by the author to about one hundred undergraduate students of the Distributed E-business Techniques (DEBT) course at University of Twente, for three consecutive years (2003-2005).

The teaching experience helped the author to understand how to introduce protocol analysis to students. More importantly, we believe the experience has been positive for the students, as a significant percentage appreciated and understood the potential of formal analysis. For example, in the final exams for the course some questions were (almost always) successfully answered:

- In one question, the students were required to fix system scenarios (see Section 2.2.2) to analyse different toy protocols (e.g. simplified versions of the Needham Schroeder protocol). In particular, we required scenarios for testing secrecy and authentication.

- In another question, the students were required to explain why in a given scenario different fields were instantiated (e.g. a nonce) while other variables were left uninstantiated (e.g. an unknown communication party) (see e.g. Example 2.2.8).

However, although the theoretical aspects of the constraint solving procedure were understood, not more than a handful of students actually used the tool to explore further (e.g. to analyse other protocols or design their own).

We conclude this section by mentioning that our implementation is also one of the recommended protocols analysis tools [158] for the course "CS 395T - Design and Analysis of Security Protocols" taught by Vitaly Shmatikov during Fall 2004 at University of Texas.

2.11 Related Work

Related work for Constraint-based analysis of Security Protocols We first revisit references as given originally by Millen and Shmatikov [149]. The work of Millen and Shmatikov [149] for verification of protocols (supporting the presence of encryption with non-atomic keys) follows earlier work of Huima [112], for symbolic exploration of an infinite state generated from a limited number of protocol participants. There, a term algebra with reductions which cancel each other is used, although details about their completeness result are missing.

Amadio, Lugiez and Vanackère [32] present a similar decision procedure to Millen and Shmatikov, although only covering encryption with atomic keys (in contrast to non-atomic keys as in [149]). Rusinowitch and Turuani [157] show the problem to be NP-complete when using a free algebra and constructed keys. Algorithms for computing symbolic traces are given by Boreale [59] and Fiore and Abadi [92]; the algorithms are technically involved, with the latter proving completeness only for encryption with atomic keys.

The first efficiency improvement to Millen and Shmatikov's procedure was first presented by the authors [5], as shown in this chapter in Section 2. Our efficiency improvement is dubbed *common path optimization* by Millen, due to fact that if an interleaving is unsolvable then other interleavings extending it, and then sharing a common initial path, are also unsolvable and thus not need to be considered (see Observation 2.3.1). Another efficiency optimization is presented by Basin, Mödersheim and Viganò [39]. Briefly, the idea there is that new constraints are added only when they cover new ground state space, and thus never include constraints which overlap with existing ones: hence the name *constraint differentiation*. This technique is used in the OFMC back-end of the AVISPA tool, see below.

The strand space [162] is adopted both by [149] and our work [5]. Also using strand spaces is Athena [159], although it only supports encryption with atomic keys. The *fake* function $\mathcal{F}(\cdot)$ is similar to Paulson's [155] synth and analz functions to characterize the attacker capabilities.

Several relaxations to the free algebra setting have been proposed. A reduction from constraint solving to systems of Diophantine equations is proposed by Millen and Shmatikov [150], to deal with abelian groups, like products and Diffie-Hellman exponentiation. Chevalier presents a constraint-solving decision procedure to deal with XOR [67]. The complexity of deciding security with explicit destructors (one example are our 'explicit checks' as given in Section 2.6; another is presented by Millen [148]) is studied by Delaune and Jacquemard [79], where an NP time procedure is presented.

Active research is carried within the scope of the AVISPA project [16], a framework in which protocols can be specified in the HLPSL language, which is then translated to four different back-ends. Two of these back-ends are related to our constraint solving approach. The on-the-fly model checker OFMC [37, 39] employs several symbolic techniques to explore the state space in a demand-driven way. In particular, it implements the constraint differentiation technique discussed above. Another back-end based on constraint solving is presented by Chevalier and Vigneron [68], dubbed CL-AtSe, standing for Constraint-Logic-based Attack Searcher. CL-AtSe applies constraint solving with simplification heuristics and redundancy elimination techniques. The specification language of the AVISPA project allows the writing of fixed goals (e.g. authentication), and thus it is less expressive than a full language as our $\mathcal{PS}$-LTL of Section 2.5. Regarding efficiency, while in original work [37] heuristics were

needed, the later work [39] is much more developed and heuristics do not play a significant role on efficiency anymore. The technique of *lazy intruder* stands for a Dolev Yao intruder whose actions are generated 'on-demand'. This work does not require encryption hiding as [149] and hence our approach.

Recently, Drielsma and Mödersheim [86] and Kähler and Küsters [115] study contract signing protocols (e.g. the ASW protocol [35] or the recently proposed protocol in [2]). In the former work of Drielsma et al. studies the ASW protocol [35] restricting to reachability properties, while the latter work of Kähler and Küsters addresses complete verification of game-theoretic properties (e.g. fair exchange and non-repudiation [35, 121, 66]), by requiring an initial stage of guessing the right attack interleaving, and then turning the problem to a standard constraint solving problem.

Related Work for $\mathcal{PS}$-LTL In previous work [4] the authors study *local* security properties. $\mathcal{PS}$-LTL, as given in Section 2.5, provides more powerful operators (e.g. the `Y` and `S` operator), which in turn allows the writing of more expressive security properties. Besides that work, $\mathcal{PS}$-LTL is inspired by the successful and elegant NPATRL logic [161]. As shown in subsequent work [144], NPATRL is strictly less powerful than LTL; also in that paper it is mentioned that the implementation of NPATRL to NRL Protocol Analyzer [141] presents difficulties (e.g. the inability to mention several `learn`'s in the same formula, a restriction we do not impose). Further examination is required to better compare NPATRL and our logic.

Our treatment of pure-past LTL is an adaptation of Havelund and Rosu [109]. We provide a different semantics, tailored for security and constraint solving, but also include a different definition for `H`, which we believe preserves better the faithfulness to standard LTL.

Finally, we use special flags like *run* and *end*, to later specify authentication properties as correspondence assertions à la Gordon and Jeffrey [102] (which we also use in Chapter 5).

Related Work for guessing attacks The first definition of guessing attacks in a formal setting is given by Lowe [135]. The definition basically aims at capturing the notion of deriving some value in two ways.

However, Lowe's definition is not constructive, and its proposed implementation is based on a CSP model which is difficult to embed in other verification tools.

Recently, another approach to guessing attacks was presented in [78], were also the complexity of guessing attacks was studied. This work was subsequently extended in [80], where an algorithm for finding guessing attacks is described, for a finite number of participants.

Also recently, Abadi and Warinschi [28] relate 'formal' guessing attacks as defined in Section 2.7 and in the above mentioned approaches with *computational models*, in a similar vein to the work of Abadi and Rogaway [27] and the one presented in Chapter 6 of this thesis, based on work from Laud and Corin [9]. The definition of guessing attacks as presented by Abadi and Warinschi assumes probabilistic encryption, and also the fact that the decryption algorithm accepts only valid ciphertexts and rejects otherwise. Also, pairing is considered also verifiable, thus leading to a tight definition of guessing attacks. Our definitions as presented in Section 2.7 look for guessing attacks in a model with less verification means, and thus requires a more involved treatment.

2.12 Conclusions

Constraint solving is a simple and efficient technique for analyzing security protocols. Its simplicity comes from the clean transformation of protocol runs to constraint sets, which decision algorithm can be easily implemented. This simplicity is also evidenced by the compact Prolog implementations of the analyzers. For example, the original implementation given by Millen and Shmatikov [149] consists of just three pages of Prolog code. Our extensions do not add much more code in their implementations. The technique is also efficient, thanks to its lazy intruder strategy: variable instantiations only occur when required, and otherwise are left uninstantiated. Constraints express neatly the knowledge of an intruder at a given point in time, and its symbolic structure allows naturally to handle a potential infinite state.

Our optimization of Section 2.3 yields a more efficient system, which in turn allows the verification of more complex protocols and scenarios. In turn, our extensions allow more expressivity (e.g. the *checks*) and possibility to verify guessing attacks.

Furthermore, the approach can cope with realistic scenarios (e.g. industrial protocols), as shown by our case studies analysis in Section 2.9.

Endowing constraint solving with a powerful language to specify properties $\mathcal{PS}$-LTL yields a practical and expressive system. $\mathcal{PS}$-LTL is based on linear temporal logic (LTL) with pure-past operators, and it allows to specify several security properties including authentication [133, 75] (*aliveness*, *weak agreement* and *non-injective agreement*), secrecy (*standard secrecy* [56] and *perfect forward secrecy* [84]) and also data freshness. We present a sound and complete decision procedure to check a fragment of $\mathcal{PS}$-LTL against symbolic traces, thus allowing to attach a $\mathcal{PS}$-LTL interpreter into our protocol verification tool [5] thereby providing a full verification system. There are many possible directions to extend $\mathcal{PS}$-LTL. One direction is to implement formula checking more efficiently: for example, such implementation would not recompute the translation of $\mathcal{PS}$-LTL to elementary formula EF *every* time a property is checked, but maintain an internal data structure which can be optimized as the trace gets expanded, following the ideas of [109]. Another possible direction would be to enlarge the subclass Φ of $\mathcal{PS}$-LTL thus obtaining a more expressive language (eg. to cover stronger authentication notions like the ones in [75]). Also, it is interesting to model more protocols and their properties, e.g. from the Clark and Jacob library [71] (we already tested 18 protocols, including the four variants of the Andrew RPC protocol, and also the Needham-Schroeder Public Key protocol, see [7]). Finally, further comparison to other logics would be beneficial, such as NPATRL [161].

CHAPTER 3

A Timed Automata Analysis Model

3.1 Introduction

In this chapter we develop an analysis model that explicitly takes into account *time* flowing during the execution of a protocol. This is realistic since security protocols, like distributed programs in general, are sensitive to the passage of time, hence affecting the security.

In general, in the design and implementation of a security protocol two aspects of timing must be considered at some stage:

1. Time can influence the flow of messages. For instance, when a message does not arrive in a timely fashion (i.e. *timeouts*), retransmissions or other actions have to be considered.

2. Time information can be included within protocol messages (e.g. timestamps).

Consider first (1) above. In general, the influence of time on the flow of messages is not usually considered by current state of the art methods for analysing security protocols. However, we believe it to be crucial because (i) Even if the abstract protocol does not decide what action to take at a particular moment of the execution (e.g. in the case of timeouts), the actual implementation will eventually have to consider these issues anyway; (ii) The efficiency and security of the implementation depends critically on these specific decisions; and (iii) The timing of message flows in a protocol can be exploited by an attacker.

Now consider item (2) above. There, we believe that making judicious use of timing information in a protocol has received attention but mostly in the limited setting of using time stamps as opposed to nonces. However, time information can be used to influence message flows as well, as we illustrate in Section 3.6.

In this chapter we cover several issues in the study of time in security protocols:

- Firstly, in Section 3.2 we study which kinds of timing issues, like timeouts and retransmissions, may arise in the study of security protocols. We then proceed in Section 3.3 to present a method for the design and analysis of security protocols that consider these timing issues. The method is based on modelling security protocols using timed automata [30]. In support of the method we use UPPAAL [33] as a tool to simulate, debug

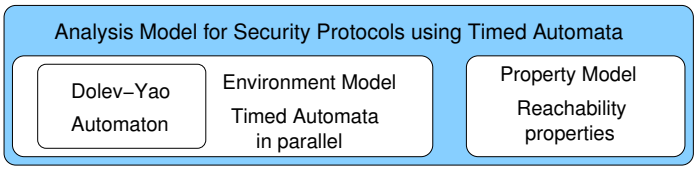


Figure 3.1: Analysis model for security protocols with timed automata

and verify security protocols against classical safety goals like secrecy and authentication, *in a real time scenario*, using reachability properties. As examples, we analyse a simplified version of the Needham Schroeder protocol [131] and the full Yahalom protocol [71] in Section 3.4.

Our developed analysis model is illustrated in Figure 3.1. The general analysis model of the Introduction (shown in Figure 1.1) is here instantiated using timed automata, which enables timed analysis. In Figure 3.1, the attacker submodel implements a timed Dolev Yao attacker, which can synchronize on communications with the other participants' automata, as detailed in Section 3.3.4. Also, the protocol participants are represented as sequential timed automata. Then, both the attacker and participants are composed in parallel to conform a system that can be analysed to establish reachability properties. These properties state either secrecy (i.e. whether it is possible to reach a state in which the attacker automaton knows a secret) or authentication (i.e. whether it is possible to reach a state in which a participant automaton ends its execution believing wrongly it communicates to some participant).

- Secondly, in Section 3.5, we categorize all the protocols from the Clark and Jacob library and the SPORE library into different (more abstract) patterns of message flows with timeouts. We then analyse each abstract pattern, independently of the actual protocols, and establish their timing efficiency and security.

- Finally, in Section 3.6 we illustrate some novel opportunities and difficulties that appear when considering time in the design and analysis of security protocols:

  - In Section 3.6.1 we give an example protocol that accomplishes authentication by exploiting the *timeliness* of messages. The protocol uses time in a conceptually new way, by employing *time challenges* as a replacement for nonces.
  - As a second example of a novel difficulty in Section 3.6.2, we describe how timing attacks [88] can be applied to security protocols, by describing an attack over a (careless) implementation of Abadi's private authentication protocol [19]. Although these protocols can be modelled as timed automata, thus permitting general verification, we leave the detailed verification as future work since for this we need a model checker that is also *probabilistic* (like [77] or [123]): our nondeterministic intruder of UPPAAL is too powerful, since it can always guess correctly times and values even if the probability of guessing is negligible.

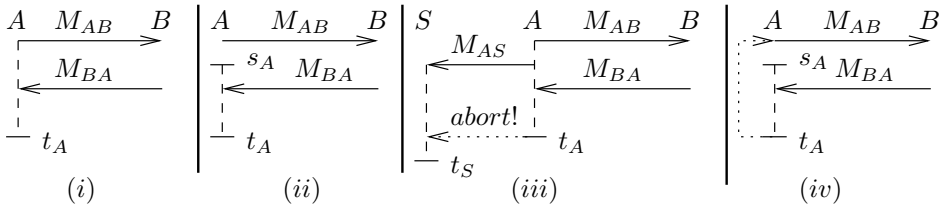


Figure 3.2: *Left*: Timeouts (i) typical (ii) windowed; *Right*: timeout actions (iii) chained abort and (iv) retransmission

3.2 Timeouts and Retransmissions

To illustrate how time influences the analysis of security protocols (even when it does not explicitly use timing information), consider Protocol 3.1, written in the usual notation.

Message 1.	$A \rightarrow B : M_{AB}$
Message 2.	$B \rightarrow A : M_{BA}$

Protocol 3.1: A schematic two message protocol

Here, first A sends message M_{AB} to B, and later B sends message M_{BA} to A. This high-level view does not consider timing. To consider time, we first need to assume that both A and B have *timers*. In this chapter, we do not require timers between parties to be synchronized (see below for a discussion). The next step consists in distinguishing the different operations that occur, with their respective times. In Step 1, it takes some time to create M_{AB}. The other operation that takes time is the actual sending of the message, i.e. the time it takes M_{AB} to travel from A to B. This transmission time is unbounded, since the message may be lost or intercepted, and therefore A may need to *timeout*: After A sends M_{AB}, she starts a timer that will timeout if M_{BA} (Step 2 of the above protocol) is not received after some waiting, say t_A (Figure 3.2 (i)). Clearly, t_A should be greater than the time of creating M_{BA}, plus the average time of sending both M_{AB} and M_{BA}. In general, A does not need to start waiting for a response immediately after sending a message; for instance, A could hibernate (or start doing another task) for some time s_A before beginning to expect the response M_{BA}. This results in a *windowed* timeout (Figure 3.2 (ii)). Typically, the values for s_A and t_A depend on implementation details. However, an implementation independent quantitative analysis could already give an early indication of what attacks can be mounted for some values that are no longer possible for others (e.g. a smaller t_A and a larger s_A).

Another issue that is not considered either is that the *action* to be taken when a timeout occurs is sensitive. Typically, the implicit assumption is that the protocol should abort, as it is the case in Figure 3.2 (i). This means that the protocol party that reaches the timeout deduces that a fault has happened. However, aborting may not consist only of stopping execution altogether. For example, if we consider protocols with several parties, we may wish that when a party timeouts it also communicates its decision to abort to other, still active parties. For instance, consider the following schematic protocol shown as Protocol 3.2.

Message 1.	$A \rightarrow B : M_{AB}$	A starts timer expecting M_{BA}
Message 2.	$A \rightarrow S : M_{AS}$	S starts session timer
Message 3.	$B \rightarrow A : M_{BA}$	

Protocol 3.2: A schematic protocol with timeouts

In Protocol 3.2, if A times out on Step 2, it could communicate the abort decision to S, as shown in Figure 3.2 (iii).

Aborting execution is not the only feasible action to perform after a timeout [124], and in principle protocols could successfully execute when messages *do not* arrive at certain moments. Even if we do assume that a fault occurred, aborting may not be the best choice: sometimes, message retransmission is a better, more efficient and also more realistic option, as depicted in Figure 3.2 (iv). In this case, a question which arises is whether to retransmit the original message (M_{AB} for Figure 3.2 (iv)), or to recompute some parts before resending the message. Here, the trade off is, as usual, between efficiency versus security.

Time information can also be included in the contents of M_{AB} and M_{BA}. A typical value to include is a timestamp, to prevent replay attacks. However, this requires *secure* clock synchronization of A and B, which is expensive (see Mills [151] for a security protocol to achieve this). In fact, this is the reason for which Bellovin et al. recommend to switch to nonces in the Kerberos protocol [46]. Recently, the analysis of security protocols using timestamps has received considerable attention from the research community (see Related Work in Section 3.7). Therefore, in this chapter we do not pursue this direction.

3.3 A Method for Analysing Security Protocols

We use timed automata [30] to model protocol participants, and this has several advantages. Firstly, our method requires the designer to provide a precise and relatively detailed protocol specification, which helps to disambiguate the protocol behaviour. Secondly, timing values like timeouts need to be set at each state, while retransmissions can be specified as transitions to other protocol states.

Once modelled as automata, the protocol can be fed to the real time model checker UPPAAL, which allows the protocol to be simulated and verified. The simulation provides the designer with a good insight of the inner workings of protocol, and already at this stage specific timing values like timeouts can be tuned. Then the designer can proceed with verification of specific properties. As usual in model checking, verification of the protocol is automatic for finite scenarios.

The resulting automata model is an informative and precise description of the security protocol, and thus, it provides a practical way to strengthen implementations while keeping efficiency in mind.

As a third and final step we propose to transfer timing information back to the high level protocol description. This serves to highlight the role of time in the implementation, but also (as we will demonstrate in Section 3.6.1), to make timing an integral aspect of the protocol design.

3.3.1 Timed Automata and UPPAAL

In this chapter, the timed automata of Alur and Dill are used for modelling [30]. In general, timed automata models have an infinite state space. The region automaton construction, however, shows that this infinite state space can be mapped to an automaton with a finite number of equivalence classes (regions) as states [30]. Finite-state model checking techniques can then be applied to the reduced, finite region automaton. A number of model checkers for timed automata is available, for instance, Kronos [167] and UPPAAL [33].

Parallel composition of automata is one of the main sources for expressiveness. This operation allows to decompose complex behaviour, thus supporting transparent modelling. When composing automata in parallel, we need also to provide some form of communication. For the timed automata we use in this chapter, communication comes in form of hand-shake synchronization. Two parallel automata can share a synchronization channel, i.e. both have a transition labelled with a complementing channel name, e.g. *synchronize!* (and *synchronize!*) in the example of Figure 3.3. These transitions may only be taken together, at the same moment. In Figure 3.3 we see an example for a transition, labelled by a guard that has to be true when the transition is taken, a synchronization channel, and a variable update.

Data transmission is typically modelled by a synchronization, where global variables are updated. These global variables contain the data that are transmitted.

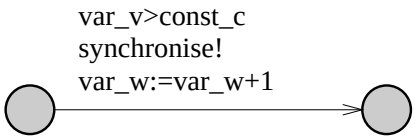


Figure 3.3: Example transition with guard, synchronization and update

Timed automata extend "classical" automata by the use of real-valued clock variables. All clock variables increase with the same speed (derivation 1). For timed automata we make a difference between a state and a location: a state is a location where all clocks have a fixed value. In this sense a location symbolically represents an infinite set of states, for all the different clock valuations. In Figure 3.4 an elementary fragment of timed automata is shown. When the transition from location I to location II takes place, the clock *clock* is reset to 0. Location II may only be left at time D, where D is a constant. The invariant *clock<=D* at location II enforces that the transition to III has to be taken at time D.

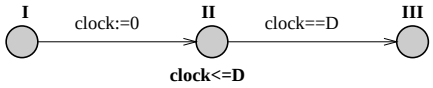


Figure 3.4: Basic timed automaton fragment with a clock and a constant D

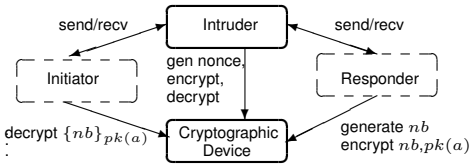


Figure 3.5: The UPPAAL Model

Typically, the initial location of an automaton is denoted by a double circle. We also make use of committed locations, which have the property that they have to be left immediately. In most cases committed locations are used to model a sequence of actions with atomic execution.

The properties verified by the model checker of UPPAAL are reachability properties, like "there is a state where property *p* holds reachable from the initial state", or the dual "in all reachable states property *p* holds". The latter is falsified if the model checker finds a state that does not satisfy *p*. In this case a *diagnostic trace* from the initial state to the state that does not satisfy *p* is produced by the model checker; it serves as counterexample.

We use this mechanism to find attacks. If we can characterize for example the fact that some secret is not secret any more as a propositional property, and the model checker finds a state where this property holds, the diagnostic trace describes a sequence of actions that leads to this state, which gives precisely the attack.

Note that in this context verification comes very much in the guise of debugging. Finding an attack requires an adequate problem model. Not finding an attack increases the confidence in the modelled protocol, but does not exclude that attacks could be found in other models for the same protocol.

3.3.2 Overview of the UPPAAL Model

Let us now describe the general form of our model, in some detail. We model the protocol participants (initiator, responder, etc) and the intruder as timed automata. Additionally, we model cryptography as another automaton, the *cryptographic device*, which acts as an impartial party that regulates the access to data. In Figure 3.5 we illustrate a scenario consisting of one initiator and one responder. Here, boxes in bold represent our general intruder and the cryptographic device, while dashed boxes represent the actual initiator and responder. These participants use the cryptographic device to perform operations, but communicate through the intruder (thus the intruder is identified with the network itself, obtaining a Dolev Yao like intruder [85]). Our modelling is *modular*, and allows us to "plug in" different participants (e.g., in the analysis of the Yahalom we add a *server*), while the bold boxes, i.e. the intruder and the cryptographic device, are the core model.

While modelling security protocols as timed automata in UPPAAL, we will focus on modelling the times required by the principals to encrypt and decrypt values (and generate nonces), but not on the actual time that takes the sending (transmission times are assumed to be unknown). Therefore, for our results to be useful, we assume that computing times (e.g. cryptographic operations) are not negligible w.r.t. communication times, and thus choices for

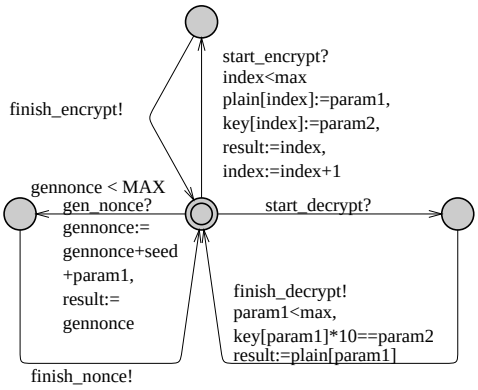


Figure 3.6: Timed automaton for a Cryptographic Device

timeout values depend both on communication and computing times.

3.3.3 Modelling Cryptography

The automaton for a cryptographic device is presented in Figure 3.6. This cryptographic device performs nonce generation and public key cryptography. Later we also use a device for symmetric cryptography, which can be obtained from the one in Figure 3.6 in a straightforward manner. In fact, our method allows different cryptographic devices to be plugged in as needed (e.g. to add hashing). Basically, the device model is a shared table containing pairs of plaintexts and keys. The first service of the cryptographic device is to provide fresh nonces to the protocol participants (and also the intruder). The process of nonce generation is started via synchronization on the *gen_nonce* channel. To model the new nonce creation, the local variable *gennonce* is incremented with the constant *seed* plus the value of `param1` that includes the ID of the requesting participant (this ensures that initial generated nonces differ from each other). The number of possible nonces is limited, by bounding *gennonce* by constant *MAX*. After synchronization, a global `result` variable is updated with a generated nonce, and the device finishes by synchronizing on the *finish_nonce* channel.

Encryption and decryption are modelled by two local arrays to the cryptographic device, namely `plain` and `key`. When a party wants to encrypt some value d with key k, it synchronizes with the device via the channel *start_encrypt*. If the device has still room in its tables, it stores d in the `plain` array and k in the `key` array. As a result, it sets in the global variable `result` the *index* in which d and k reside in the arrays. This index is the "ciphertext". Upon decryption, the ciphertext is provided to the cryptographic device, which then checks that the provided key is correct: Since we model public-key encryption, the private key of a public key k is simply modelled as a function f s.t. $f(k) > MAX$, so that private keys do not clash with generated nonces and hence are never known by the attacker simply by guessing. In this simple case we simply let $f(k) = 10k$, which since only one nonce is needed by the participants in the toy protocol, gives enough room for the attacker to generate nonces.

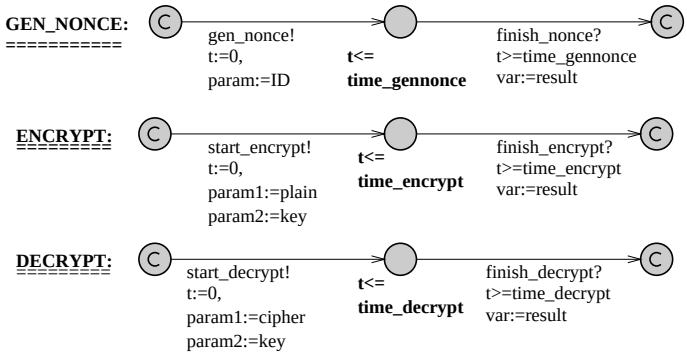


Figure 3.7: State constructions: nonce generation (above), encryption (middle) and decryption (bottom)

State constructions Now that we have the cryptographic device, an honest principal can use different state constructions to perform cryptographic operations. In Figure 3.7 we show the different kinds of state constructions used in our models, which designers should use as building blocks for the representation of protocol participants.

In the upper left of Figure 3.7 we see the building block for nonce generation. Here, a protocol participant first resets the clock `t`, assigns its identity to variable `param` (used by cryptographic device to provide different nonces to different participants) and then fires via the gen_nonce channel. Then the participant enters a state in which it waits until the time of nonce generation happens (`time_gennonce`), synchronizes via the $finish\_nonce$ channel and obtains the return value via variable `result`. Encryption and decryption are analogous, and only differ in that they use two parameters `param1` and `param2` (for plaintext and key in the former, and ciphertext and key in the latter).

3.3.4 Modelling the Adversary

The intruder, presented in Figure 3.8, works basically as a Dolev Yao intruder [85]. The intruder models the network itself, by acting as an intermediary of communication between the initiator and responder. This is modelled by letting the intruder synchronize on both channels $init\_msg$ and $resp\_msg$. Upon synchronizing by receiving a message, the intruder moves to state (SINT1), where it saves the message `msg` in its local variable `data` and resets an index variable i which bounds the total number of actions allowed to do before continuing execution. Then, the intruder moves to state (SINT3), where it makes a nondeterministic choice for an action. More precisely, it can decide to:
– Choose an identity in its local variable `pk` (State SINT4)
– Encrypt a value (State SINT5)
– Decrypt a value (State SINT6)
– Generate a nonce (State SINT7)
– Save variable `data` as message `msg`.
The intruder can then continue to perform these actions, choose to send a message or simply

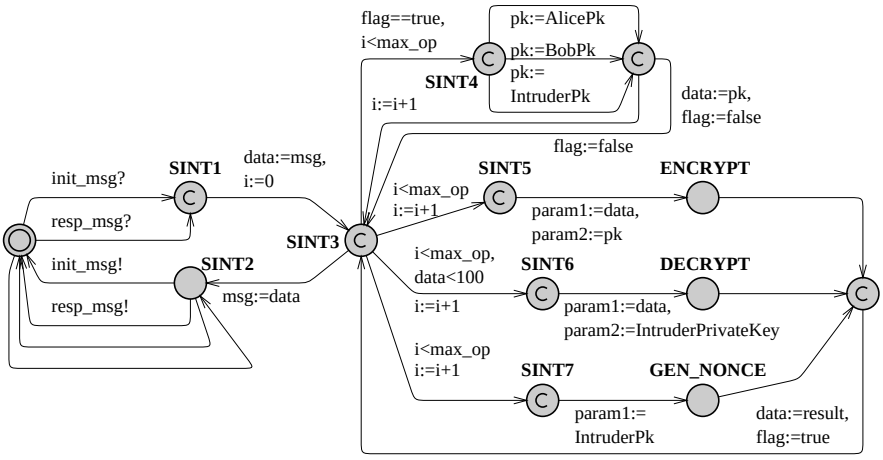


Figure 3.8: Schema for the timed automaton for the Intruder

block a message and continue the execution. Moreover, the intruder can also delay arbitrarily a message, by waiting in state (SINT2).

Note that the intruder is independent of the actual protocol under study, and hence it is generic to analyze protocols using public key encryption (although this intruder is not able to concatenate messages; we extend it in the next section).

3.4 Analysing Protocols

We first consider a simple protocol to illustrate our technique. Later, we move on to analyse the more complex Yahalom protocol.

3.4.1 An Example Protocol

In this section we study and model in UPPAAL a simplified version of the Needham Schroeder protocol, thoroughly studied in the literature (see e.g. [131]), shown as Protocol 3.3. Differently from the Needham Schroeder protocol whose goal is to achieve mutual authentication, our simpler protocol aims at authenticating the initiator A to a responder B only (we do not lose generality here, this is just a simplification to improve presentation).

Message 1.	$A \rightarrow B : A$
Message 2.	$B \rightarrow A : \{N_B\}_{K_A}$
Message 3.	$A \rightarrow B : \{N_B\}_{K_B}$

Protocol 3.3: Simplified Needham Schroeder Exemplary protocol

In the first message, the initiator A sends a message containing its identity to the responder B. When B receives this message, it generates a nonce N_B, encrypts it with the public key K_A of A and sends it back to A. Upon receipt, A decrypts this message with her private key, obtains the nonce N_B, reencrypts it with the public key K_B of B and sends it back to B.

We can now move on to describe the actual initiator, responder and intruder. Both the initiator and responder have local constants `time_out`, which represent their timeout values. Also, the initiator, responder and intruder have local constants `time_gennonce`, `time_encrypt` and `time_decrypt` that represent the time required to generate a nonce, encrypt a value or decrypt a value, respectively for *each* principal.

The automata for the initiator and responder of our simple protocol presented above are given in Figure 3.9 (the dashed transitions of the responder correspond to retransmissions, discussed in Section 3.4.1). The initiator A starts her execution when activated via channel $start$ (State SI0). The actual identity of the initiator role is set via the global variable `init_id` (this and other role variables are chosen by the `Init` automaton, described below). The initiator saves `init_id` as the first message (see protocol message 1). Then, the initiator starts her protocol execution, by firing via the channel $init\_msg$. After this, the initiator starts a clock t and waits for a response, or until t reaches `time_out` (State SI2). If the timeout occurs, the protocol is aborted (a retransmission at this point would be equivalent to restart the protocol). If a response is received before the timeout via the $init\_msg$ channel, A tries to decrypt the received message `msg`. This takes time `time_decrypt` for the initiator. After the decryption, the initiator reencrypts the obtained nonce (stored in `result`) and finally sends the last message via the $init\_msg$ channel, setting to `true` its local boolean variable `finish`.

The responder automaton B works similarly to the initiator. After receiving the start signal, B waits for the message containing the claimed identity of A (State SR1). When received, B saves the first message in the local variable `claimed_id`. After this, B generates a nonce by contacting the cryptographic device. When ready (State SR3), B encrypts the nonce with the value received in Message 1 (we identify identities with public keys). After finishing the encryption (State SR4), the message is sent and B starts to wait for a response (State SR5). If an answer comes before the timeout, B decrypts the message and checks that the challenge is indeed the one B sent. If so, the local boolean variable `finish` is set to `true`.

Verification

We wish to verify that our simple protocol indeed accomplishes authentication of A to B. To this end, we will model check one session of the protocol containing one initiator, one responder and one intruder. We use a special $Init$ automaton that instantiates the initiator and responder with identities (like A, B and I), and then starts the execution run by broadcasting via the $start$ channel. The init automaton is given in Figure 3.10.

The property we check, AUT, is shown in Table 1. AUT states that if we reach a state in which the responder has finished executing but the claimed id (corresponding to the first message of protocol) does *not* coincide with the actual identity of the initiator, the protocol is flawed. Indeed, a state in which the initiator can "lie" and still force the responder to finish means that authentication is violated. This is one of the possible forms of authentication

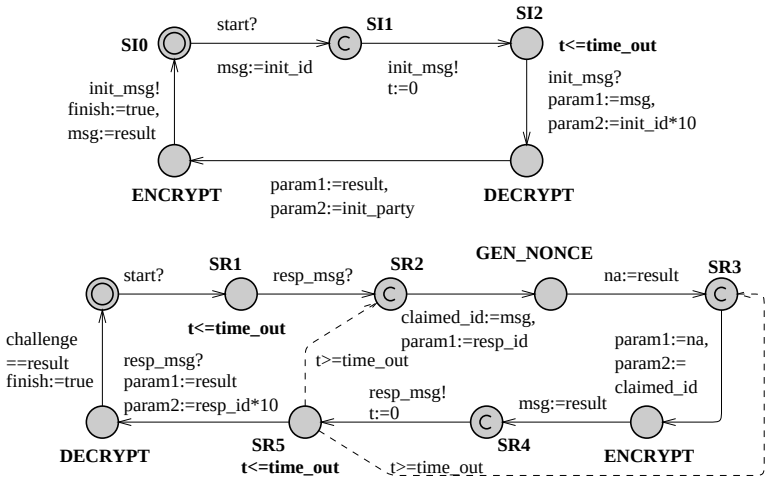


Figure 3.9: Schemas of timed automata for the Initiator (top) and the Responder (bottom)

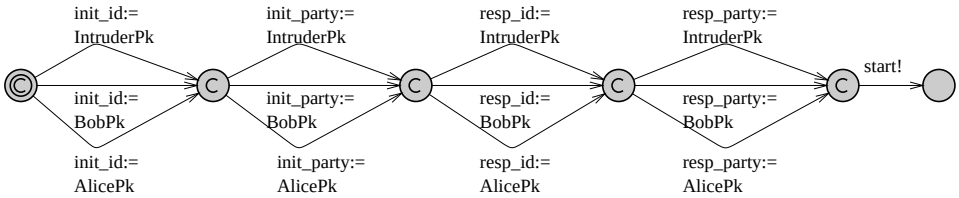


Figure 3.10: Timed automaton for the Init automaton

AUT	$=$	$\texttt{E <> Responder.finish and}$
		$\texttt{Responder.claimed\_id!} = \texttt{resp\_party}$
AUT_y	$=$	$\texttt{E <> Initiator.finish and}$
		$\texttt{Initiator.ticks} < (\texttt{Responder.time\_encrypt}$
		$+\texttt{Responder.time\_gennonce} + \texttt{Server.time\_encrypt} * 2$
		$+\texttt{Server.time\_decrypt}) - 1$

Table 3.1: UPPAAL properties

$\alpha.1$	$A \rightarrow I$	$:A$	1. $I(B) \rightarrow B$	:	B
$\beta.1$	$I(A) \rightarrow B$	$:A$	2. $B \rightarrow I(B)$	:	$\{N_B\}_{K_B}$
$\beta.2$	$B \rightarrow I(A)$	$:\{N_B\}_{K_A}$	3. $I(B) \rightarrow B$	:	$\{N_B\}_{K_B}$
$\alpha.2$	$I \rightarrow A$	$:\{N_B\}_{K_A}$			
$\alpha.3$	$A \rightarrow I$	$:\{N_B\}_{K_I}$			
$\beta.3$	$I(A) \rightarrow B$	$:\{N_B\}_{K_B}$			

Table 3.2: A man-in-the-middle attack (left) and a replay attack (right)

failure. It is outside the scope of this chapter to illustrate different authentication flaws (see Lowe [133] and Cremers et al. [75] for more on authentication notions).

Suppose we use a long timeout for B, i.e. satisfying:

$$B.\texttt{time\_out} \geq \quad Intruder.\texttt{time\_decrypt} + Intruder.\texttt{time\_encrypt} + A.\texttt{time\_encrypt} + A.\texttt{time\_decrypt}$$

Here, UPPAAL finds a man-in-the-middle attack, presented on the left hand side of Table 3.2. This attack is similar to Lowe's attack [131], in which an attacker fools B into thinking he is communicating with A, while in reality A only talks to I. Of course, we could patch the protocol as Lowe did. But, in the context of time, it is interesting to model-check the protocol with a *tighter* timeout, i.e. $B.\texttt{time\_out} < Intruder.\texttt{time\_decrypt} + Intruder.\texttt{time\_encrypt} + A.\texttt{time\_encrypt} + A.\texttt{time\_decrypt}$. When this constraint is verified, the man-in-the-middle attack vanishes. Of course, we cannot pretend that B knows the intruder's times of encryption and decryption. Nevertheless, B can set $B.\texttt{time\_out} = A.\texttt{time\_encrypt} + A.\texttt{time\_decrypt}$, leaving *no* space for any interruption.

A second attack which is independent of timeouts (even if we set $B.\texttt{time\_out} = 0$!) was also found by UPPAAL; this time, the vulnerability is much simpler. We report it on the right hand side of Table 3.2. This attack corresponds to a "reflection" replay attack [160]. This attack occurs when the intruder simply replies B's message. The attacker fools B into thinking its communicating with himself, while it is not true in reality. Interestingly, suppose we change message 3 of the protocol to $3'$. $A \rightarrow B : \{N_B + 1\}_{K_B}$. Now, the above replay attack is prevented, since message 2 is not valid as message 3 anymore. Of course, a patch à la Lowe for *both* also prevents both problems, as shown in Protocol 3.4.

Message 1.	$A \rightarrow B : A$
Message 2.	$B \rightarrow A : \{B, N_B\}_{K_A}$
Message 3.	$A \rightarrow B : \{N_B\}_{K_B}$

Protocol 3.4: Patched Single Authentication Protocol

Having found confirmation that our framework is capable of finding *untimed* attacks (and thus confirming known attacks), we proceed to provide a good baseline to study extended security protocols with timing issues, like timeouts and retransmissions.

Retransmissions

Consider again the automaton for the responder, given in Figure 3.9. In state SR4, the responder sends the challenge $\{N_B\}_{K_A}$, and waits for a response in state SR5. If the response does not arrive before the timeout, the responder simply aborts. Now we consider possible retransmissions that allow the protocol to recover and continue its execution. With timed automata, retransmissions are easy to model by adding transition arrows from state SR5 to previous states of the automaton (the dashed lines in Figure 3.9); These transitions are guarded, allowing to perform the action only when the timeout is reached (i.e., $\texttt{t} >= \texttt{time\_out}$). A further refinement not explored here would be to add counters so that the number of retransmissions can be limited before aborting.

We consider two potential target states for the timeout of the Responder in SR5, namely states SR3 and SR2. Choosing the former corresponds to retransmitting the exact same message that was sent before, $\{N_B\}_{K_A}$. On the other hand, linking the retransmission arrow to SR2 corresponds to recomputing the whole message, by creating a new nonce N'_B and sending $\{N'_B\}_{K_A}$.

We implemented both strategies in our UPPAAL model. As can be expected, retransmitting the exact message *once* has the effect of *duplicating* the timeout for B, and thus the man-in-the-middle attack becomes possible even for tight timeout values. On the other hand, recomputing the whole message preserves the security of the protocol, at a higher computational cost. This provides evidence that indeed these design decisions are important for both security and efficiency, and a careful analysis can help to choose the best timeouts and retransmissions for a practical implementation.

3.4.2 A Real Protocol

Having illustrated our approach with a simple example we now study a more realistic protocol, the Yahalom protocol [71], shown as Protocol 3.5. This protocol aims at authentication of A and B as well as key distribution of A and B using a shared server S with whom both A and B share secret keys K_{AS} and K_{BS}.

Our choice is based on the fact that Yahalom is a complex and strong protocol, with no known attacks over it (However, a modification proposed by Abadi et al. [65] has a known type-flaw attack). Our aim is to study the protocol in more detail (and thus closer to an implementation) with timing information.

Message 1.	$A \rightarrow B : A, N_A$
Message 2.	$B \rightarrow S : B, \{N_B, A, N_A\}_{K_{BS}}$
Message 3.	$S \rightarrow A : \{B, K_{AB}, N_A, N_B\}_{K_{AS}}, \{A, K_{AB}, N_B\}_{K_{BS}}$
Message 4.	$A \rightarrow B : \{A, K_{AB}, N_B\}_{K_{BS}}, \{N_B\}_{K_{AB}}$

Protocol 3.5: Yahalom protocol

Here we use symmetric encryption, and key K_{XY} is shared between X and Y.

To model concatenation in an efficient way, we gathered several message components into a 16 bit field, thus keeping the state space as small as possible. In our case, we assume that nonces have 4 bits, principal id's 2 bits and keys 4 bits. To access these values, we use

bit-wise `and` with appropriate masks, and (left,right) bit shifts. Our intruder has also the capability to do the shifts and mask, and we also removed the "public key" choice from the intruder of Figure 3.8. We have modelled the protocol in UPPAAL (the initiator, responder, server and intruder are shown in Figures 3.11 and 3.12).

As we did with the previous protocol, first we check whether authentication of A to B could be falsified, using property AUT from Table 3.1. This property is not satisfied, confirming that Yahalom is secure. Now we move to study time sensitive issues.

There are two places in which timeouts and retransmissions can occur in this protocol. The first one is in Message 1: After A sends her message, she starts a timer waiting for message 3. Now, suppose that a timeout occurs, and A wants to retransmit her message. We can be confident that resending the same nonce N_A will *not* affect security, since in any case it was already sent in the clear in the first time. However, an interesting timing issue arises here. An answer that is received *too early* by A could be suspicious, because some time must pass while B talks to S. If A knows B's and S's encryption and decryption times, A could even deliberately "hibernate" (e.g. to save energy) until the response is likely to arrive (this models a windowed timeout, see Figure 3.2 (ii)). We model checked this property by measuring the time after A sends her message, and a response arrive (we count `ticks`, the dashed loop transition of the initiator in Figure 3.11). The specified property is AUT_y, shown in Table 3.1. This property is not satisfied, confirming that there is no way that the initiator can receive a valid answer *before* the time required by the responder and server to process A's request. In an implementation, it is reasonable for A to set a timeout like above, since it is realistic to assume that A can know the responder and server's times of encryption and decryption.

The second timeout is set by B after sending his message at step 2. If a timeout occurs, the retransmission decision is more delicate: It is not clear whether B should resend the original message, should recompute N_B or whether B should abort, since clearly N_A cannot be recomputed. Intuitively, N_B could be reused. We modelled in UPPAAL the retransmission of the exact message (as the dashed transition of the responder in Figure 3.11). When we model check again property AUT_y, we obtain that it is still unreachable, confirming that in that case an efficient retransmission of the same message 2 by B is secure.

However, by observing the messages flow, we know that if B timeouts then it is very likely that A has also reached its timeout and aborted (see Figure 3.2 (iv)). This mainly happens because since A is unsure whether B is alive or not, and thus A's timeout needs to be tight. If A knew that B is alive and waiting for an answer from S, then A could *extend* its timeout. We then sketch a more efficient implementation in Figure 3.13, in which at Message 2 B also sends a special $subm$ message notifying A of the submission to S. Then A can extend its timeout with more confidence (the second dashed line in Figure 3.13). In the case $subm$ is never received by A, she can send an *abort* message to B. Of course, in this simple model the attacker can also send this messages, thus performing denial of service attacks; in any case, our attacker is powerful enough to stop communication altogether.

In summary, for the Yahalom protocol we obtain that retransmitting for the responder is secure, and also that the initiator can be implemented to efficiently "hibernate" a safe amount of time before receiving a response.

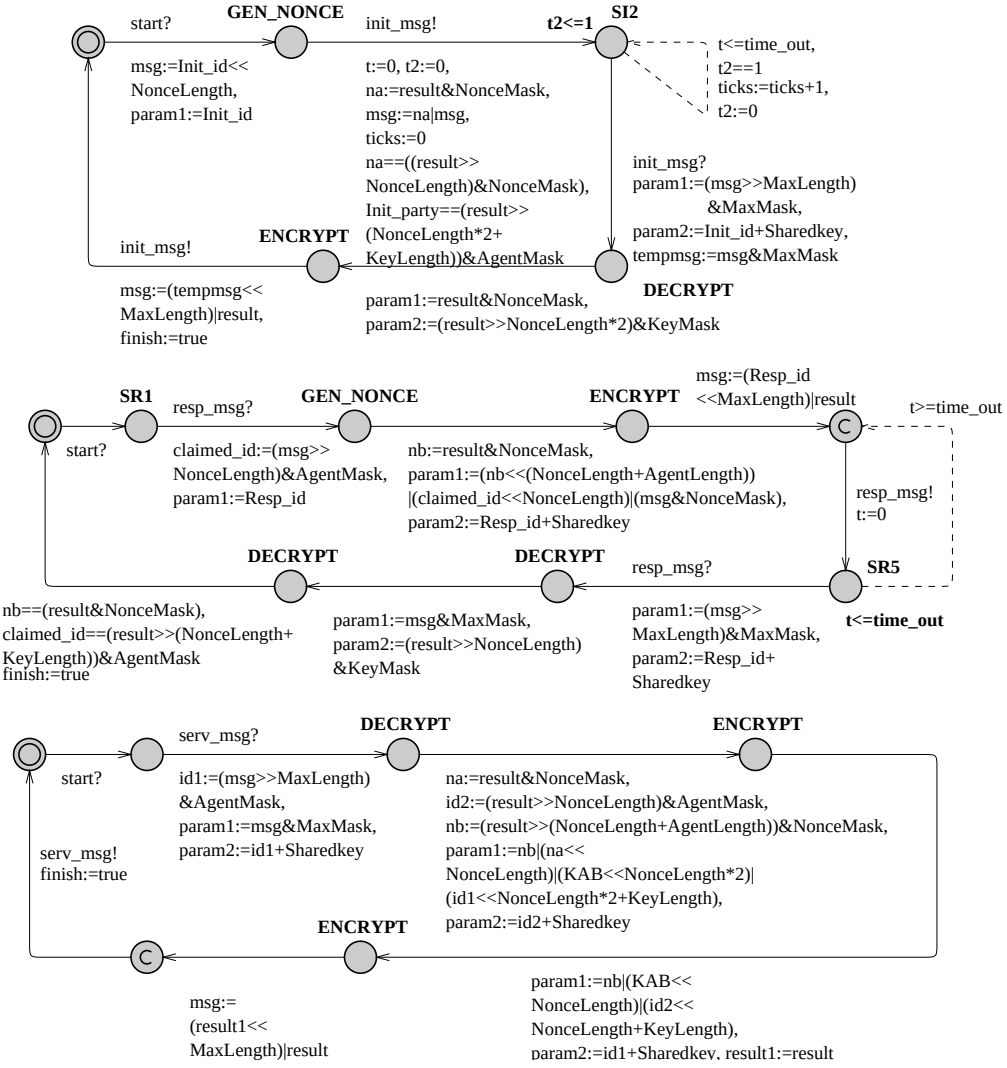


Figure 3.11: Timed automata schemas for the Yahalom Initiator (top), Responder (middle) and Server (bottom)

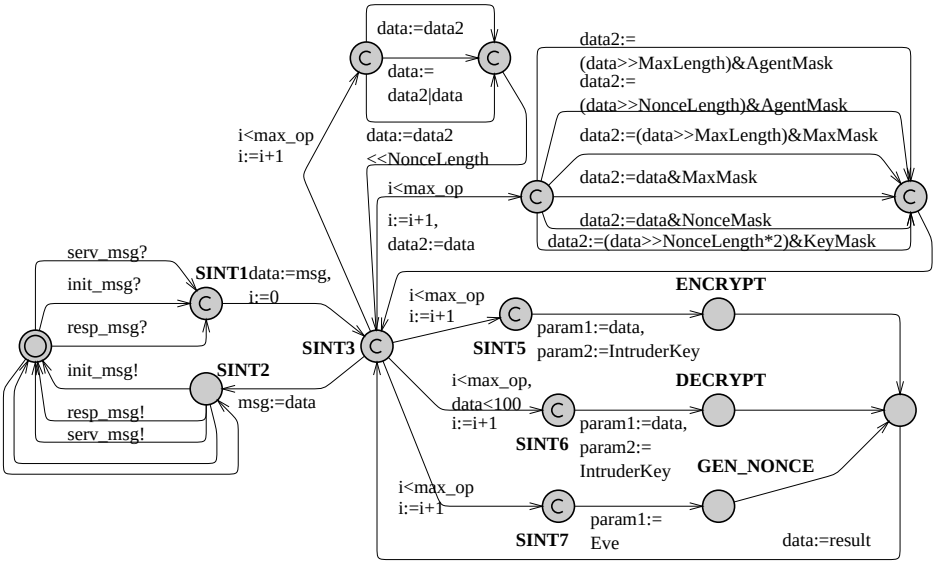


Figure 3.12: Timed automata schemas for the Intruder

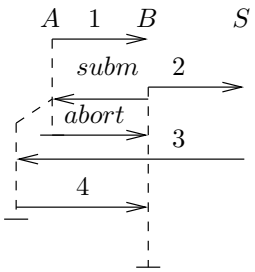


Figure 3.13: A more detailed implementation of Yahalom

3.5 Taxonomy of Message Flows and Timeouts

The flow of messages of many protocols follow a small set of specific patterns. By exploring the well known Clark Jacob library [71] of authentication protocols and the Security Protocols Open Repository (SPORE) [15], we were able to categorize the protocols in four categories, as shown in Figure 3.14. To each pattern, we add the corresponding timeouts, and analyze their impacts on security and efficiency. For the original references of the protocols, the interested reader may consult the Clark Jacob library [71] and the SPORE library [15].

Not shown in this categorization are non-interactive protocols which do not wait for messages and thus do not require timeouts. In this category fall the Wide Mouthed Frog protocol, the CCITT X.509 simple pass protocol and the CAM protocol for mobile IP.

First, we discuss the simplest pattern in Figure 3.14 (i). This is a three-message exchange with two participants. This pattern is the simplest and also the most secure one from timing point of view, since timeouts can be set tight, due to the ping-pong nature of the exchanges. To this pattern correspond both the example protocol of Section 3.4.1 and the one in Section 3.6.1, and also the protocols CCITT.509 three pass, the Shamir Rivest and Adleman Three-Pass protocol, the ISO XXX Key Three-Pass (and their repeated protocols), the SmartRight view-only protocol (from SPORE) and the Diffie-Hellman key exchange protocol. With a fourth message from B to A in the same fashion we find the Andrew Secure RPC protocol. Adding a third participant S, but still doing ping-pong exchanges, we can add the Needham Schroeder symmetric key protocol and the Denning Sacco protocol.

Secondly, we identify three-party protocols, in which a server S also takes part in the communication (Figure 3.14 (ii)) but ping-pong exchanges are not anymore used. This pattern is potentially unsafe and inefficient for A, since she has to wait until a long timeout as elapsed after the first message before receiving an answer from S. This is due to the fact that three messages have to be exchanged after A's initial message. By consulting again the Clark and Jacob library and the SPORE repository, we see that the Otway Rees protocol, the Gong mutual authentication protocol, the Woo-Lam mutual authentication protocol, the Carlsen protocol, and finally the Kehne Schoenwalder Langendorfer (KSL) protocol all fall in this category. Adding ping-pong exchanges before and after the exchanges of Figure 3.14 we find the Needham Schroeder Public key protocol. Adding a ping-pong exchange before Figure 3.14, and removing the last exchange gives us the SPLICE/AS protocol.

Thirdly, we see a pattern to which only the Kao Chow protocol belongs in Figure 3.14 (iii). This pattern is however better than (ii), since shorter and fewer timeouts are used: A needs to wait for the timeout corresponding to only two messages (instead of three as in (ii)), and B has to wait for only one timeout (comparing to two timeouts in (ii)).

Finally, in Figure 3.14 (iv) we see the last pattern. This pattern is worse than (iii) since B needs to wait longer (for two messages instead of one in (iii)). However, it is unclear whether it is better than (ii), which uses two timeouts of one message each: the actual efficiency and security depends on the actual timeout values used in each case. This category is inhabited by the Yahalom and Neuman Stubblebine protocols.

This taxonomy shows how authentication protocols can be categorized into a handful of patterns. The efficiency and security that an implementation of a protocol will have depends on which pattern the protocol follows, and thus it is useful to analyze the patterns in isolation from the actual protocols.

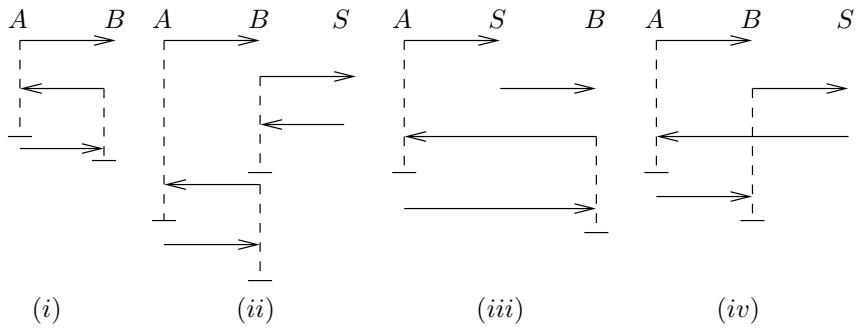


Figure 3.14: Typical message flows for authentication protocols

3.6 Beyond Model Checking

So far our method has been used for analysis purposes, i.e. to model, classify and debug security protocols as a source of hints for the improvement of the protocol implementations. We now explore some ideas to improve the protocols themselves, and also present the threat of a more subtle attack, based only on timing.

3.6.1 Using Time as Information: Timed Challenges

Sometimes it can be useful to include other timed information than timestamps, *even* if the clocks are not synchronized. Consider now Protocol 3.6, obtained by omitting the encryption of the last message of the (patched) protocol of Section 3.4.1. Even though N_B is now sent

Message 1. $A \rightarrow B : A$
Message 2. $B \rightarrow A : \{B, N_B\}_{K_A}$
Message 3. $A \rightarrow B : N_B$

Protocol 3.6: Modified Protocol from Section 3.4.1

in the clear, this protocol still achieves authentication of A to B, although now the nonce obviously cannot be regarded as a shared secret. Still, the intruder can prevent a successful run of the protocol (e.g. by intercepting message 3), hence the protocol is as strong as it was before in this respect.

Imagine now a situation in which there is a link from A to B in which data can be sent very fast, but at a high cost per bit sent. For example, think that the high cost of sending information comes from the fact that we have devices with a very limited amount of energy, like wireless sensor networks for instance. Alternatively, in some networks, operators charge according to quality of service, and many networks have asymmetric links (e.g. Cable modem and ADSL).

Assume therefore that, sending N_B in message 3 is expensive and not desirable. We propose a solution based exclusively on using *time* as information. Let δ_{AB} be the average

time it takes for a message to be sent from A to B, and analogously δ_{BA}. Then consider the "timed" variant of the above protocol, demonstrating how timing information is brought back to the (abstract) protocol level (i.e. Step 3 of Section 3.3), shown as Protocol 3.7. In Message

Message 1.	$A \rightarrow B : A$
Message 2.	$B \rightarrow A : \{B, t_B\}_{K_A}$
Message 3.	$A \rightarrow B$: "ack" at time $t_B - \delta_{AB} - \delta_{BA}$

Protocol 3.7: Single Acknowledgement Authentication Protocol using timed challenges

2, instead of a nonce, B generates some random time value $t_B > \delta_{AB} + \delta_{BA}$, concatenates it with B's identity and encrypts the message with A's public key. Then, B starts a timer t and sends the message. Upon reception, A extracts t_B, *waits* time $t_B - \delta_{AB} - \delta_{BA}$, and replies the single bit message "ack". When B receives this message, the timer is stopped and B checks that t is *sufficiently* close to t_B; if so, A is authenticated to B. Of course, the amount of noise in the time measurements influences what we mean by "*sufficiently* close" above. Also, to be realistic, the length in bits of t_B should be small enough, otherwise B would be waiting too long; this would give an intruder the chance to guess t_B, and answer the "ack" at the appropriate time. However, we can strengthen the protocol as follows in Protocol 3.8.

Message 1.	$A \rightarrow B : A$
Message 2.	$B \rightarrow A : \{B, t_{B_1}, \ldots, t_{B_n}\}_{K_A}$
Message 3.	$A \rightarrow B$: "ack" at time $t_{B_1} - \delta_{AB} - \delta_{BA}$
$\vdots$	
Message $n+2$.	$A \rightarrow B$: "ack" at time $t_{B_n} - \delta_{AB} - \delta_{BA}$

Protocol 3.8: Multi Acknowledgement Authentication Protocol using timed challenges

For example, if t_{B_i} is of length 4 bits, for $i \in [1..n]$, then the total answer is n bits, in comparison with an answer of $4n$ bits required in the nonce protocol.

Of course, sending several short messages can be worse than sending one long message, in which case our protocol would not be so useful. In general, the value of n must be chosen as small as possible, depending on the desired security and network latency. A fast network allows us to reduce n and at the same time increment the length of t_{B_i}, for $i \in [1..n]$.

Intuitively, the sent times of the "ack"'s represent information, and the above protocols exploit that. To the best of our knowledge, this is a novel usage of time in security protocols.

Application This protocol can be used to authenticate a whole chain of network packets, as follows. Suppose A has a large sequence of n packets which must be streamed to B over a network. For instance, these packets can represent an audio stream in the Internet. We want to authenticate the audio stream, but we do not wish to spend lots of resources on doing this. Let $t_{B_i} \in \{\delta_{AB} + \delta_{BA}, \delta_{AB} + \delta_{BA} + C\}$ for some constant C and p_i denote packet i, for $1 \leq i \leq n$. The new protocol is shown as Protocol 3.9.

Message 1.	$A \rightarrow B : A, n$
Message 2.	$B \rightarrow A : \{B, t_{B_1}, \ldots, t_{B_n}\}_{K_A}$
Message 3.	$A \rightarrow B : p_1$ at time $t_{B_1} - \delta_{AB} - \delta_{BA}$
$\vdots$	
Message $n+2$.	$A \rightarrow B : p_n$ at time $t_{B_n} - \delta_{AB} - \delta_{BA}$

Protocol 3.9: Application Protocol using timed challenges

When t_{B_i} is $\delta_{AB} + \delta_{BA}$, it is the delay introduced by A is zero, i.e. p_i is sent right away. However, when t_{B_i} is $\delta_{AB} + \delta_{BA} + C$, the delay is C. To be as efficient as possible, C should be chosen to be the minimum amount of time that allows B to distinguish the delay modulated by A.

In this protocol, only one bit is authenticated per packet. However, the larger the n is, the more confidence we can obtain of A's authentication.

Discussion In this protocol, we are in reality exploiting a well-known feature of channel coding: a so-called *timing covert channel*. In such a channel, the transmitting party modulates packets so that information can be passed even if its not allowed by the environment. Our usage differs in three ways:

- Firstly, we use a mixed approach, in which some information is sent in the standard channel, and other is sent in the timing channel.
- A second difference is more fundamental than the previous one. Our usage of timing channel is purposely public, and there is no environment trying to stop the unauthorized information flow. Timing is used only because of its practical advantages, namely low-bandwidth.
- Finally, in our protocol both communicating parties do not initially trust on the other's identity, in principle: Indeed, ours is an *authentication* protocol.

3.6.2 Timing Leaks in an Implementation of the Private Authentication Protocol

We now present a threat over implementation of security protocols with branching: the so called timing attack. We illustrate this by showing an attack over a careless implementation of Abadi's Private Authentication (PAP) protocol [19] (The *second* protocol). It is worthwhile to mention that the protocol has been proved correct by Abadi and Fournet [94], in a setting without time. We assume that each principal X has a set of communication parties S_X, listing the principals with whom X can communicate. The aim of the protocol is to allow an principal A to communicate *privately* with another principal B. Here "privately" means that no third party should be able to infer the identities of the parties taking part in the communication (i.e. A and B) (PAP Goal 1). Moreover, if A wants to communicate with B but $A \notin S_B$, the protocol should also conceal B's identity (and presence) to A (PAP Goal

1. A generates a nonce N_A. Then, A prepares a message $M = \{\text{“}hello\text{”}, N_A, K_A\}_{K_B}$, and broadcasts $(\text{“}hello\text{”}, M)$.

2. When a principal C receives message $(\text{“}hello\text{”}, M)$, it performs the following three steps:

   (a) C tries to decrypt M with its own private key. If the decryption fails, (which implies that $C \neq B$), then C creates a “decoy” message $\{N\}_K$ (creating a random K, and keeping K^{-1} secret), broadcasts $(\text{“}ack\text{”}, \{N\}_K)$ and finish its execution. If decryption succeeds, then $C = B$ (and so from now on we will refer to C as B). B then continues to the next step.

   (b) B checks that $A \in S_B$. If this fails, i.e. $A \notin S_B$, then B creates a “decoy” message $\{N\}_K$, broadcasts $(\text{“}ack\text{”}, \{N\}_K)$ and finishes execution. Otherwise B continues to the next step.

   (c) Finally, B generates a fresh nonce N_B, and broadcasts the message $(\text{“}ack\text{”}, \{\text{“}ack\text{”}, N_A, N_B, K_B\}_{K_A})$.

Protocol 3.10: Abadi's Private Authentication protocol

2). A run of the protocol in which A wants to communicate with B proceeds as shown in Protocol 3.10.

It is interesting to see the use of “decoy” messages, to prevent attacks in which an intruder I prepares a message $M = \{\text{“}hello\text{”}, N_C, K_A\}_{K_B}$, impersonating principal A. If decoy messages were not present, then I would send $(\text{“}hello\text{”}, M)$, and deduce whether $A \in S_B$ by noticing a response from B. However, using decoys only helps to confuse an attacker doing traffic analysis, and breaks down when considering a “timed” intruder, as we will show in the next section.

An Attack Over an Implementation of the Private Authentication protocol

We show an attack in which I can find whether $A \in S_B$. The attack is illustrated in Figure 3.15, where I is trying to attack A, B and C, which since I does not know their identities are called X, Y and Z. First, suppose that $I \notin S_B$ (the attack for the case in which $I \in S_B$ is analogous). First I needs to know how long, on average, it takes to B to compute each step of the protocol as described above. To discover this I could prepare various messages:

1. Firstly, I sends a message $(\text{“}hello\text{”}, \{N\}_K)$, where K is not the public key of any other participant. This would generate a number of decoy responses from the other participants, which I can time (Step 1 in Figure 3.15, for times x, y and z).

2. Secondly, I sends a message $(\text{“}hello\text{”}, \{\text{“}hello\text{”}, N_I, K_I\}_{K_B})$. Again, this generates decoy responses from the other parties which I can time (Step 2 in the figure). However, if B is present, then *one* response will have longer time (i.e. we get times x, z and y' with y' longer than y), reflecting the successful decryption and check that $I \notin S_B$ performed by B (Recall we assume that $I \notin S_B$). Up to this point, I has information

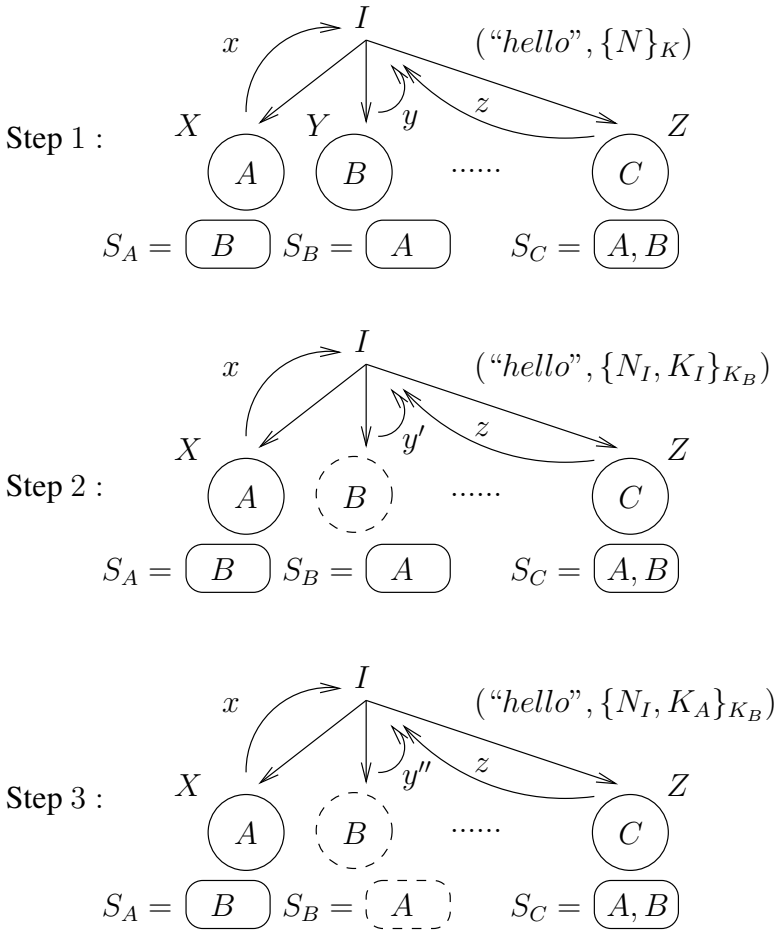


Figure 3.15: The attack over a time-careless implementation of the PAP protocol, where X, Y and Z are unknown identities by the intruder I. A, B and C are real identities, with corresponding S_A, S_B and S_C sets; x, y, y', y'' and z are timing values (Dashed circles indicate the intruder's knowledge of inner values)

that allows him to infer B's presence (hence the dashed circle in the figure); Thus, this attack already violates goal 2 of Abadi's requirements [19]: B should protect its presence if a party X is willing to communicate with B but $X \notin S_B$ (Step 2).

3. Finally, if B is present, then I sends message $(\text{“}hello\text{”}, \{\text{“}hello\text{”}, N_I, K_A\}_{K_B})$. This would generate again the same decoy responses, except one that takes longer (Step 3 in the figure, for x, z and y'' with y'' longer than y'). If this response takes the same time as the above item, then I can deduce that $A \notin S_B$. Otherwise, if the response takes longer (reflecting the nonce generation N_B and encryption performed by B) then I can deduce that $A \in S_B$.

If $I \in S_B$, then the second step above returns the *longest* time, and the third message would take either *less* time or equal.

After recording this information, I has three time values t_0, t_1 and t_2. t_0 corresponds to the time in which B is not present; t_1 corresponds to the time in which B is present but its communicating party $X \notin S_B$. Finally, t_2 corresponds to the case in which B is present and its communicating party $X \in S_B$. With these values at hand, now an attacker can check if $A \in S_B$ for an arbitrary A.

Timing in networks is often accurate but if the accuracy is too low, the intruder can repeat the timings (i), (ii) and (iii) and perform statistical analysis to increase the probability of the inferences to be correct [120]. We propose this as future work, when we have a probabilistic, timed model checker at disposal.

3.7 Related Work

Many approaches focus on the study of protocols that use timestamps [82, 89, 133, 42, 104]. Recent work of Delzanno et al. [82] presents an automatic procedure to verify protocols that use timestamps, like the Wide Mouthed Frog protocol. In their work, differently from ours, a global clock is assumed, and timeouts and retransmissions are not discussed. Evans and Schneider [89] present a framework for timed analysis. Differently from our (UPPAAL) model checking, it is based on a semi-decision procedure with discrete time. In that work, the usage of retransmissions is hinted at as future work, but not (yet) addressed. Lowe [133] also analyses protocols with timing information; his work shares with us the model checking approach, although Lowe's approach is based on a discrete time model. A global clock is also assumed, and timeouts nor retransmissions are addressed. Closer to ours is the work of Gorrieri et al. [104], in which a real-time process algebra is presented for the analysis of time-dependent properties. They focus on compositionality results, and no model checking is presented. Gorrieri et al. also show how timeouts can be modelled, although retransmissions are not discussed.

Regarding our timing attack upon Abadi's protocol, Focardi et al [93] develop formal models for Felten and Schneider's *web privacy* timing attack [90]; their modelling activity shares with our work the idea of using timed automata for analysis, although our attack illustrates a timing attack over a "pure" security protocol.

3.8 Conclusions

In this chapter we address some of the issues that need to be considered when including time-related parameters in the engineering process of a security protocol.

Our first contribution is a method for the design and analysis of security protocols that consider timing issues. We model security protocols using timed automata, and use UPPAAL to simulate, debug and verify security protocols in a real time setting. To this end, we employ a general Dolev Yao style intruder (naturally encoded in UPPAAL), and we remark that modelling the intruder as a timed automata implicitly extends its power to take into account the time sensitivity. Our method allows us to specify security protocols in detail, with timeouts and retransmissions. This increases the confidence in the analysis, since the modelled protocols are closer to their implementations than the classical analysis (e.g. CASPER [133] or the constraint-based methods of [149] and Chapter 2).

Secondly, by analyzing the protocols in the Clark and Jacob library and the SPORE library, we see that most protocols schemas (w.r.t. timeouts) fit into a small number of common patterns. We analyse the efficiency and security of each of the patterns. Still, as possible future work we would like to perform a full UPPAAL analysis of each of these literature protocols (just as we do for Yahalom in Section 3.4.2).

Other novel and more real-life protocols which are sensitive to timing issues (e.g. besides timeouts, use for instance puzzles) may benefit from our analysis, e.g. the Host-Identity-Protocol (HIP), initially analysed by Aura et al. [36].

Our third contribution is an illustration of the implicit information carried by timing. The mere act of sending a message at a specific moment in time, and not another, carries information. We propose a novel security protocol that exploits this fact to achieve authentication. The protocol replaces the standard nonces with timed challenges, which must be replied at specific moments in time to be successful. Although it is a preliminary idea, it exposes clearly the fact that security protocols can use and take advantage of time.

Finally we address threats specifically involving timing should also be considered; specifically, timing attacks. We illustrate these attacks in the context of security protocols, where branching allows an intruder to deduce information that is intended to be kept secret. Specifically, we mount an attack over an imperfect implementation of Abadi's private authentication protocol [19]. Solutions to avoid timing attacks in the implementations are usually expensive (e.g. noise injection or branch equalization), and it is not our purpose to investigate them. Here we merely lift the known problem of timing attacks, typically mounted against the cryptosystem to obtain secrets keys, to security protocols in general where the information leakage can be, in principle, anything.

One possible direction of future work is to consider a timed and probabilistic model checker (in the lines of [77] or [123]), that would allow us to study the protocols of Section 6. Moreover, a probabilistic setting would allow us to model, more realistically, the network latency. This, in turn, would provide us with a finer method to tune sensitive timing values. Another possible direction for future research would be to implement a compiler from a meta notation (similar to the standard notation, plus timing information) supporting symbolic terms, to UPPAAL automata.

CHAPTER 4

A Process Algebraic Analysis Model for Guessing Attacks

4.1 Introduction

As we already illustrated in Section 2.7, password protocols can be subject to guessing attacks. To analyze the security of protocols against these attacks, in that section we describe a procedure that searches for possible guessing attacks. The procedure is based on the Dolev Yao attacker which assumes ideal encryption, as described in the Introduction.

However, when studying password protocols in particular, relaxing ideal encryption to model closely the properties of the underlying encryption scheme is more realistic. This is so since in practice these protocols are designed to be secure when instantiated with a particular encryption scheme. This makes the security against dictionary attacks dependent on the chosen cryptosystem.

Typically, the security of an encryption scheme is characterized by certain properties that the ciphertexts satisfy. For instance, an encryption scheme is said to be *repetition concealing* [27] if an adversary cannot detect two instances of the same message encrypted with the same key, something that can be achieved using probabilistic [99] or stateful encryption is needed. Similarly, an encryption scheme is *which-key concealing* if an adversary cannot deduce if two messages are encrypted under the same key [27]. Besides these general properties, usually each particular cryptosystem has its own subtleties that can also provide useful information to an adversary. For example, a public key in RSA consisting of a pair (n, e) can be distinguished from a random string because e is odd and n contains no small prime factors. As discussed by Mellovin and Merritt [47], this simple fact allows a dictionary attack over EKE (the *Encrypted Key Exchange* [47] protocol) when instantiated with RSA.

In this chapter, we take a different direction than the one proposed in Section 2.7, and study password protocols using the Applied Pi Calculus [23]. Our choice is based on the fact that this process calculus allows the modelling of relations between data (e.g. cryptographic details of the underlying cryptosystem) in a simple and precise manner using equational theories over the term algebra. We thus instantiate the general analysis model of the Introduction (shown in Figure 1.1) with the one illustrated in Figure 4.1. There, the property model is an

equivalence between processes (as detailed in Section 4.2.5), while the attacker is thought as an arbitrary process running in parallel with the protocol process representing the environment model, which is the parallel composition of the (sequential) protocol participants' processes. The considered attacker is stronger than the basic Dolev Yao attacker since it can exploit particular relations between the messages (see Section 4.2.1) by using particular equational theories stating the message relations.

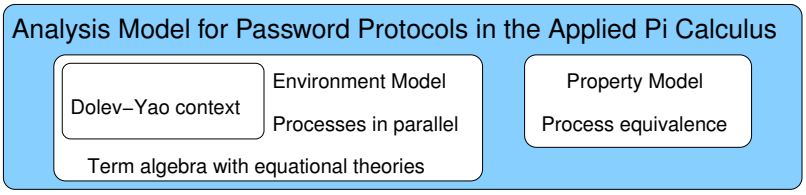


Figure 4.1: Analysis model for Password protocols in the Applied Pi Calculus

The contribution of this chapter is twofold: First, we show how to analyse, in a precise formal framework, the security of password protocols when they are instantiated with particular encryption schemes which may or may not satisfy specific properties. We model (most of) these properties by extending the equational theory of the Applied Pi Calculus. In particular, we show how to model encryption schemes which are repetition and which-key revealing, and also encryption schemes that allow an adversary to distinguish ciphertexts and public keys from random noise. Second, we study, as illustrating examples, two well-known protocols: the EPT protocol of Halevi and Krawczyk [107], and the already mentioned EKE protocol [47]. For EPT, we show that security against dictionary attacks is achieved when encryption is repetition concealing. For the EKE protocol we show that security can be established if encryption is which-key concealing, and ciphertexts and public keys are indistinguishable from random noise. Interestingly, our analysis helped us to identify a vulnerability of EKE that arises when ciphertexts are identifiable. To solve this, we propose a simple modification that (to the best of our knowledge) is novel.

Of course, assuming such a realistic scenario does not come for free: our security proofs are manual (however automatic proofs are possible, see the Conclusion), while the approach of Section 2.7 provides an automatic procedure.

4.2 Applied Pi Calculus

We first *very* briefly summarize the Applied Pi Calculus from Abadi and Fournet, borrowing from [23] (with definitions copied *verbatim*). The process grammar is similar to the one in the pi calculus, differing in the fact that messages can contain *terms* and names need not be just channel names:

$$P, Q ::= 0 \mid P|Q \mid !P \mid \nu n.P \mid if\ U = V\ then\ P\ else\ Q \mid u(x).P \mid \overline{u}\langle V\rangle.P$$

Here U and V belong to the term grammar, which we describe along with its equational theory later in Section 4.2.1.

In this calculus, plain processes as defined above are extended with *active substitutions* $\{x = V\}$, meaning the replacement of variable x with the term V. $\{x = V\}$ can represent the situation in which a term V has been sent to the environment, but the environment may not have the atomic names that appear in V. Still, the environment can refer to V by using x.

We use the standard notions of equivalences between processes, *reduction* ($\rightarrow$), structural equivalences ($\equiv$) and observational equivalence ($\approx$) as defined in [23] (we do not define them here, due to space constraints and the fact that we will mainly use *static equivalence*, described below).

Frames, ranging over $\phi, \varphi, ...$ are processes built up from active substitutions by parallel composition and restriction. The domain $dom(\phi)$ of a frame ϕ are all the variables that the frame exports, i.e. the variables such that $\{x = V\}$ is in ϕ and x is not restricted. Given a term M, we denote $fn(M)$ as the set of free names appearing in M. Now we can define when two terms are equal in a given frame (this definition and the next one follow from [23]).

Definition 4.2.1. *We say that two terms M and N are equal in the frame φ, and write $(M = N)\varphi$, if and only if $\varphi \equiv \nu\widetilde{n}.\sigma$, $M\sigma = N\sigma$[1] and $\{\widetilde{n}\} \cap (fn(M) \cup fn(N)) = \emptyset$ for some names $\widetilde{n}$ and substitution σ.*

Example 4.2.2. *We borrow an example from [23]. Let f and g be two functions with no equations (they can be thought as two independent one-way functions). Let $\phi_0 = \nu K, s.\{x = K, y = s\}$, $\phi_1 = \nu K.\{x = \mathsf{f}(K), y = \mathsf{g}(K)\}$, and $\phi_2 = \nu K.\{x = K, y = \mathsf{f}(K)\}$. In ϕ_0, x and y are unrelated, and hence they cannot be distinguished by any pair of terms M and N. The same happens in ϕ_1. However, in ϕ_2, x and y are related: y is obtained from x by applying f. More formally, let $M = \mathsf{f}(x)$ and $N = y$. Then $(M = N)\phi_2$ but* not *$(M = N)\phi_i$ for $i \in [0, 1]$.*

The example motivates the definition of *static equivalence* $\approx_s$:

Definition 4.2.3. *We say that two closed (i.e. with no free variables) frames φ and ψ are* statically equivalent*, and write $\varphi \approx_s \psi$, when $dom(\varphi) = dom(\psi)$ and when, for all terms M and N, we have $(M = N)\varphi$ if and only if $(M = N)\psi$.*

Given two closed extended processes A and B, we can "extract" their frames $\phi(A)$ and $\phi(B)$, by replacing every plain subprocess of A and B with $\mathbf{0}$. We then say that two closed extended processes are statically equivalent, and write $A \approx_s B$, when their frames are statically equivalent. We quote a useful lemma from [23].

Lemma 4.2.4 (Lemma 1 and 2 from [23])**.** .

1. *Observational equivalence and static equivalence coincide on frames. Observational equivalence is strictly finer than static equivalence on extended processes: $\approx \subset \approx_s$.*

2. *Static equivalence is closed under reduction and structural equivalence.*

Now we are ready to instantiate the calculus with our term grammar and equational theory.

[1] This equality is modulo the equational theory.

U, V	$::=$	terms				
	$a, n, r, N_A, K, \ldots$	name				
	$x, y, ...$	variable				
	(U, V)	pair	$\mathsf{sdec}(\{x\}^r_y, y)$	$=$	x	
	$\mathsf{fst}(U)$	field selector	$\mathsf{pdec}(\{x\}^{r,\rightarrow}_{pk(y)}, y)$	$=$	x	
	$\mathsf{snd}(U)$	field selector	$\mathsf{fst}((x, y))$	$=$	x	
	$pk(U)$	public-key derivation	$\mathsf{snd}((x, y))$	$=$	y	
	$h(U)$	cryptographic hash				
	$\{U\}^r_V$	symmetric encryption				
	$\{U\}^{r,\rightarrow}_V$	public-key encryption				
	$\mathsf{sdec}(U, V)$	symmetric decryption				
	$\mathsf{pdec}(U, V)$	public-key decryption				

Table 4.1: Grammar for terms (left) and Equational theory EQ_0 (right)

4.2.1 Equational Theories for Standard Adversary Abilities

In this section we first instantiate the Applied Pi Calculus with our term grammar. Then, we present the equational theory EQ_0 that represents the standard adversary abilities. Afterwards, we extend EQ_0 with EQ_1 and EQ_2, two equational theories that provide further adversary abilities. Our grammar for terms is shown in Table 4.1 (left) (This grammar is similar to Table 2.1 of Chapter 2). Besides names and variables, we have the usual pairing constructor, along with its projections. Given a name K representing a private key (and thus usually appearing restricted) we can derive a public-key $\mathsf{pk}(K)$ that can then be used for (public key) encryption[2]. We also define the usual hash constructor. Our constructors for encryption (both symmetric and asymmetric) take a name r as *randomness* parameter. This allows us to model probabilistic encryption. By explicitly considering the randomness parameter as a name, say r, we can model easily repetition concealing vs. revealing cryptosystems by simply restricting or not r (This modelling was already suggested in [23]). On the other hand, decryption is deterministic.

The standard equational theory EQ_0 is shown[3] in Table 4.1 (right). In EQ_0 we encode all the standard abilities of the adversary, given by decryption identities and pair projection. Here, $pk(\cdot)$ and $h(\cdot)$ are modelled as non-reversible operations by implicitly not adding any ability in EQ_0.

Under the standard equational theory EQ_0, non-deterministic encryption behaves as a "secure envelope", modelling a repetition concealing cryptosystem. Let us illustrate the practical implications of this with one example. Consider $P(M) \doteq \nu r, K.\{x = \{M\}^{r,\rightarrow}_{\mathsf{pk}(K)}, y = pk(K)\}$. Process $P(M)$ exports (in x) the encryption of M with $pk(K)$, and also exports (in y) the used public key. Now, the property of "secure envelope" can be stated as $\forall M : P(M) \approx_s \nu s, k.\{x = s, y = k\}$, which intuitively holds since r is an unguessable

[2]However, note that in general it may be not possible to obtain the public key from a previously created private key: Typically, the key pair is created *simultaneously*. Thus, when a private key K is created using restriction ν, we assume that the corresponding public key is also created. Then, the constructor $pk(K)$ is just a pointer to this public key.

[3]Excluding all the equations obtained from reflexivity, symmetry, transitivity and substitution of terms for the variables x, y and r.

$\mathit{sym\_ciphertext}(\{x\}_y^r)$	$=$	true	$\mathit{same\_k}(\{x\}_k^r, \{y\}_k^s)$	$=$	true
$\mathit{pk\_ciphertext}(\{x\}_y^{r,\rightarrow})$	$=$	true	$\mathit{same\_k}(\{x\}_k^{r,\rightarrow}, \{y\}_k^{s,\rightarrow})$	$=$	true
$\mathit{valid\_pk}(pk(x))$	$=$	true			

Table 4.2: Extended equational theories: EQ_1 (left) and EQ_2 (right)

seed and K is secret. This expresses that the resulting ciphertext of encrypting a message M with a public key derived from K is indistinguishable from random "noise" (analogously for the public key derived from K). Notice that this holds even if M is possibly known (or guessable) by the environment.

However, one could argue that this approach models a too strong, often unrealistic encryption mechanism, in at least three ways:

(A1) First, in particular cryptosystems, it can be the case that ciphertexts are distinguishable from pure random noise, even though the plaintext or encrypting key is not leaked. For example, a usual indication of the presence of a ciphertext can be found in the length of the messages (this can happen, for example, when padding is weak or non-existent). In block ciphers, for instance, ciphertexts typically consist of a certain number of blocks (e.g. a multiple of 128/256 bits). Similarly, numbers close to each other in RSA also give a good indication of a ciphertext. As another example, in the McEliece cryptosystem [140] every ciphertext is a *codeword*, with a small vector error added to it. This makes ciphertexts distinguishable from random noise.

(A2) Second, and similar to item 1 above, public keys can also be distinguishable from random noise, even if the private key is kept secret. As an example (already mentioned in the introduction), a public key in RSA consists of two large naturals n and e, where e is always odd and n does not contain small prime factors.

(A3) Finally, encryption could be which-key *revealing*, allowing an adversary not only to detect ciphertexts (as in item 1 above) but also to deduce if two ciphertexts were encrypted under the same key.

4.2.2 Equational Theories for Extended Cryptographic Relations

To study the security of protocols under these more realistic scenarios, we first need to model (A1), (A2) and (A3) as adversary abilities. We achieve this with the equational theories of Table 4.2.

In EQ_1, we model the ability of an adversary to distinguish ciphertexts and public keys from regular messages: *sym_ciphertext* detects ciphertexts created with symmetric encryption, while *pk_ciphertext* recognizes public-key ciphertexts. Similarly, *valid_pk* detects public keys. Now, $\mathrm{EQ}_0 \cup \mathrm{EQ}_1$ models an adversary whose abilities include (A1) and (A2).

In EQ_2, we add the ability to deduce whether two messages are encrypted under the same key (with equality *same_k*.), modelling (A3).

When the equational theory is $\mathrm{EQ}_0 \cup \mathrm{EQ}_1 \cup \mathrm{EQ}_2$, the secure envelope property does not hold anymore: x can be detected by *pk_ciphertext*, y by *valid_pk* and finally x and y in

conjunction can be recognized by *same_k*. However, a weaker form of a "secure envelope" still holds, namely

$$\forall M, M' : P(M) \approx_s P(M') \tag{4.1}$$

which states that even though an adversary can recognize a ciphertext $\{\cdot\}^{r,\rightarrow}_{pk(K)}$ and its encrypting public key $pk(K)$, the adversary can still not glean any information about the plaintext M. Thus, this weaker notion of secure envelope is reduced to express secrecy of M.

4.2.3 Modelling Guessing Attacks

In an off-line dictionary attack, the adversary guesses the (weak) shared password and then tries to verify the guess with the eavesdropped session [135]. Thus, we can regard a protocol as secure if it provides no such verification possibility to the adversary. As we already mentioned, in the Applied Pi Calculus the information gathered by the adversary can be characterized by using *frames*. Let the frame φ represent the information of an eavesdropped session and let K be the shared, weak password (K is free in φ, representing the fact that K can be "guessed" by the adversary). Then, we can represent the notion of security of a password protocol against dictionary attacks by checking whether the adversary can distinguish φ from $\nu K.\varphi$, in which K is bound, representing unguessability. More precisely, by "seeing" we use static equivalence ($\approx_s$), as defined in Definition 4.2.3.

Definition 4.2.5. *Let ϕ be a frame in which K is free. Then we say that ϕ verifies a guess of K if $\phi \not\approx_s \nu K.\phi$. Conversely, we say that ϕ is secure w.r.t. K if $\phi \approx_s \nu K.\phi$.*

Intuitively, a distinction of φ and $\nu K.\varphi$ models the situation in which the adversary "hits" the correct guess, and he can verify that fact by using φ. On the other hand, if φ and $\nu K.\varphi$ are indistinguishable, then an adversary has no way of verifying from φ that a given word (from e.g. a dictionary) is actually the correct password.

For example, if $\phi = \nu Na.\{x = (Na, K)\}$, then $\nu K.\phi \not\approx_s \phi$. Intuitively, this follows from the fact that K is free (guessable!) in ϕ, and thus can be used in a term to distinguish x from random noise. To see this formally, we let $M = \mathsf{snd}(x)$ and $N = K$. Then $(M = N)\phi$ but not $(M = N)\nu K.\phi$. On the other hand, if $\phi = \nu Na.\{x = \{Na\}^r_K\}$ then $\nu K.\phi \approx_s \phi$. To see the $(\Rightarrow)$ of this claim is easy; however the converse is much more difficult. Intuitively, we need to see that using K does not help in the equality of M and N in ϕ. We can assume that Na does not occur in M or N (because anyway we can rewrite Na to Na' in ϕ, by alpha conversion). The only case in which M and N can use K in ϕ is by decrypting x; for example, $M = \mathsf{sdec}(x, K)$. But in this case N cannot be Na, and so for $(M = N)\phi$ to hold we must set $N = M = \mathsf{sdec}(x, K)$. But this implies that $(M = N)\nu K.\phi$ also, by alpha converting K (to some suitable K') in ϕ.

4.3 Case Study 1: Encrypted Password Transmission

In this section we study the *Encrypted Password Transmission* (EPT) protocol [107]. We first present the protocol, then translate it into the calculus and finally analyse the security against dictionary attacks.

EPT is designed to be run between a server S and a user U. We assume that S and U share a weak password P, and that the server S has a strong public-private key pair. Also, U

has stored a hash of S's public key, which has been previously securely communicated. The goal of the protocol is to authenticate U to S, as shown in Protocol 4.1.

Message 1.	$S \to U : (N, pk(K_S))$
Message 2.	$U \to S : \{(N, P)\}^{r,\rightarrow}_{pk(K_S)}$

Protocol 4.1: Encrypted Password Transmission (EPT) protocol

The protocol proceeds as follows: First, the server sends in the clear to the user a message consisting of a random challenge ("nonce") N and his public key $pk(K_S)$ (EPT.1). Then, the user checks that the received public key, when hashed, matches with his own (stored) hashed copy of the key. If it does not, then the user aborts. Otherwise, the user answers by encrypting a pair of N and P with the server's public key (EPT.2).

4.3.1 Translation in the Calculus

First, we translate the user and server into appropriate processes. Let c_{SU} be a channel name for communication from S to U and c_{US} a channel for communication from U to S. By keeping c_{ij} free in the processes U and S, we allow adversaries (i.e. the environment) to eavesdrop on these channels. We define S as:

$$\begin{aligned} S \quad \doteq \quad & \nu K_S, N.(\overline{c_{SU}}\langle (N, pk(K_S))\rangle \,.\, c_{US}(y). \\ & \qquad \text{if } ((N, P) = \mathsf{pdec}(y, K_S)) \text{ then } P_S) \mid \{pks = pk(K_S)\} \end{aligned}$$

Here, we use $\mathsf{if}\ C\ \mathsf{then}\ P$ as syntactic sugar for the process $\mathsf{if}\ C\ \mathsf{then}\ P\ \mathsf{else}\ 0$. Process P_S models what happens after the session was established successfully. Note how the public key $pk(K_S)$ is exported in variable pks. We assume that none of the values used during the protocol appear in P_S, and also that P_S never discloses K_S. If the decryption fails, then the process would abort (executing the implicit **0** of the *else* branch). The user process U is:

$$\begin{aligned} U \quad \doteq \quad & \nu r.(\, c_{SU}(x).\ \text{if } (h(\mathsf{snd}(x)) = h(pks)) \text{ then} \\ & \qquad \overline{c_{US}}\langle \{(\mathsf{fst}(x), P)\}^{r,\rightarrow}_{\mathsf{snd}(x)}\rangle.P_U) \end{aligned}$$

Similarly, P_U is the process that the user executes after successful execution of the protocol (again, no value of the protocol appears in P_U). Now, a system of one user and one server can be setup by letting them share a password P: $\nu P.(U \mid S)$. A normal execution of this process can now be modelled by applying reductions and equivalences, as in [23]:

$$\begin{aligned} & \nu P.(U \mid S) \to\to\equiv (P_S \mid P_U \mid \nu P, K_S, N, r.\varphi) \\ & \varphi \doteq \{x = (N, pk(K_S)), y = \{(\mathsf{fst}(x), P)\}^{r,\rightarrow}_{\mathsf{snd}(x)}, pks = pk(K_S)\} \end{aligned}$$

The first two reductions come from the message communications (EPT.1) and (EPT.2), and the last equivalence corresponds to scope tightenings. Moreover, by structural equivalence we can rewrite φ and obtain:

$$\varphi \equiv \{x = (N, pk(K_S)), y = \{(N, P)\}^{r,\rightarrow}_{pk(K_S)}, pks = pk(K_S)\}$$

Intuitively, φ (along with its restrictions) represents the information that the environment has learnt from eavesdropping a run of EPT between a user U and a server S.

Next, we study whether the information recorded in φ can be exploited to mount an off-line dictionary attack.

4.3.2 Security Against Guessing Attacks

The following lemma states that EPT is secure, in the sense that it does not provide a verification of a guess of a password to an adversary. To model this, we let P be guessable by removing it from the restriction, and then compare the result to the case in which it is still restricted, as in our formalization of guessing attacks in Definition 4.2.5:

Lemma 4.3.1. *Let P_S and P_U be processes where the names P, K_S, N and r do not appear free. Then:*

$$(P_S \mid P_U) \mid \nu P, K_S, N, r.\varphi \approx (P_S \mid P_U) \mid \nu K_S, N, r.\varphi \tag{3}$$

Proof. (Sketch) We first establish the static equivalence:

$$\nu P, K_S, N, r.\varphi \approx_s \nu K_S, N, r.\varphi \tag{4}$$

Equivalence (4) is an instance of the following equivalence:

$$\nu N, K_S, r.\begin{pmatrix} \{x = (R, pk(K_S)), \\ y = \{U\}^{r,\rightarrow}_{pk(K_S)}, \\ pks = pk(K_S)\} \end{pmatrix} \approx_s \nu N, K_S, r.\begin{pmatrix} \{x = (R, pk(K_S)), \\ y = \{V\}^{r}_{pk(K_S)}, \\ pks = pk(K_S)\} \end{pmatrix} \tag{5}$$

Where R is a term s.t. $\{K_S, r\} \cap fn(R) = \emptyset$, and U and V are arbitrary terms. Equivalence (5) can be established similarly to Equivalence A.4 in Lemma 11 of [24] (there, constructor hello is similar to our pairing constructor). The following equivalence also holds:

$$\nu N, K_S, r.\begin{pmatrix} \{x = (R, pk(K_S)), \\ y = \{V\}^{r,\rightarrow}_{pk(K_S)}, \\ pks = pk(K_S)\} \end{pmatrix} \approx_s \nu P, N, K_S, r.\begin{pmatrix} \{x = (R, pk(K_S)), \\ y = \{V\}^{r}_{pk(K_S)}, \\ pks = pk(K_S)\} \end{pmatrix} \tag{6}$$

When $P \notin fn(R)$ and $P \notin fn(V)$. Finally consider the equivalence:

$$\nu P, N, K_S, r.\begin{pmatrix} \{x = (R, pk(K_S)), \\ y = \{W\}^{r,\rightarrow}_{pk(K_S)}, \\ pks = pk(K_S)\} \end{pmatrix} \approx_s \nu P, N, K_S, r.\begin{pmatrix} \{x = (R, pk(K_S)), \\ y = \{X\}^{r}_{pk(K_S)}, \\ pks = pk(K_S)\} \end{pmatrix} \tag{7}$$

For arbitrary terms W and X. This equivalence can be shown in the same way as (5).

Hence, we can instantiate $R = N, U = X = (N, P), V = W = (N, N)$ in (5), (6) and (7) to obtain (4). Note that this holds even when we consider as equational theory our most powerful adversary, with $EQ_0 \cup EQ_1 \cup EQ_2$.

Finally, by Lemma 4.2.4 (1), we lift the result to observational equivalence ($\approx$) to obtain (3). □

Intuitively, the lemma follows from the fact that r is an unguessable, freshly generated seed, and then the environment can never distinguish y in φ from noise.

While proving this lemma, the main cryptographic requirement of the protocol w.r.t. its underlying encryption is uncovered: it must be repetition concealing. Discovering an attack in the case of repetition revealing (and thus deterministic) encryption is not difficult. To model deterministic encryption, we remove r from the restriction operator, thus letting the environment control it. So, let ϕ_0 be $\nu P, K_S, N.\varphi$ and ϕ_1 be $\nu K_S, N.\varphi$. Then, $\phi_0 \not\approx_s \phi_1$. Let $M_1 = y$ and $M_2 = \{(\mathsf{fst}(x), P)\}^{r,\rightarrow}_{\mathsf{snd}(x)}$. Then, $(M_1 = M_2)\phi_1$ but not $(M_1 = M_2)\phi_0$. Thus, when encryption is not repetition concealing the protocol is not secure. This is in accordance with [107], where encryption is asked to be semantically secure [99][4].

The underlying encryption does not have to be which-key concealing to establish the security of EPT. In fact, all the abilities introduced by EQ_1 and EQ_2 do not affect (3). This makes EPT a robust protocol. Interestingly, in the next protocol (EKE) we analyse, the requirements are the other way around: repetition concealing is irrelevant, whilst the feature of being which-key concealing is crucial to establish security against dictionary attacks.

4.4 Case Study 2: Encrypted Key Exchange

In this section we analyse the Encrypted Key Exchange protocol, presented in [47]. The EKE protocol is designed to solve the problem of *authenticated key exchange* while being resistant against dictionary attacks.

Differently from the EPT protocol studied in the previous section, which required U to have a stored hashed copy of the server's public key, EKE (shown here as Protocol 4.2) is a password-only protocol, which assumes *only* a weak shared password in common.

Message 1.	$A \rightarrow B : \{pk(K)\}^r_P$
Message 2.	$B \rightarrow A : \{\{R\}^{t,\rightarrow}_{pk(K)}\}^s_P$
Message 3.	$A \rightarrow B : \{N_A\}^u_R$
Message 4.	$B \rightarrow A : \{(N_A, N_B)\}^v_R$
Message 5.	$A \rightarrow B : \{N_B\}^w_R$

Protocol 4.2: Encrypted Key Exchange (EKE) protocol

First, A generates a new private key K, and then derives the public key $pk(K)$. Then, A encrypts $pk(K)$ with the shared password P and sends it to B (EKE.1). Then, B extracts $pk(K)$, generates a fresh session key R and encrypts it with $pk(K)$. Then, B encrypts again the resulting message with P and sends it to A (EKE.2). The following three messages (EKE.i), $i = 3, 4, 5$, exchange nonces N_A and N_B to perform the "hand-shaking" necessary to defend against replay attacks.

[4] In fact, in [107] a stronger notion of security is required, necessary to resist active adversaries. We do not need that here, since we are dealing with passive adversaries only.

4.4.1 Translation in the Calculus

The user process A can be defined as:

$$\begin{aligned} A \doteq \quad & \nu K, r.(\overline{c_{AB}}\langle\{pk(K)\}^r_P\rangle \,.\, c_{BA}(x_1).(\nu k.(\{k = \mathsf{sdec}(\mathsf{pdec}(x_1, K), P)\} \\ & \mid (\nu u, N_A.\overline{c_{AB}}\langle\{N_A\}^u_k\rangle \,.\, c_{BA}(x_2) \,. \\ & \text{if } (N_A = \mathsf{fst}(\mathsf{sdec}(x_2, k))) \text{ then } \nu w.\overline{c_{AB}}\langle\{\mathsf{snd}(\mathsf{sdec}(x_2, k))\}^w_k\rangle.P_A)))) \end{aligned}$$

Process P_A is executed after the successful execution of the protocol. We assume that none of the values introduced by the protocol appear in P_A, except for the exchanged session key (represented as free variable k in P_A.) Similarly, the user process B is defined as follows:

$$\begin{aligned} B \doteq \quad & (c_{AB}(y_1) \,.\, \nu s, t, R.\overline{c_{BA}}\langle\{\{R\}^{t,\rightarrow}_{\mathsf{sdec}(y_1,P)}\}^s_P\rangle \,.\, c_{AB}(y_2) \mid \nu v, N_B. \\ & \overline{c_{BA}}\langle\{(\mathsf{sdec}(y_2, R), N_B)\}^v_R\rangle \,.\, c_{AB}(y_3).\ \text{if } (N_B = \mathsf{sdec}(y_3, R)) \text{ then } P_B) \end{aligned}$$

Here, we ask the same restrictions for P_B that we asked for P_A. Similarly to the EPT protocol, we set up a session $\nu P.(A \mid B)$. Now, this protocol reduces to $\nu P.(A \mid B) \rightarrow^5 \equiv\equiv \nu k.(P_A \mid P_B \mid \varphi)$, where $\varphi = \nu P, K, N_A, N_B, R, r, s, t, u, v, w.\varphi_0$, with:

$$\begin{aligned} \varphi_0 \doteq \{ \quad & y_1 = \{pk(K)\}^r_P, x_1 = \{\{R\}^{t,\rightarrow}_{\mathsf{sdec}(y_1,P)}\}^s_P, y_2 = \{N_A\}^u_k, \\ & x_2 = \{(\mathsf{sdec}(y_2, R), N_B)\}^v_R, y_3 = \{\mathsf{snd}(\mathsf{sdec}(x_2, R))\}^w_k \ \} \end{aligned}$$

The five above reductions correspond to the messages exchanges (EKE.i), $i = 1, 2, 3, 4, 5$. The last two equivalences correspond to scope tightenings plus scope extrusion of k. Finally, φ_0 can be shown equivalent to:

$$\begin{aligned} \varphi_0 \equiv \{ \quad & k = R, y_1 = \{pk(K)\}^r_P, x_1 = \{\{R\}^t_{pk(K)}\}^s_P, y_2 = \{N_A\}^u_R, \\ & x_2 = \{(N_A, N_B)\}^v_R, y_3 = \{N_B\}^w_R\} \end{aligned}$$

4.4.2 Security Against Guessing Attacks

Consider the frame $\varphi_P = \nu P, K, N_A, N_B, R, r, s, t, u, v, w.\varphi_0$. Here, φ_{P^-} is the same as φ_P but without P being restricted. Our analysis against dictionary attacks can be carried out by relating φ_P to φ_{P^-} by static equivalence. By lifting the restriction on same names and by adding equational theories (EQ_1 and EQ_2) to the framework, we can analyze a range of different scenarios. We first start by weakening the underlying encryption and consider EKE being instantiated with a repetition revealing cryptosystem (although we still consider only standard abilities EQ_0.)

Repetition Concealing. An interesting fact to note in EKE is that security does not really depend on whether encryption is repetition concealing or not. To see this, consider the frame $\phi_P = \nu P, K, N_A, N_B, R, k.\varphi_0$ and similarly $\phi_{P^-} = \nu K, N_A, N_B, R, k.\varphi_0$. Frames ϕ_P and ϕ_{P^-} are the same frames as φ_P and φ_{P^-} respectively but with the randomness values (r, s, t, u, v and w) being "guessable". This models an encryption scheme that is not repetition concealing, since now the adversary controls the seeds and thus can detect repetitions of same messages. However, the following lemma states that this extra information cannot be exploited by the adversary. If the adversary can distinguish ϕ_P and ϕ_{P^-} where the seeds are known, then she could also distinguish φ_P and φ_{P^-} where the seeds are unguessable.

Lemma 4.4.1. *$\phi_P \approx_s \phi_{P^-}$ iff $\varphi_P \approx_s \varphi_{P^-}$.*

The lemma is shown similarly to Lemma 4.3.1. Intuitively, the lemma holds since letting the adversary have the seeds r, s, t, u, v and w never helps in the task of distinguishing x_i nor y_j, for $i = 1, 2$ and $j = 1, 2, 3$. Having r does not help to recognize y_1, since $pk(K)$ is indistinguishable from random noise when K is private (However this is not true anymore when considering EQ_1, as explained below.) Having s and t does not help to recognize x_1, since now R is random. The remaining cases are similar (for x_2, y_2 and y_3.) Then, we obtain that $\varphi_P \approx_s \phi_P$ and $\varphi_{P^-} \approx_s \phi_{P^-}$, which then allow us to conclude that $\phi_P \approx_s \phi_{P^-}$ iff $\varphi_P \approx_s \varphi_{P^-}$.

While establishing the lemma, we see two cryptographic requirements of EKE:

1. Encryption may be repetition revealing since R is strong. In other words, it must be difficult to compromise R by brute-force attacking y_2, x_2 and y_3 when encryption is repetition revealing, since otherwise x_1 could be distinguished from noise (and thus $\varphi_P \not\approx_s \varphi_{P^-}$.)

2. K must not be used more than once. This can be an important deficiency of EKE, since generation of new keys can sometimes be expensive. In fact, this is where the later protocol OKE [136] improves on EKE. To model a key K used twice, we represent two sessions sharing the same K:
   Let $\varphi_{K,P}$ be $\nu K, P.(\nu R, N_A, N_B.\varphi_0 \mid \nu R', N'_A, N'_B.\varphi'_0)$ where φ'_0 is analogous to φ_0.
   Similarly, let φ_{K,P^-} be $\nu K.(\nu R, N_A, N_B.\varphi_0 \mid \nu R', N'_A, N'_B.\varphi'_0)$ where P is guessable. Now, $\varphi_{K,P} \not\approx_s \varphi_{K,P^-}$ since the environment can distinguish them by comparing $\mathsf{sdec}(y_1, P)$ and $\mathsf{sdec}(y'_1, P)$.

Which-key Concealing. Now we consider the possibility of an adversary to recognize under which key is a message encrypted. To model this, we consider as our equational theory $\text{EQ}_0 \cup \text{EQ}_2$.

We want to see if, under a cryptosystem that is not which-key concealing, $\varphi_P \approx_s \varphi_{P^-}$. By Lemma 4.4.1, it suffices to show that $\phi_P \approx_s \phi_{P^-}$. However, this is not the case, and actually we can see that $\phi_P \not\approx_s \phi_{P^-}$. Let $M = \mathsf{sdec}(x_1, P)$ and $N = \{x\}^{w,\rightarrow}_{\mathsf{sdec}(y_1,P)}$. Then, $(same\_k(M, N) = \mathsf{true})\phi_{P^-}$ but not $(same\_k(M, N) = \mathsf{true})\phi_P$.

Identifying public keys. While showing Lemma 4.4.1, we used the argument that $pk(K)$ is indistinguishable from random noise when K is restricted and thus unguessable. If we consider $EQ_0 \cup EQ_1$ as our equational theory, this does not hold anymore. Intuitively, if an adversary is able to tell whether a public key is valid or not, an adversary could compare many eavesdropped sessions (with many messages (EKE.1)) and narrow the password space considerably, therefore mounting a successful attack over P. The attack is exposed in our setting since when we add EQ_1, we see immediately that $\varphi_P \not\approx_s \varphi_{P^-}$, simply by noticing that decrypting message (EKE.1) returns a valid public key: Let $M = valid\_pk(\mathsf{sdec}(y_1, P))$ and $N = \mathsf{true}$. Then $(M = N)\phi_P$ but not $(M = N)\phi_{P^-}$.

Identifying ciphertexts. EQ_1 allows to distinguish ciphertexts. Then, a possible attack (similar to the one presented above on (EKE.1)) can be mounted on (EKE.2). Here, decrypting (EKE.2) with a good guess returns a valid ciphertext that *pk_ciphertext* recognizes, thus allowing an attack. However, we propose to change (EKE.2) to:

$$B \rightarrow A \quad : \quad \{\{R\}^t_P\}^{s,\rightarrow}_{pk(K)} \qquad \text{(EKE'.2)}$$

Now, message (EKE'.2) does not provide any verification of a guess of P, since the fact that *pk_ciphertext* can recognize (EKE'.2) is now irrelevant if K is secret. Thus, the above attack is prevented. To the best of our knowledge, this modification to EKE has not been proposed before.

Security against dictionary attacks. If we consider only EQ_0, that is, we assume that encryption is which-key concealing and public keys and ciphertexts are not recognizable, then we can state the security of EKE against dictionary attacks. The next theorem shows that the established key session is a *secure* key, i.e. an adversary cannot discover it by only overhearing the messages exchanged during the execution of the protocol. Let φ'_{P^-} be $\nu q, r, s, t, u, v, w.\varphi'_0$, where φ'_0 is $\varphi'_0 \doteq \{y_1 = \{r\}^q_P, x_1 = \{t\}^s_P, y_2 = u, x_2 = v, y_3 = w\}$. Then, we relate $\nu k.(P_A \mid P_B \mid \varphi_P)$, the message exchanges in EKE, with $\nu R.((P_A \mid P_B)\{k = R\} \mid \varphi'_{P^-})$, in which P is made guessable but encrypting random values r and t.

Theorem 4.4.2. *Let P_A and P_B be processes with free variable k where R does not appear. Then, $\nu k.(P_A \mid P_B \mid \varphi_P) \approx \nu R.((P_A \mid P_B)\{k = R\} \mid \varphi'_{P^-})$.*

Proof. (Sketch) Thanks to Lemma 4.4.1, it suffices to show that $\phi \approx_s \phi_{P^-}$, which can be established by case analysis over the messages of φ_P as in Lemma 4.4.1. Then, we can see that $\varphi_{P^-} \approx_s \varphi'_{P^-}$ to conclude that $\varphi_P \approx_s \varphi'_{P^-}$. To obtain the desired observational equivalence we apply Lemma 4.2.4. □

Intuitively, Theorem 4.4.2 says that EKE gives no verification of a guess of P, and furthermore the session key k between P_A and P_B is indistinguishable from random noise.

4.5 Conclusions

In this chapter we revisit the analysis of password protocols against guessing attacks, also known as off-line dictionary attacks, using the Applied Pi Calculus [23]. This calculus follows previous successful proposed process algebraic languages for security (e.g. [25, 138, 105]).

Guessing attacks were already initially explored in Section 2.7, under an attacker based on the Dolev Yao framework and capable of guessing weak keys. Here, we use a process algebraic language that allows the specification of more powerful attacker abilities on top of the standard ones (represented in our approach by the equational theory EQ_0), so we have a more realistic setting. To this end, we have introduced two further equational theories EQ_1 and EQ_2, that model additional adversary abilities. The latter ability modelled by EQ_2, namely which-key revealing, was already considered in [94, 24], where the presence of such an ability would spoil immediately the privacy guarantees of the protocol. In this chapter, we allow EQ_2 to enter the scene explicitly, and study whether its presence allows or not a dictionary attack. We also introduced the equational theory EQ_1, that models the ability of an adversary to distinguish ciphertexts and public keys from random noise. To the best of our knowledge, we do not know of other formal approach for security protocol analysis that considered such an adversary ability. We also considered the ability in which an adversary detects repetition of same messages.

These non-standard abilities turned out to be crucial to decide the security of our case studies, EPT and EKE protocols. Moreover, we believe that our analysis helps to identify

which are the precise cryptographic assumptions that a protocol needs to rely on. For example, as we illustrated with the analysis of EKE, a protocol designer can decide whether to strengthen the underlying encryption or to require stronger key session and nonces, but asking for both may be unnecessary. Furthermore, our technique allows one to spot possible sources of confusions and possible attacks. In particular, we found a new vulnerability for EKE when instantiated with an encryption scheme in which ciphertexts can be distinguished from random noise (i.e. EQ_1). To prevent the attack we propose a simple modification to EKE consisting on changing the order of encryption in Message (EKE.2) to Message (EKE'.2).

An interesting future work is to apply our technique to study the rich field of (later) proposed password protocols (e.g. the protocols proposed in [136, 62, 118].)

Recently, Blanchet proposed an extension to the tool ProVerif [55] (the tool that is used by the TulaFale language in the next chapter) to prove automatically *strong secrecy* for security protocols [56]. Indeed, their (independently developed) notion of strong secrecy, presented in Definition 2 of [56] (and later refined as *weak secrecy* by Blanchet, Abadi and Fournet in [57]), generalizes our Definition 4.2.5. Thus, it would be interesting to validate the (manual) proofs presented in this chapter using Blanchet's technique. This seems to be feasible, as evidenced by the fact that Blanchet's technique has been recently used successfully by Kremer and Ryan [122] to analyse the FOO 92 [97] voting protocol, where security against guessing attacks is analysed using our notion of security (our Definition 4.2.5 appears as Definition 2 in [122]).

Even more recently, Baudet [40] proposes a decision procedure for our Definition 4.2.5, using a constraint solving algorithm with an equational theory (implemented with a specialized convergent rewrite system) for a bounded number of sessions.

As future work, it would be interesting to relate (and further clarify the similarities and differences) between Baudet's work [40], our developments of the present thesis (including also Chapter 2) and Blanchet's et al. work [56, 57].

CHAPTER 5

A Process Algebraic Model for Session-based Web Services

5.1 Introduction

In this chapter we consider the analysis of security protocols deployed as part of web services. These protocols are built on asynchronous communication of XML encoded messages called SOAP envelopes [163]. In this setting, the protocol principals are web clients (typically running as initiators), web services (usually the responders) and several third-party participants like servers that provide additional resources.

Just like regular messages, SOAP envelopes need to be secured on transit. The mechanisms of WS-Security [153] provide means to secure the SOAP envelopes to achieve end-to-end security, using *security tokens*. Examples of security tokens include X.509 certificates, username tokens, and XML-encoded Kerberos tickets.

In itself, WS-Security provides mechanisms for securing a single envelope. However, typically a web service and a client may interact by exchanging series of messages grouped in sessions. In principle WS-Security could secure each separate message of the session, although doing this is inefficient (for example, if X.509 certificates are used in each message). Also, it is often desirable to guarantee integrity of a whole session, and not just each message. For instance, a client querying two services should not be led to attribute a response to the wrong service. On top of WS-Security, the recent specifications WS-Trust [117] and WS-SecureConversation [116] provide mechanisms for communicating parties to establish shared *security contexts* and to use them to secure SOAP-based sessions.

This chapter investigates the security guarantees offered by the WS-Trust and WS-SecureConversation specifications by constructing formal models in the TulaFale scripting language [52]. Briefly, TulaFale is an XML version of a process algebraic language (similar to the Applied Pi Calculus of the previous chapter) which allows the specification of a rich environment model, by providing means to specify processes that describe each protocol principal. Each process may access special (prolog like) predicates, used to filter SOAP envelopes. For example, predicates can be specified for checking that SOAP envelopes have the appropriate expected XML syntax. Moreover, a predicate can perform cryptographic checks on parts of the SOAP envelope, possibly returning data to the calling process. Furthermore,

just like the Applied Pi Calculus of the previous chapter, TulaFale processes can be *replicated* (the bang construct (!) that provides infinite instances of a process) and composed in parallel, thus providing powerful modelling capabilities. In particular, replicating processes allow us to model a possibly unbounded number of participants. Also, replication provides means to encode recursive processes, thus allowing to encode protocol participants which may execute by looping for an (a-priori) unknown number of times. This feature allows us to provide an expressive environment model that encodes participants like clients and services which engage on open-ended message exchanges, thus modelling protocol sessions faithfully.

Our analysis model is presented in Figure 5.1. It is an instantiation of the general model of the Introduction (shown in Figure 1.1), by letting the environment model consist of a TulaFale model. The language also provides means to specify properties (namely secrecy and agreement properties, to cover authentication), conforming the property model. Finally, the attacker model describes an attacker being an unspecified, arbitrary process that runs in parallel with the TulaFale system model (called a *context*). Still, the attacker has the capabilities of the Dolev Yao attacker described in the Introduction, although in addition we also model the leaking of information, thus modelling insider attacks. We elaborate on the analysis model in Section 5.2.

We build our models by studying both the specifications and the WSE implementation [146]. Modelling reveals some potential vulnerabilities as well as allowing us to prove some formal properties.

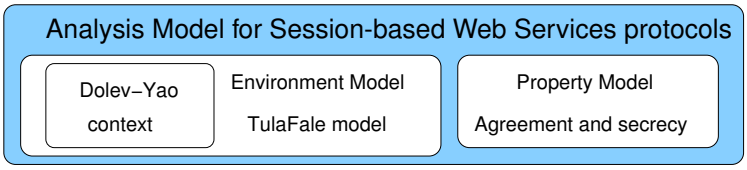


Figure 5.1: Analysis model for Session based Web-service security protocols

Session establishment, WS-Trust and WS-SecureConversation Session establishment is of course not a new issue in cryptography: indeed, numerous classic protocols aim at the mutual authentication of the parties involved in the session, the negotiation of parameters for the session, and the protection of further traffic. (See for example [83, 154, 96, 108].) Moreover, their main secrecy and authentication properties have often been thoroughly studied. Most of their concepts and mechanisms can be usefully applied to SOAP-based protocols, but experience also suggests that this adaptation is not straightforward.

Building on top of WS-Security, WS-Trust [117] describes how security tokens can be requested and issued by SOAP processors; it relies on a dedicated *security token service* (STS) to evaluate requests and issue tokens. Moreover, WS-SecureConversation [116] describes the usage of one such token, named a *security context token*. The token points to a *security context* (SC) typically shared between a client and a web service; its contents can be used to derive keys to protect traffic between these two parties. (There exist two versions of WS-Trust and WS-SecureConversation, released in May 2004 and February 2005. In this chapter, their differences are unimportant, so we focus on the latter version.)

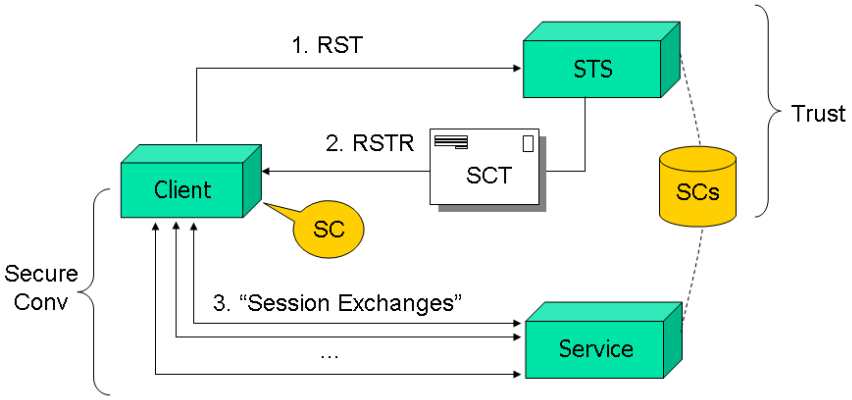


Figure 5.2: A typical protocol relying on WS-Trust and WS-SecureConversation

Figure 5.2 shows a typical usage scenario of WS-Trust and WS-SecureConversation. It roughly corresponds to the sample protocol given in a WSE tutorial [106]; we refer to this tutorial for additional implementation details for this protocol.

Three SOAP processors are displayed: a client, a security token service STS, and a web service. For simplicity, both the STS and the service are co-located and share a session cache (the dashed line in the figure). The STS is configured to establish secure contexts, SCs, with authenticated clients, to be used between clients and services. The first two steps rely on mechanisms covered by WS-Trust, while the latter exchanges (step 3) are specified by WS-SecureConversation.

The execution proceeds as follows. At step 1, the client contacts the STS with a *Request Security Token* (RST) message, including some form of identity token plus information about the target service. The STS, after authorization of the request, generates a new secure context SC, caches it, and replies with a *Request Security Token Response* (RSTR) message that includes a security context token (SCT) to indicate that a new SC has been created (step 2). Crucially, the RSTR contains enough information to allow the client to compute the same SC (with the same shared secret) as the cached version. This allows the client and service to start exchanging messages (step 3) protected using keys derived from the shared secret of the SC.

Chapter Contribution We describe, in Section 5.2, a realistic attacker model for web services, essentially an active attacker with some access to insider secrets. In Sections 5.3 and 5.4 we propose a formal counterpart to web services security specifications for session management, as a collection of predicates and processes reflecting their semantics. Then we develop simple, typical protocols relying on these specifications, and experiment with their implementation using WSE. We state and prove a series of core security properties for these protocols, thereby gaining confidence in our model of these specifications. In Section 5.5 we apply our developments to analyse a concrete protocol used in the Interoperability workshop by several industrial companies to test their implementations.

To the best of our knowledge, this is the first formal analysis of these two web services specifications, and these are the most complex SOAP-based security protocols yet formal-

ized.

The chapter includes only small excerpts from the TulaFale scripts used to establish its main results. The complete scripts are included as examples in the TulaFale distribution, available at `http://Securing.WS`.

5.2 An Analysis Model for Web Service Security Protocols

As mentioned in the Introduction, our formalizations are based on TulaFale [52], an XML version of the Dolev-Yao model embedded within the pi calculus, of which the applied pi calculus of the previous chapter is a variant. Briefly, a computation in the pi calculus consists of a sequence of *reductions* in which a message is passed from a sender to a receiver process. When considering cryptographic protocols in the pi calculus, protocol participants are written as explicit processes, whereas the active attacker is thought of as an arbitrary unknown process running in parallel to the protocol participants. There is a wide range of formal techniques, including automated tools, for analyzing such models of cryptographic protocols expressed in variations of the pi calculus. In particular, our TulaFale tool makes use of the ProVerif analyzer [55].

This section divides into two parts. The first part reviews the TulaFale language. The second part explains the particular threat model considered in this chapter.

5.2.1 Building an Analysis Model in TulaFale

A TulaFale script defines an explicit system of multiple processes running in parallel, representing protocol participants. Processes interact by sending and receiving messages on a fixed set of channels. Messages are terms in an algebraic model of XML, with signature and encryption primitives represented by idealized cryptographic functions [50]. The message formats of typical Dolev-Yao formalisms are rather abstract, and omit many details of the wire representation. In contrast, the TulaFale message format has the detailed structure of XML, and hence is sensitive to rewriting attacks that exploit this structure, such as for instance the compound structure of XML digital signatures. Moreover, we can directly transcribe the message formats of web services specifications into TulaFale, and experimentally check fidelity of the model with respect to messages generated by implementations.

We use logic programming to construct and check messages. For example, the following predicate asserts that the term tok is a username token [153, Section 6.2] for a principal with username u and password pwd, and that k is the symmetric key derived from this password using the nonce n and the timestamp t embedded in the token.

```
predicate isUserToken (tok:item,u,pwd:string,n:bytes,t:string,k:bytes) :-
  tok = <UsernameToken>
          <Username> u </>
          <Nonce> base64(n) </>
          <Created> t </>
        </>,
  k = psha1(pwd,concat(utf8("WS-Security"),concat(n,utf8(t)))).
```

All the predicates shown in the remainder of the chapter are extracted from the code of our formal model [49]. For the sake of brevity, TulaFale omits all XML namespace information, and uses some non-standard abbreviations, such as omitting the tag in a closing element bracket `</>`. Also, literal strings such as `"WS-Security"` are always quoted, whether they are within attributes or elements. Unquoted identifiers, such as u in `<Username>u</>`, are variables. Every variable has a sort: a **string** is an XML string, an **item** is an XML element or string, and a **bytes** is an array of bytes. Function symbols such as base64 and psha1 are abstract representations of operations on the data model; the function psha1 is an idealized hash function with no inverse. Given certain implementability constraints [50], predicates may be used in different modes, depending on whether each parameter is an input or an output. In our example, if tok and k are output parameters, and all the other parameters are inputs, TulaFale computes the two outputs, to yield a username token and its associated key.

We model web services and their clients as explicit processes that send and receive messages on a single 'soap' channel, which models arbitrary transport layers for SOAP messaging. In addition, we annotate processes with events that record the completion of the different phases of protocols. There are two kinds of events, **begin** and **end**, to mark initiation and apparently successful completion, respectively. Events carry data such as participant identities and the contents of messages exchanged.

We model the (unknown) active attacker as an implicit process that runs alongside the explicit processes, and which may interact with them via public channels, such as soap. By default, all channels are public; the attacker process has no direct access to any private channels, which typically model private databases shared between clients and servers.

For a given TulaFale script, a *run* is any series of (potentially nondeterministic) pi calculus reductions and events starting from the explicit system composed with the attacker. The attacker process is arbitrary, except it may not itself generate any events. The observable result of a run is a set of events. Our authentication results are one-to-many correspondences [164] (also known as non-injective agreements [133]) between events. We formulate these as *robust safety* theorems: in every run of the system, every occurrence of an **end** event has a corresponding **begin** event with the same data. For instance, authentication of an RST message is expressed as a correspondence between events marking the client sending and the server receiving the RST. Most security properties of TulaFale models in this chapter are proved automatically by compiling to an intermediate pi calculus, and then running ProVerif.

5.2.2 Principals, Authentication Materials, and Key Leakage

Our models assume the following participants and authentication materials:

- A single certification authority (CA), with public/private keypair kr/sr, that issues X.509 public-key certificates identifying clients and services, signed by the private key sr.
- Multiple principals, each identified by a username u, and equipped with passwords or X.509 certificates issued by the CA.

We assume a single trusted database (coded as messages on a private channel anyPrincipal) that relates usernames to passwords or private keys and certificates. We allow each principal to have multiple passwords and multiple certificates. A certificate for principal u has subject

name u. This database is not accessible to the attacker, but is accessible to client and server processes acting on behalf of users. (In practice, of course, each principal would access instead a local database that contains only a small subset of all passwords and private keys, but this is outside our model.)

We do not fix a particular principal population; instead, we provide public channels to allow the attacker to trigger the generation of fresh authentication materials for arbitrary usernames. Similarly, we do not fix any particular protocol sessions or bound the number of concurrent sessions. We allow the attacker to initiate sessions with arbitrary principals in the roles of clients and servers, and with other parameters chosen by the attacker.

We assume the private key of the CA never leaks to the attacker. However, to model insider attacks, we allow passwords, other private keys, and security contexts to leak to the attacker. In our setting, we say a principal is *unsafe* if any of their passwords or private keys has been leaked to the attacker; otherwise, we say the principal is *safe*. Similarly, we say a WS-Trust security context is *unsafe* if it has been leaked to the attacker, and is *safe* otherwise.

In summary, our system model provides an interface—a set of public channels—to the attacker, giving it the following abilities:

- To send and receive on the soap channel.
- To trigger the generation of a fresh password or a new certificate for any principal.
- To initiate sessions and provide their parameters to clients and servers.
- To cause the leak of passwords or certificates for any principal (but not the certificate authority).
- To cause the leak of established security contexts.

This amounts to a realistic threat model for XML rewriting attacks on web services; it is essential to consider vulnerabilities due to unsafe principals—insider attacks—and indeed we describe such vulnerabilities. (Other threats to web services outside the scope of this model include SQL injection attacks in SOAP payloads and buffer overruns on the networking stack.) Our formal properties concern safe principals, and hold despite the active attacker's ability to craft messages using the authentication materials of unsafe principals. This model of systems and potentially unsafe principals is similar to the TulaFale model in an earlier chapter [51].

5.3 Web Services Trust Language

WS-Trust "provides a framework for requesting and issuing security tokens, and to broker trust relationships" [117]. We survey and discuss its contents, focusing on the parts modelled in this chapter. We refer to the specification for additional information.

5.3.1 WS-Trust as a Protocol Framework

WS-Trust introduces dedicated web services, named *security token services* (STS), that handle requests for security tokens (RSTs) and send responses (RSTRs). Like any SOAP messages, envelopes carrying RSTs and RSTRs may be protected using a selection of mechanisms described in WS-Security.

WS-Trust is deliberately abstract; it provides a general terminology, some precise XML syntax for exchanged data, and an informal description of their usage in context establishment protocols. On the other hand, it avoids defining complete protocols; for instance, it never prescribes any kind of authorization procedure for establishing a security context.

In a common case, a single exchange establishes context: the client sends an RST as the body of an envelope; the STS replies with an RSTR as the (partly encrypted) body of another envelope; and both envelopes include security headers for authentication.

However, other flows of RSTs and RSTRs are possible. In more complex exchanges, any subsequent messages received by the STS are also formatted as RSTRs. In addition, STSs may initiate exchanges by sending unsolicited RSTRs. STSs implement four SOAP actions and corresponding message elements for managing security tokens: for issuance, renewal, cancellation, and validation. Moreover, RST and RSTRs need not appear as envelope bodies; they may also be embedded in the security headers of envelopes carrying other primary payload. WS-Trust allows security token exchanges to be nested. For example, a client may need to contact several STSs to accumulate enough cryptographic evidence before accessing a service; similarly, an STS may contact other STSs in the process of gathering adequate security tokens. Finally, this traffic may itself be protected using tokens previously exchanged.

The goal of these exchanges is to reach an agreement on a *security context* (SC) shared between different parties. The nature of the agreement is left unspecified. For instance, an STS may simply be a public repository for X.509 certificates that accepts anonymous requests and responds with matching certificates, with no particular trust relationship or agreement at the end of the exchange. On the other hand, an STS may establish a protected session between a client and a service, after authenticating the client and enforcing access control to the service, thereby ensuring a precise agreement between the client and the service.

Our formal model (in Section 5.3.4) focuses on the core security aspects of WS-Trust. The model omits some other aspects: renewal, cancellation, and validation actions; error handling; unsolicited RSTRs; and advanced algorithm negotiation and delegation mechanisms. WS-Trust also proposes an optional attribute RequestSecurityToken/@context for correlation between RST and RSTRs. In our model, we rely instead on the message identifier of the enclosing envelope, which plays a similar role.

5.3.2 Syntax for RST/RSTR Exchanges

In what follows, we focus on STSs that implement a simple, two-message RST/RSTR exchange for establishing a security context, as described in the specification [117, Section 6.1-2]. We begin by explaining the detail of the syntax of these messages and their intended semantics, which we reflect in our models.

Principals: An RST may contain a BaseToken element, typically an X.509 certificate or a username token, that identifies the requesting principal and that can be used to authenticate the enclosing envelope. Alternatively, the RST may be anonymous. The RST may also contain an `<AppliesTo>` element indicating the service with whom the client wishes to establish a security context.

Keying: WS-Trust provides optional mechanisms for key establishment: both the client and the STS may include some (encrypted) fresh random valuereferred to as *entropy*; the established context key, if any, is either one of these values, or their joint hash. In

the latter case, for instance, each party decrypts the other party's entropy, then computes sckey = psha1(clientEntropy,stsEntropy) and stores this key as part of the newly-established security context. A benefit of this computation is that the freshness of the key is guaranteed, irrespective of the other party's choice of entropy. (Conversely, if for instance the client accepts an STS-only key, an unsafe STS may supply an arbitrary key, possibly already used in another session.)

Negotiation: RSTs may include additional information, used for instance to demand some choice of cryptographic algorithm or policy, or to provide further authorization materials. We deal abstractly with such additional information, by recording it in the security context.

As a first concrete example of TulaFale code modelling WS-Trust, we give the predicates that verify the structure of RST and RSTR elements in our script. Anticipating WS-SecureConversation, we assume that `"SecurityContextToken"` (SCT) is the type of the requested security token: a basic token with an identity and a key, computed here from client and server entropies.

```
predicate EntropicRST(rst:item,srvURI:string,etok,ExtraInfo:item):−
  rst = <RequestSecurityToken>
          <TokenType>"SecurityContextToken"</>
          <RequestType>"Issue"</>
          <AppliesTo><EndpointReference>srvURI</></>
          <Entropy>etok </>
          ExtraInfo
        </>.
```

The STS decomposes each incoming RST with this predicate; it relies on pattern matching to decompose a (presumed) rst element passed as the first argument into a series of elements. The constant parts in the pattern ensure the RST is a request for SCT issuance; srvURI provides information on the intent of the SCT, here the URI of the web service; etok is the client entropy, encrypted for the service; finally, ExtraInfo collects elements not explicitly used in our model, but perhaps trusted by the protocol participants.

RSTRs returned by the STS include a *requested security token* that indicates the identifier of the (newly created) SCT and a *requested proof token* that contains (typically encrypted) server entropy used to compute the SCT key. It may also include an `<AppliesTo>` (not necessarily matching the RST).

```
predicate EntropicRSTR(rstr:item,srvURI:string,BaseToken:item,
                       uriSTS:string,sctid:string,etok:item):−
  rstr = <RequestSecurityTokenResponse>
           <AppliesTo><EndpointReference>srvURI</></>
           <RequestedSecurityToken>
             <SecurityContextToken>
               <Identifier>sctid</></></>
           <Entropy>etok </>
           <RequestInfo>
             BaseToken
             uriSTS </>
         </>.
```

The client decomposes each incoming RSTR with this predicate; it extracts the srvURI (implicitly comparing it to the request srvURI if this parameter is bound when calling the predicate) and the STS's contributions to the SC (namely its identifier sctid and its encrypted entropy etok).

Our model extends the specification to require that the RSTR include a non-standard element, `<RequestInfo>`, with additional information about the RST, namely the token used to sign the RST and the URI it was sent to. Without this extension, it is not possible to securely correlate the RSTR with its RST.

5.3.3 Towards an Explicit Agreement on the Exchange

Session establishment is a well-studied goal for cryptographic protocols. In contrast to specifications of fixed protocol, however, WS-Trust leaves open several important design decisions that should be carefully considered when assembling a protocol.

Crucially, RST/RSTR exchanges aim to establish shared security contexts, but the contents of these contexts (including the participants' intentions) is left implicit. This can be a source of confusion, inasmuch as the flexibility of web services enables many different levels of agreement between processors sharing a context. Ideally, the specifications should help secure precise agreements on security contexts between clients, STSs, and servers. Following well-established prudent practices, a simple way to achieve strong agreement would be to supplement the syntax of RSTs and RSTRs with (optional, well-defined) data on the exchange, such as selected modes for authentication and keying, and identities of the requester, issuer, and target service. It is also important that this syntax be specified, so that its presence and contents can be validated.

For a given system, one should explicitly state what is guaranteed, both when an STS accepts an RST and issues an RSTR, and then when a client accepts an RSTR. These guarantees depend both on the contents and processing of the RST and RSTR. Hence, one should carefully review:

1. what needs to be agreed upon—typically not just the SCT key;

2. what is passed in the RST/RSTR (notably the signed materials in these messages);

3. whether the web service implementations actually provide an agreement based on their processing of the exchange.

In a given implementation, an effective agreement depends on details of envelope processing. Still, the safety of security contexts should not overly rely on implementation choices. At least, whenever an exchange succeeds, the protocol designer may expect that any piece of data recorded in the security context is authentic. In comparison, traditional session establishment protocols like SSL [96] or IKE [108] have specific options and guarantees to reach precise agreements, typically covering at least any data exchanged by the protocol.

Following the scenario illustrated in Figure 5.2, we define a concrete agreement. The agreement should at least cover the actual contents of SCs observed in the WSE implementation: a shared SCT identifier, a key, and some identity information for the three involved principals. It should also cover security parameters used in the exchange, such as:

- Whether the RST client is authenticated or anonymous. For instance, a client may be convinced it is authenticated because its signature was accepted, whereas the STS received an unsigned request with the same message identifier and assumed an anonymous token was requested.
- Whether both the client and server, or the server alone, provided entropy. A mismatch may lead to an apparently successful SC recording a bad key.
- Any data used to authorize the issuance of the RSTR, such as the security token providing client authentication, or credentials presented in the RST.
- The URI and action for the STS. This may matter if different STSs enforce distinct authorization policies for the same service.

We arrive at the following content for the security context in our model, expressed as a TulaFale predicate:

```
predicate sctSC(SC:item,sctid:string,sckey:bytes,mode:string,
                UserToken,StsInfo:item,appTo:string,extra:item) :−
  SC = <SecurityContext>
         <Identifier>sctid</>
         <Key>base64(sckey)</>
         <Base>UserToken</>
         <STSInfo>StsInfo</>
         <AppliesTo>appTo</>
         <EntropyMode>mode</>
         <ExtraInfo>extra</></>.
```

Crucially, the security context records information on the identity of the three principals involved: `<Base>` records the token for the client; `<STSInfo>` records the URI and token for the STS; `<AppliesTo>` records the URI for the service.

5.3.4 Modelling and Verifying Uses of WS-Trust

We present our model of a single RST/RSTR exchange, such as the first exchange of the protocol depicted in Figure 5.2. The goal of the exchange is to ensure agreement on a shared security context. We first describe the exchange, detailing our implementation of principals and the processing of RST and RSTR envelopes, then we state our main theorems for the resulting script, verified using the TulaFale and ProVerif tools. For simplicity, we only provide a few exemplary excerpts of the script—we refer the reader to `ssws-trust.tf` for complete definitions.

Mapping Principals to TulaFale Processes

A basic RST/RSTR exchange consists of two messages exchanged between a client and an STS process, as depicted in Figure 5.3. The goal of the exchange is for the two processes to establish and agree on an SC with a fresh identifier sctid and shared key sckey. This agreement is achieved in two steps. After the RST message has been accepted, the client and STS agree on a *partial* SC that consists of all the elements of the SC except `<Identifier>`,

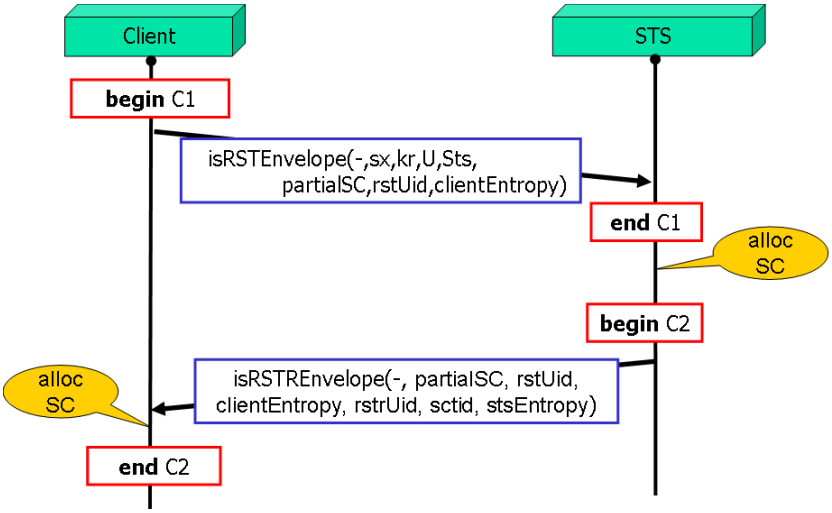


Figure 5.3: Establishing a secure context in entropic mode

<Key>, and the STS token (within <STSInfo>), which are undetermined at this stage. After the RSTR message is accepted at the client, both processes agree on the full established SC. In the rest of this section, we describe how the two processes construct and check messages to achieve this agreement.

To begin with, both processes know kr, the public key of the CA used to check the validity of public key certificates, and share the trusted database of principal secrets.

In our exchange, each envelope embeds a globally unique identifier; its structure is represented by the predicate uid; it consists of a freshly generated message identifier, id, and a public timestamp, t. This identifier is typically used to protect against message replays.

The process Client represents an instance of a SOAP client sending RSTs and processing RSTRs on behalf of a user. Each run is in one of two operation modes: either *entropic mode*, where the client provides entropy for the security context, or *non-entropic* mode, where it does not. In both modes, the server provides entropy. (We do not model a third mode, allowed by the specification, where the client alone provides entropy.) Figure 5.3 depicts a typical run in entropic mode. In both modes, the attacker initializes the client process by sending it a PartialSC that provides the parameters for a new security context, including the name of the user and the URIs for the STS and service. The client then retrieves the user record U from the trusted database (see Section 5.2) and constructs an RST message corresponding to PartialSC. A user record contains either a username and password or an X.509 certificate and its associated private signing key. The RST message has a unique identifier, rstUid, and in entropic mode, it also contains a fresh client-generated value, clientEntropy.

The process STS represents an STS server. It first retrieves a server principal record Sts, containing a URL address uriSTS, an X.509 public-key certificate certSTS, and the associated private signing key skSTS. When it receives an RST, it also retrieves the principal record U for the client. The trusted database thus represents all authorized clients of the STS. After checking the RST, the STS process generates a new security context with fresh

identifier sctid and the received parameters PartialSC. It generates its own stsEntropy and uses it to compute the shared sckey associated with this context. It then returns an RSTR, uniquely identified by rstrUid, containing the sctid and stsEntropy to the client.

The server entropy in the RSTR, and, in entropic mode, the client entropy in the RST, are encrypted and then signed, as in the WSE implementation [146].

The intentions of and the agreement between the client and STS are recorded using **begin** and **end** events, as follows. Before sending the RST, the client records its proposed security context as the event **begin** C1. After checking the RST, the STS indicates its acceptance of the proposed context as the event **end** C1. Similarly, before sending the RSTR, the STS records the established security context as **begin** C2, and after checking the received RSTR, the client indicates acceptance of the context with **end** C2.

The correspondence assertion C1 after the first message requires that the client and STS processes agree on the values of the proposed parameters PartialSC, the rstUid, and the clientEntropy:

$$\text{C1} = (\text{PartialSC},\text{rstUid},\text{clientEntropy})$$

Including rstUid in C1 enables replay detection: if the STS process were to further ensure that it never accepts two RSTs with the same Uid, then agreement on C1 implies that each message sent by the client is accepted at most once by the STS process.

The correspondence assertion C2 after the second message requires that the client and STS processes agree on the full established SC and the unique identifiers of both messages (again to enable replay detection).

$$\text{C2} = (\text{SC},\text{rstUid},\text{rstrUid})$$

We say that a principal is a client, STS, or server in C1 (or C2) if it is recorded under the corresponding role in the security context PartialSC (or SC). For instance, we say that C1 has a *safe client* if the principal recorded in the Base field of PartialSC is safe.

Processing the RST and RSTR Envelopes

In our exchange, the SOAP envelope that carries the RST has a header consisting of a message identifier, `<To>` and `<Action>` elements designating an STS for issuing an SCT, and a `<Security>` element that itself consists of a timestamp, a token identifying the client, and a digital signature. This structure is expressed as a predicate:

```
predicate envRST(env,rst:item,uriSTS,id,t:string,Sig,BaseToken:item) :−
  env = <Envelope>
          <Header>
            <MessageId>id</>
            <To>uriSTS</>
            <Action>"RSTSCT"</>
            <Security>
              <Timestamp><Created>t</></>
              BaseToken
              Sig</></>
          <Body>rst</></>.
```

The client uses this predicate (and others) to assemble an RST envelope; conversely, the STS uses this predicate to check that a received envelope complies with this structure, as a first step of its processing. The full processing for the RST envelope at the receiving STS is coded by the predicate:

```
predicate isRSTEnvelope(msgrst:item,kr:bytes,U,Sts:item,
                        PartialSC,rstUid:item,clientEntropy:bytes) :-
  envRST(msgrst,rst,uriSTS,id1,t1,sig1,BaseToken),
  EntropicRST(rst,srvURI,etok,ExtraInfo),
  isSTS(Sts,StsInfo,uriSTS,subjSTS,sx,certSTS),
  isAsymEncryptedKey(etok,clientEntropy,sx),
  isX509Token(BaseToken,kr,u,a,ek),
  isSignature(sig1,"rsasha1",ek,
        [<Body>rst</><To>uriSTS</><Action>"RSTSCT"</>
         <MessageId>id1</><Created>t1</>]),
  uid(rstUid,id1,t1),
  PartialSctSC(PartialSC,"Both",BaseToken,StsInfo,srvURI,ExtraInfo).
```

Here, we depict the clause used to check an entropic RST signed using a user's X.509 public-key certificate. The script contains similar clauses for checking the other cases, and it defines a symmetric predicate for preparing RST envelopes on the client side.

This predicate takes as input the received envelope (msgrst) and checks it using the public key of the CA (kr) and the principal records for the user (U) and for the STS (STS). It then extracts as output the proposed context PartialSC, the unique identifier rstUid, and the received clientEntropy.

The predicate first parses msgrst calling envRST, extracting the rst, the relevant header fields, the message signature sig1, and the user's authenticating BaseToken. It then calls isEntropicRST to parse the rst and retrieve etok, which contains the encrypted clientEntropy, checking that it contains a fragment URI BaseTokenId pointing to the user's BaseToken. The predicate isSTS checks that the STS record Sts has a uriSTS that matches the `<To>` header of the RST, and extracts the certificate certSTS and private key sx corresponding to the STS. This private key is used to decrypt etok to retrieve the clientEntropy. Then, the predicate isX509TokenPub extracts the user's public key from BaseToken and the predicate isSignature checks that the corresponding private signing key has been used to generate the message signature sig1, and that sig1 covers the message body and all the parsed header elements. Finally, the predicates uid and PartialSctSC construct the outputs rstUid and PartialSC embedded in C1.

The SOAP envelope carrying RSTRs has a similar structure to the RST envelope, expressed in the following predicate:

```
predicate envRSTR(msgrstr:item,rstr:item,id2:string,t2:string,
                  STSToken:item,sig2:item,rto:string) :-
  msgrstr = <Envelope>
              <Header>
                <MessageId>id2</><RelatesTo>rto</>
                <Action>"RSTRSCT"</>
                <Security>
                  <Timestamp><Created>t2</></>
```

STSToken
sig2</></>
<Body>rstr</></>.

The main difference is that the <To> header is replaced with a <RelatesTo> header that contains the message identifier of the RST being responded to.

Both the creation of RSTRs at the service and the corresponding checks at the client are again expressed as symmetric predicates.

Authentication and Secrecy Results

As described in Section 5.2, our full TulaFale script models our system as an explicit pi calculus process consisting of an unbounded number of clients and STSs running in parallel and willing to communicate over a public channel, under the control of the attacker, for any choice of operation modes.

Although the opponent is powerful, our theorems assert that the RST/RSTR exchange preserves our authentication, correlation, and secrecy goals. The authentication and correlation goals are stated in terms of the correspondence between **begin** and **end** events generated by client and STS processes; the attacker cannot generate events.

Theorem 5.3.1 (Robust Safety of C1, C2)**.** *For all runs of script* `ssws-trust.tf` *in the presence of an active attacker, we have:*

- *For each* ***end*** *C1 event with a safe client, there is a matching* ***begin*** *C1 event.*
- *For each* ***end*** *C2 event with a safe client and a safe STS, there is a matching* ***begin*** *C2 event.*

Hence, the exchange guarantees that any RST envelope from a safe client, accepted by a safe STS, and used to allocate a secure context actually corresponds to a genuine request with matching parameters.

Secrecy is stated in terms of the attacker's knowledge of the established session key.

Theorem 5.3.2 (Session-Key Secrecy)**.** *For all runs of script* `ssws-trust.tf` *in the presence of an active attacker, for each* ***begin*** *C2 with safe client and STS, the Key element recorded in SC remains secret.*

Hence, even if the service immediately uses the SC key to encrypt messages, at most the client who signed the RST may decrypt those messages. Combining the two theorems, this also holds once the client issues a matching **end** C2 and starts using the SC key. In addition, we have checked that the result holds even if, in entropic mode (that is, where both client and server provide entropy), one of the participants uses a value selected by the attacker instead of a fresh value as its entropy.

As a corollary, if both the client and the STS are safe and the client completes the exchange, then the two parties agree on an SC containing a shared, secret key.

These results are automatically proved by running TulaFale on script `ssws-trust.tf`. In addition to security properties, we also check a series of basic functional properties, checking for instance that the protocol can successfully complete for each choice of mode and safe principals.

5.4 Web Services Secure Conversation Language

WS-SecureConversation "defines mechanisms for establishing and sharing security contexts, and deriving keys from established security contexts (or any shared secret)" in order to secure series of messages [116]. We survey the specification, and in particular focus on the new security tokens it introduces.

5.4.1 Tokens for Contexts and Key Derivations

WS-SecureConversation introduces two new kinds of security token: SCTs and DKTs.

A *security context token* (SCT) in the security header of an envelope represents (an abstract pointer to) a shared security context (SC), typically established using an RST/RSTR exchange, as described in previous sections. The SCT simply embeds a *context identifier*, so that the recipient can access the relevant context, notably the authenticated identity of the sender. Local references to the SCT can appear in the envelope whenever a symmetric key is needed, to indicate that the recipient should read the key from the SC. In our scripts, we use the following structural predicate for SCTs:

```
predicate SCT(sct:item,sctid:string):−
  sct = <SecurityContextToken><Identifier>sctid</></>.
```

A *derived key token* (DKT) provides a reference to a master key, an algorithm, and additional parameters to compute a separate key. For instance, a typical DKT embeds a fresh nonce and a reference to an SCT, and thereby indicates that the recipient should compute a derived key as the hash of that nonce keyed with the SC key. Such DKTs may be used to secure independent requests relying on the same SC, or to derive distinct keys for encryption and for authentication.

In our scripts, we use the structural predicate DKSCT to decompose DKTs that refer to SCTs and the predicate deriveKey to compute the associated key:

```
predicate DKSCT(dksct:item,sctid:string,nonce:bytes):−
  dksct = <DerivedKeyToken>
            <SecurityTokenReference>
              <Reference>sctid</>
              <valueType>"SCT"</></>
            <Nonce>base64(nonce)</></>.

predicate deriveKey(dk:bytes,key:bytes,nonce:bytes):−
  dk = psha1(base64(key),concat(utf8("WSSecureConversation"),nonce)).
```

In general, the parent token need not be an SCT; instead one can use, for example, a Kerberos token, or even another DKT. WS-SecureConversation also supports other variants of key derivation, a lightweight derived-key mechanism that provides the same functionality as a DKT within a key reference, and some SCT propagation and amendment mechanisms. We do not model these advanced mechanisms in this chapter.

5.4.2 Modelling and Verifying Uses of WS-SecureConversation

Continuing with the example of Figure 5.2, we now consider exchanges between a client and a service following the completion of an RST/RSTR exchange, as modelled in Section 5.3.4.

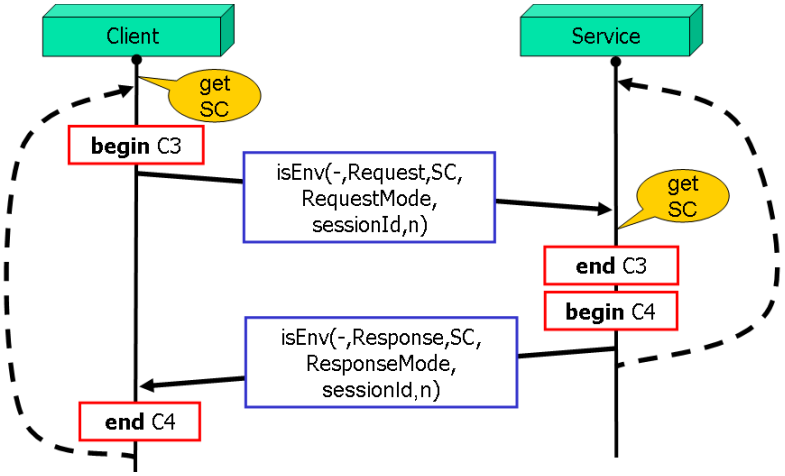


Figure 5.4: Iterating exchanges

The security goals of these exchanges are to achieve mutual authentication between client and service, to ensure message correlation between requests and responses, and to preserve the secrecy of all message bodies. We first model a single request/response exchange, before generalizing our results to "open-ended conversations" comprising arbitrary sequences of exchanges.

A typical run of the protocol is depicted in Figure 5.4 (but disregard the dashed lines for now). It involves a process Client that sends a request to a web service using an existing security context and waits for a response, and a process Service that handles such requests, for some given address and SOAP action (srvURI,srvAC).

Mapping Principals to TulaFale Processes

When considering each envelope in this protocol, we use an abstract parameter, DestInfo, to represent the concatenation of some WS-Addressing [60] headers included in the envelope.

First, the client inputs from the attacker a Request envelope that provides a security context identifier sctid, a timestamp t, and target service information srvURI and srvAC. The client then fetches from the SC database a security context that matches sctid, if any, and extends the Request envelope by adding a fresh message identifier and a secret request body. (Hence, Request is an envelope with some DestInfo that includes headers containing the request message identifier and target service information.)

In addition, the attacker can also choose between two operation modes: either securing the request with the shared SC key, or securing it with fresh keys derived from the SC key. These key-derivation details are recorded in an element, RequestMode, which contains either two nonces used to derive keys for encryption and signature, or a constant indicating that the SC key is directly used.

As in Section 5.3.4, our processes issue events that record their intent: before sending the request, the client emits **begin** C3; after receiving the request and checking its validity, the web service records the acceptance by emitting **end** C3. These events record the following

data:

C3 = (SC,Request,RequestMode)

After accepting a request from a client, the service similarly prepares a Response containing a response body and some addressing headers: DestInfo now includes headers echoing the server address srvURI and the request identifier, plus a header containing a fresh response identifier. For simplicity, the service uses the same operation mode as the client: if the request used derived keys, so does the response. The corresponding key derivation details are recorded in ResponseMode.

Before sending its response, the service emits **begin** C4. After checking the validity of the response, the client emits **end** C4. These events record data for both the request and the response:

C4 = (C3,Response,ResponseMode)

where C3 includes data on the request, and Response and ResponseMode include data on the response.

Next, we describe the structure and processing of envelopes that effectively protect these requests and responses.

Processing Request and Response Envelopes

Since the request and response envelopes are processed similarly, we use generic predicates for both purposes. When using derived keys, the structure of these SOAP envelopes is given by the predicate:

```
predicate isEnv(Env:item,DestInfo:items,t:string,sig,ebody:item,
                sctid:string,mode:item) :−
  Env = <Envelope>
          <Header>
            <Security>
              <Timestamp><Created>t</></>
              sct dksctEnc dksctSig
              sig</> @
            DestInfo </>
          <Body>ebody</></>,
  SCT(sct,sctid),
  DKSCT(dksctEnc,sctid,EncNonce), DKSCT(dksctSig,sctid,SigNonce),
  derivedKeyMode(mode,EncNonce,SigNonce).
```

The structure of the envelope differs from those of Section 5.3.4 in three ways:

- the envelope includes a security context token (sct) and two derived key tokens (dksctEnc and dksctSig) used to indicate keys for encryption and signing;
- the envelope includes a generic parameter (DestInfo) that provides headers specific to requests and responses;
- the envelope includes an encrypted body (ebody).

After setting the structure of the envelope, the isEnv predicate inspects the SCT and DKTs, in order to return an SC identifier (sctid) and a mode descriptor embedding the two nonces used for key derivation (EncNonce and SigNonce).

We now give a predicate used for validating incoming envelopes: requests for the service, and responses for the client.

```
predicate isMessage(Env,SC,EnvelopeInfo,mode:item) :−
  isEnv(Env,DestInfo,t,sig,ebody,sctid,mode),
  sctSC(SC,sctid,sckey,entropyMode,UserToken,StsInfo,appTo,extra),
  computeKeys(mode,sckey,EncKey,SigKey),
  isEncryptedData(ebody,body,EncKey),
  body = <Body>b</>,
  EnvInfo(EnvelopeInfo,t,sctid,DestInfo,b),
  isSignature(sig,"hmacsha1",SigKey,
              [<Body>ebody</> <Created>t</> @ DestInfo ]).
```

In the predicate, isEnv parses the envelope, extracting DestInfo and other sub-elements.

Next, predicate sctSC checks sctid against the security context SC and then retrieves the security context key sckey. Predicate computeKeys uses that key and the two nonces passed in mode to compute the keys EncKey and SigKey protecting the envelope, as explained in Section 5.4.1. EncKey is then used to decrypt the message body, whereas SignKey is used to verify a signature binding the encrypted message body, a timestamp, and the addressing headers. Finally, information extracted from the envelope is returned in EnvelopeInfo.

Authentication and Secrecy Results

The following theorem establishes the agreement, message correlation, and secrecy properties for the exchange described above. Its proof is obtained by running TulaFale.

Theorem 5.4.1 (Robust Safety of C3, C4 and Secrecy)**.** *For all runs of script* `ssws-secconv.tf` *in the presence of an active attacker, we have:*

- *For each* ***end*** *C3 with a safe security context, there is a matching* ***begin*** *C3.*
- *For each* ***end*** *C4 with a safe security context, there is a matching* ***begin*** *C4.*
- *For each exchange with a safe security context, the request and response bodies are kept secret.*

Open-Ended Conversations

We now extend our protocol to allow clients and services to iterate their exchanges—as suggested by the dashed lines of Figure 5.4—thus modelling a more substantial conversation. For simplicity, we fix the operation mode and always use derived keys.

Each session is identified by a sessionId string, freshly generated by the client before sending its first request. Within the session, each request is indexed by a sequence number. To this end, we (mostly) comply with the syntax of WS-ReliableMessaging [91] and use its simple request acknowledgment mechanism: requests carry a <Sequence> header, including the sessionId and a message number n, set to zero by the client in the first request, and

incremented by one in every subsequent request. The structure of this header is given by the predicate:

predicate sequence(Sequence:**item**,sessionId:**string**,msgNumber:**string**):−
  Sequence = <Sequence>
  <Identifier>sessionId</>
  <MessageNumber>msgNumber</></>.

Similarly, responses carry a <SequenceAcknowledgement> header echoing the received sessionId and message number.

To specify an agreement on the conversation as a whole, our client and service collect detailed information in events, as follows. For the n-th request and response, respectively, C3n and C4n record as history H not only the envelope just sent (Req_n for **begin** events) or accepted (Resp_n for **end** events), but also the preceding sequence (denoted using parenthesis and spaces between elements) S=[Resp_{n-1} Req_{n-1} $\cdots$ Resp_0 Req_0] of all previously-processed envelopes for the session. Thus, for C3n, the recorded history is H=[Req_n @ S] (here @ denotes concatenation) while for C4n the recorded history is H'=[Resp_n Req_n @ S]. C3n and C4n also record the shared session identifier sessionId, the message number n of the last exchange, and the security context SC (which provides in particular client and service identification).

C3n = (SC,sessionId,H,n)
C4n = (SC,sessionId,H',n)

To establish the correspondences, we use a script that protects the service from replays of initial requests with identical session identifiers. (This is necessary because the server does not contribute to the generation of the session identifier, and thus could be lead to run several sessions for a single client session.)

Theorem 5.4.2 (Robust Safety of C3n, C4n and Secrecy)**.** *For all runs of* `ssws-secrm.tf` *in the presence of an active attacker, we have:*

- *For each **end** C3n with a safe security context, there is a matching **begin** C3n.*
- *For each **end** C4n with a safe security context, there is a matching **begin** C4n.*
- *All request and response bodies protected by a safe security context remain secret.*

The proof is shown in Appendix A.2; it uses the script `ssws-secrm.tf` for the iterated protocol. It also relies on a similar, but slightly more abstract script, `ssws-secrm-a.tf`, in which sequencing is also controlled by the environment. We use ProVerif on both scripts to establish a series of correspondences. We then manually combine these properties by reasoning on the structure of these scripts, relying on standard proof techniques for the pi calculus [23, 50].

5.5 Application: Interoperability Scenarios

We finally consider the working scenario described in the WS-Trust/WS-SecureConversation interoperability ('interop') workshop [166]. In contrast with specifications, this scenario provides a concrete protocol, used as a common test case for comparing implementations from

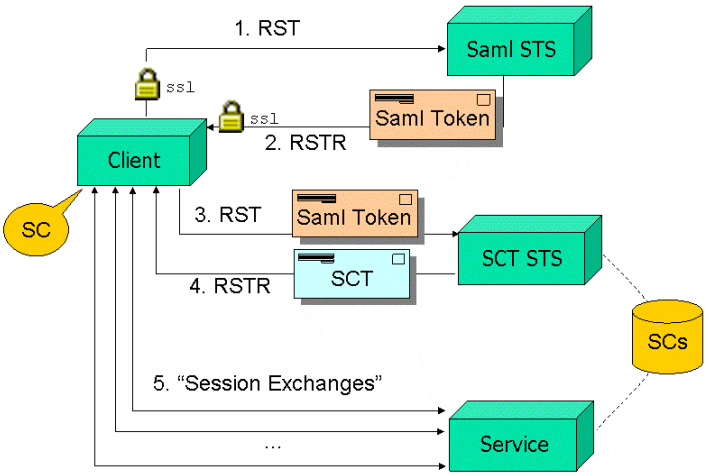


Figure 5.5: The WS-Trust/WS-SecureConversation Interop Scenario

six different vendors (IBM, Lanka Software Foundation (Apache), Microsoft Corporation, Oblix, Inc, Systinet Corporation, and Virtusa Corporation (Apache)) [1].

The scenario is outlined in Figure 5.5; it involves four roles—a client C, a SAML server, an STS server, and a web service S—and a series of exchanges initiated by the client:

- An RST/RSTR exchange with the SAML server, to obtain a token. This exchange is protected at the transport layer, using SSL.
- A second RST/RSTR exchange with the STS server. This exchange is protected using WS-Security, relying on the SAML token. The outcome of the exchange is a security context shared between the client and the target service.
- One or several exchanges with the service, protected using WS-Security and WS-SecureConversation relying on the shared SC.

As a formal counterpart, we apply the TulaFale models developed in the previous sections to this scenario, in order to verify a series of authentication properties. We reuse the processes and predicates of Section 5.3 to represent the two RST/RSTR exchanges, and those of Section 5.4 to represent the final exchange.

We spent a couple of weeks to model and verify this scenario (including extensions of our library to support SSL and SAML as required in the scenario). We believe this verification effort is comparable to the coding and testing of the scenario for an existing target

[1] This information has been gathered from the website `http://msdn.microsoft.com/webservices/community/workshops/TrustWorkshopOct2004.aspx`.

implementation of web services. Besides, our specification of target security properties for the scenario should be of interest for testing. We found several flaws in earlier versions of the scenario, although not in its latest version.

5.5.1 Simple Models for SSL and SAML

We model SSL and SAML tokens only as required in the scenario.

The SAML token enables client authentication, and also sets up a shared secret between the client and the STS server. Suppose that our client C has subject name u, and our SAML server has an asymmetric private key sT to sign tokens targeted to the STS server with asymmetric public key ka. Then, a SAML token s with distributed secret k is built by the following predicate. (The secret key k is computed with client entropy and server entropy, much as in Section 5.3.4.)

```
predicate Saml(s:item,sT:bytes,u:string,ka,k:bytes) :−
  mkAsymEncryptedKey(ekey,k,ka),
  auth = <AuthenticationStatement>
           <Subject><NameIdentifier>u</></>
           <SubjectConfirmation><KeyInfo>ekey</></></>,
  mkSignature(sig,"rsasha1",sT,[<SamlAssertion>auth</>]),
  s = <SamlAssertion>
        auth sig
      </>.
```

First, in predicate mkAsymEncryptedKey the secret key k is encrypted with the public key of the target STS server ka, to obtain an encrypted key ekey. Then an auth item is created containing the encrypted key ekey and the subject user name u. Then, the auth item is signed with the private key of the SAML server, sT. Finally, the SAML token s consists of the auth item and the corresponding signature.

Section 5.3.4 presents an RST/RSTR exchange where the token provided is a security context token SCT. In this stage of the interop scenario, the RST/RSTR exchange is similar, only that instead of exchanging an SCT we exchange a SAML token. Also, the RST/RSTR exchange is protected using SSL, modelled as follows.

SSL [96] protects SOAP envelopes as a whole, irrespective of their contents (including any embedded SOAP-level principal identifier or password). Hence, in the scenario, SSL connects an anonymous client to a server associated with a public-key certificate. We model this connection simply by letting the client C generate a symmetric key symk, pass it encrypted under the SAML-server public key, and use that key for encrypting the first RST and RSTR envelopes. For instance, our SAML server retrieves obtains the first RST envelope env from the encrypted message msg using predicate:

```
predicate isSSLRequest(msg,ek,symk:bytes,env,S,serverToken:item) :−
  isX509(S,_,cert,sT,kr),
  x509Token(serverToken,cert),
  c14n(<SSLKey>base64(symk)</>) = decrsa(sT,ek),
  c14n(<SSLRequest>env</>) = decaes(symk,msg).
```

The SAML server has an X509 certificate cert with a corresponding private key sT. Using sT the server can decrypt the encrypted key ekey and obtain symk, with which the envelope env is encrypted.

5.5.2 Authentication Results

As in the previous sections, any number of principals may be involved in parallel runs of the scenario, and they systematically record their view of the run using correspondences. For a given run of the protocol, the client records **begin** I1 before sending the first RST; the SAML server records **end** I1 after receiving and accepting a first RST, then records **begin** I2 before sending back the first RSTR; ...; finally, the client records **end** I6 after receiving and accepting the service response. The contents of the six correspondence sets our authenticity expectations for this scenario, as follows:

I1 = (partialSamlSC)
I2 = (SamlSC)
I3 = (partialSctSC)
I4 = (SctSC)
I5 = (SctSC,Request)
I6 = (SctSC,Request,Response)

In these correspondences, the various SC elements record agreements at each stage of the protocol. Their content for the interop scenario extend the one described in Section 5.3.3; it includes for instance a tag that indicates whether SSL or WS-Security is used to secure the agreement, and additional details on the keys and the issued tokens.

In particular, the name of the client principal is recorded as part of the Base token in all these SCs: first in a username token in partialSamlSC and SamlSC; then in the SAML token associated with SamlSC in partialSctSC and SctSC. The URI of STS server is also recorded: first as the target AppliesTo field in partialSamlSC and SamlSC, then in STSInfo in partialSctSC and SctSC. Finally, Request and Response record authenticated envelope contents and their correlation, as described in Section 5.4.2.

For this section only, we adapt our notion of client safety as follows: a client principal is unsafe when (1) any of its secrets has been leaked to the attacker, or (2) it has contacted any unsafe SAML server. Hence, client safety now also depends on SAML server safety. (Using TulaFale, we easily check that our properties do not hold if, instead, we request only that the SAML server contacted for the session be safe, as the client may have sent its password to another, unsafe SAML server during a prior session. That is a well-known shortcoming of SSL as used in this scenario.)

The complete script is `wstrust-secconv-interop.tf`. By running TulaFale, we establish the following properties:

Theorem 5.5.1 (Robust Safety of I1–I6)**.** *For all runs of the script in the presence of an active attacker:*

- *For each **end** I1 event with a safe client, there is a matching **begin** I1 event.*
- *For each **end** I2 event with a safe client, there is a matching **begin** I2 event.*

- *For each **end** I3 event with a safe client, there is a matching **begin** I3 event.*
- *For each **end** I4 event with safe client and STS server, there is a matching **begin** I4 event.*
- *For each **end** I5 event with safe client and STS server, there is a matching **begin** I5 event.*
- *For each **end** I6 event with safe client, STS server, and service, there is a matching **begin** I6 event.*

5.6 Conclusions

The mechanisms of WS-Security allow the creation of complex web service security protocols, whose detailed analysis can become more difficult than standard protocols (like the ones studied in the previous chapter). Moreover, on top of the standard WS-Security specification, the WS-Trust and WS-SecureConversation specifications add complexity, as now the web service protocols may consist of many exchanges between several different parties. For example, in the Interop protocol of Section 5.5 the client first establishes a SAML token with a SAML STS, then negotiates a security context with an SCT STS and finally the client performs a conversation with the web service.

The language TulaFale allows for an elaborate description of these protocols, due to its powerful constructs and XML support. As already mentioned, TulaFale compiles to an intermediate pi calculus language that can be automatically analysed by the ProVerif analyzer [55]. Differently to the constraint solving procedure presented in Chapter 2, ProVerif is a semi-decision procedure that may not always converge. This is expected since TulaFale allows the specification of environments consisting of an unbounded number of participants, which turns the security problem undecidable as discussed in the Introduction. (The procedures of Chapter 2 always terminate and give an answer since the considered system scenarios are finite). Still, all the example protocols considered in this chapter converge in ProVerif, except for the case of open-ended sessions. However, even in this case we still benefit from the usage of the automatic analyzer, as follows. From the concrete (open-ended) protocol, we build a more abstract protocol which converges in ProVerif. We then use the abstract protocol to establish some properties, and then manually relate the abstract protocol to the concrete protocol to establish the desired properties (for details, see Appendix A.2). This proof strategy, i.e. of mixing manual and automatic proofs, is useful and scales well, something indispensable since as protocols get more complex many parts can be left to the analyzer to prove automatically, while other parts can be established by hand (this strategy has already been used successfully by Abadi, Blanchet and Fournet for the JFK protocol [22] and by Abadi and Blanchet for a protocol for certified mail [21]).

Our study of the WS-Trust and WS-SecureConversation specifications complements the ongoing work to author and refine the WS-Trust and WS-SecureConversation specifications, to develop implementations, and to test conformance at interoperability workshops. Our positive results concerning secrecy and authenticity (i.e. Theorems 5.3.1, 5.4.1, 5.4.2 and 5.5.1) within a formal threat model increase confidence in particular uses of the specifications.

Related Work There are by now many implementations of SOAP and XML security, but there is comparatively little work on formalizing the resulting security properties. Damiani *et al* [76] show access control properties for SOAP-based web services, relying on an underlying secure channel such as SSL/TLS. Gordon and Pucella [103] prove authentication properties of SOAP-based security protocols, but do not consider key establishment and do not model XML syntax in detail. Kleiner and Roscoe [119] construct abstract descriptions of some simple WS-Security protocols from XML message sequences, so as to analyze the abstractions with the FDR model checker.

Other formal work on web services protocols includes a model of Atomic Transaction [114].

CHAPTER 6

Relating Analysis Models

6.1 Introduction

As mentioned in the Introduction, analysis models for security protocols can be formally related in fruitful ways. This chapter is an example of such a relation.

The models presented in the previous chapters are based upon the Dolev Yao attacker model, in which the exchanged messages of the protocol are modelled as formal expressions of a term algebra (see e.g. Table 2.1 (left) of Chapter 2 and Table 4.1 of Chapter 4). The (cryptographic) operations, such as message pairing and encryption, are modelled as term constructors. In this setting, an adversary and its abilities can be modelled in terms of the messages the adversary knows (see e.g. Table 2.1 (right)). Furthermore, the security properties a protocol is supposed to achieve are also modelled formally (e.g. $\mathcal{PS}$-LTL of Section 2.5). In these chapter we refer to these analysis models as the *formal* models.

Besides formal models, other analysis models have been developed within the cryptographic community, known as computational models [43]. In these models, messages are considered to be (more realistically) bit-strings, while cryptographic operations are seen as functions over these bit-strings. Here, an adversary is modelled as any efficient algorithm, while the security properties of a cryptographic protocol are defined on terms of the probability of the adversary to perform a successful attack [98]. The two models are illustrated in Figure 6.1.

Both of the two above models have advantages and disadvantages. On the one hand, the formal model allows to reason about cryptographic protocols more easily and generally. However, such benefits arise from the adoption of fairly strong assumptions (such as freeness of the term algebra, and fixing the adversary model). On the other hand, the computational model, by considering messages as bit-strings and modelling the adversary as any efficient algorithm, provides a more realistic model and thus offers more convincing security guarantees. However, proving protocols correct in the computational model is more difficult and less general than in the formal model.

In the work of Abadi and Rogaway [27], it is shown that if two formal expressions are similar to a formal adversary, then their corresponding computational interpretations, represented as bit-strings in the computational model, are also indistinguishable to any computational adversary (the attacker is considered to be passive). This is the soundness arrow of Figure 6.1 (for the completeness part, see below in the related work). This result comprises

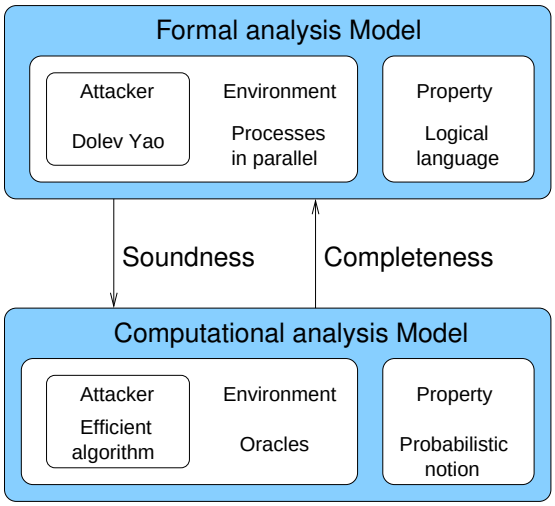


Figure 6.1: Relating formal analysis models and computational analysis models

a very important step into relating the formal and computational model.

Composed Keys. Abadi and Rogaway [27] model formal encryption by using atomic keys: that is, a formal expression $\{M\}_K$ represents encryption of message M with key K, where M is again a formal expression and K is an atomic symbol representing the cryptographic key. However, considering only atomic keys in encryption is not sufficient, and sometimes we need to be able to allow encryption with *composed keys*, representing non-atomic, constructed keys. In that setting, the formal language would need to be able to consider expressions of the form $\{M\}_N$, where both M and N are expressions. Considering composed keys as possible encryption keys is important due to that, in protocol design, it is fairly common to construct symmetric keys from shared secrets and other exchanged data as part of the protocol run. Examples of this can be found in the work of Gong [100], and, more recently, in a proposed protocol for achieving private authentication [19], which we study on Section 3.6.2. Moreover, many "real-world" cryptographic protocols use composed keys —see, for example SSL 3.0 [95].

Section 6.3 defines a computational interpretation $[\![\cdot]\!]$ for the operation $\{M\}_N$. Briefly, the interpretation $[\![\{M\}_N]\!]$ consists of encrypting $[\![M]\!]$ — the interpretation of M with a key obtained by applying the *random oracle* to $[\![N]\!]$. So, the interpretation of $\{M\}_N$ is quite intuitive. On the other hand, this forces us to use the *random oracle model* as the computational model. Using a random oracle seems to be necessary, since otherwise the goodness of keys might be questioned, as well as the independence of different keys.

We also define a relation $\cong$ over formal expressions and, as the main contribution of this chapter, we show that $M \cong N$ implies the computational indistinguishability of $[\![M]\!]$ and $[\![N]\!]$.

In the context of the present thesis, this chapter is important since reestablishing the soundness result of Abadi and Rogaway for the case of composed keys allow us to cover our

constraint solving approach of Section 2, which supports encryption with non-atomic keys explicitly.

Related Work. The work of Abadi and Rogaway [27] was later extended in Abadi and Jürjens [26] and Laud [125]. In these works, similar soundness results were obtained for richer formal languages, where instead of considering values of formal expressions, it is dealt with *outputs* of programs. However, differently from the formal language presented in this section, both of these extended languages still treat the encryption operation as using atomic keys.

Micciancio and Warinschi [145] considered the converse of the soundness result (i.e., completeness of the formal language of [27]). In their work, it is shown that a completeness result can be obtained by considering a stronger encryption scheme, namely an authenticated encryption scheme.

Further extensions of the seminal work [27] deal with *encryption cycles* in expressions. For instance, the expression $\{K\}_K$ contains a trivial cycle: key K is immediately encrypted with itself. In the computational model, the security of a traditional encryption scheme can be compromised if an adversary gets hold of a message containing an encryption cycle. Thus, in the original work of Abadi and Rogaway, formal expressions were restricted to be cycle free. However, further work of Black et al. [54] and Laud [126] has shown that, in fact, this discrepancy can be addressed in two different ways: either by considering a new, stronger security definition of the encryption scheme [54], or by strengthening the adversary model of the formal model, such that it can be able to "break" encryption cycles [126].

Several papers from the literature (see e.g. [41, 113]) continued exploring the relation between formal and computational models, and this is becoming an active area of research. This is evidenced, for example, by the recent organization of a workshop dedicated specifically to this topic (the "Workshop on the link between formal and computational models", Paris, June 2005).

Finally, Bellare and Kohno [44] have studied the security of cryptosystems against *related-key* attacks and also provided a construction of a secure cryptosystem against a certain kind of such attacks. Related keys are different from composed keys — a related key is something that is constructed from an already existing good key and some non-key data, whereas a composed key is constructed from non-key data only.

In Section 6.2 we present the formal language. Then, in Section 6.3 we introduce some basic notions of the computational model that are needed in the sequel, and also present an algorithm for translating formal expressions into computational [distributions of] bit-strings. In Section 6.4, we introduce an equivalence relation $\cong$ over formal expressions. This equivalence relation $\cong$ is elaborated and illustrated with some examples in Section 6.5. After that, in Section 6.6 we present the main contribution, a soundness result that relates the formal and computational models. Finally, Section 6.7 concludes the chapter.

6.2 Expressions and Patterns

Let $\mathbf{Bool}$ be the set $\{0, 1\}$ and let $\mathbf{Keys}$ be the set of *formal keys* — this is a fixed, infinite set of symbols. Intuitively, elements of $\mathbf{Keys}$ represent cryptographic keys. Also, let $\mathbf{Rnd}$ be the set of *formal random numbers* — again a fixed, infinite set of symbols disjoint

from $\mathbf{Keys}$. The use of $\mathbf{Rnd}$ is needed since usually some of the constructors of formal expressions (such as encryption) represent probabilistic operations. This means that if such an operation is executed twice, even with the same arguments, the results will be different. Thus, the elements of $\mathbf{Rnd}$ are used to keep track which subexpressions of an expression represent the same invocation of that operation and which subexpressions represent different invocations. Our set of formal expressions $\mathbf{Exp}$ is defined by the following grammar:

$$\begin{array}{rcll} M, N & ::= & b & \text{(bit)} \\ & | & K & \text{(key)} \\ & | & (M, N) & \text{(pair)} \\ & | & \{M\}^r_N & \text{(encryption)} \ . \end{array}$$

Here, $b \in \mathbf{Bool}$, $K \in \mathbf{Keys}$ and $r \in \mathbf{Rnd}$. Clearly, we can see that composed keys are allowed in the encryption operation. As the labels r are identifiers of invocations of the encryption algorithm, we demand that whenever we consider two expressions $\{M\}^r_N$ and $\{M'\}^r_{N'}$ with the same label r, then also $M = M'$ and $N = N'$.

Even though Abadi and Rogaway [27] did not use formal random numbers, they assumed that each occurrence of the encryption constructor represents a different invocation of the encryption operation. Furthermore, the later work of Abadi and Jürjens [26] considered a richer language in which they also needed to keep track of different invocations. This was done, similarly to the present work, by using formal random numbers.

Let us define some notation related to the structure of formal expressions. The *subexpression relation* $\sqsubseteq$ is the smallest reflexive transitive relation over $\mathbf{Exp}$ containing $M \sqsubseteq (M, N)$, $N \sqsubseteq (M, N)$, $M \sqsubseteq \{M\}^r_N$ and $N \sqsubseteq \{M\}^r_N$ for all $M, N \in \mathbf{Exp}$ and $r \in \mathbf{Rnd}$. For an expression M, we denote:

$$\begin{aligned} keys(M) &:= \{K \in \mathbf{Keys} \ : \ K \sqsubseteq M\} \\ rns(M) &:= \{r \in \mathbf{Rnd} \ : \ \{N'\}^r_N \sqsubseteq M \text{ for some } N', N \in \mathbf{Exp}\} \\ atoms(M) &:= keys(M) \cup rns(M) \end{aligned}$$

We call the elements of $atoms(M)$ the *atoms* of M.

Intuitively, a *formal pattern* describes what an adversary is able to see when looking at an expression. The elements P, Q of the set of *formal patterns* $\mathbf{Pat}$ is defined by the following grammar:

$$\begin{array}{rcll} P, Q & ::= & b & \text{(bit)} \\ & | & K & \text{(key)} \\ & | & (P, Q) & \text{(pair)} \\ & | & \{P\}^r_Q & \text{(encryption)} \\ & | & \Box^r & \text{(undecryptable)} \ . \end{array}$$

Here, $\Box^r$ denote ciphertexts that are encrypted with a key that the adversary does not know, and thus can not "see" inside. We use formal random numbers to differentiate between these ciphertexts, and therefore we require that the formal random numbers used at encryptions be different from formal random numbers used at undecryptables. Now, the relation $\sqsubseteq$, as well as the functions $keys$, rns and $atoms$ are extended to $\mathbf{Pat}$. Finally, note that the sets $rns(P)$ and $atoms(P)$ also contain formal random numbers at the undecryptables.

6.3 Computational Interpretation

In the computational model, an encryption system is a triple of polynomial-time algorithms $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ working with bit-strings. Here, $\mathcal{G}$ and $\mathcal{E}$ are probabilistic algorithms while $\mathcal{D}$ is deterministic. The *key generation algorithm* $\mathcal{G}$ takes as input the security parameter n, represented in unary, and returns a new key. The *encryption algorithm* $\mathcal{E}$ takes as input the security parameter, a key and a plaintext and produces a corresponding ciphertext. Since $\mathcal{E}$ is probabilistic, different invocations of $\mathcal{E}$ may return different ciphertexts. Lastly, the *decryption algorithm* $\mathcal{D}$ takes as input the security parameter, a key and a ciphertext and returns the corresponding plaintext.

Let $\mathbf{0}$ be a fixed bit-string. We say that the encryption system $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is *type-0 secure* [27] if, for all probabilistic polynomial-time (PPT) algorithms $\mathcal{A}^{(\cdot),(\cdot)}$ (with interfaces to two oracles), the difference of probabilities

$$\Pr[\mathcal{A}^{\mathcal{E}(1^n,k,\cdot),\mathcal{E}(1^n,k',\cdot)}(1^n) = 1 \,:\, k, k' \leftarrow \mathcal{G}(1^n)] - \Pr[\mathcal{A}^{\mathcal{E}(1^n,k,\mathbf{0}),\mathcal{E}(1^n,k,\mathbf{0})}(1^n) = 1 \,:\, k \leftarrow \mathcal{G}(1^n)]$$

is negligible in n. A function is negligible if its reciprocal grows faster than any polynomial. In [27], Abadi and Rogaway showed that type-0 security is achievable under standard cryptographic assumptions.

Let $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ be a type-0 secure encryption system, such that the distribution $\mathcal{G}(1^n)$ is the uniform probability distribution over $\{0,1\}^{\ell(n)}$, where ℓ is a fixed polynomial. Now, being type-0 secure guarantees that the algorithm $\mathcal{E}$ is probabilistic, and thus we denote by $\mathcal{E}^{\mathbf{r}}$ the invocation of $\mathcal{E}$ with random coin-flips $\mathbf{r} \in \{0,1\}^*$. Thus, if we fix $\mathbf{r}$, the algorithm $\mathcal{E}^{\mathbf{r}}$ is now deterministic. In the security definition, we assume the uniform distribution of $\mathbf{r}$.

Now, let x be a bit-string. A *random oracle* R is a machine that, on query $(1^m, x)$, first checks whether it has been queried with the same values before. If this is the case, then it returns the same answer as before. Otherwise, it proceeds as follows. First, the random oracle creates, uniformly and randomly, a bit-string y of length m. Then, the random oracle records the query $(1^m, x)$ together with y, and then finally y is returned. In the *random oracle model* [45], there is a single random oracle in the world, while all other algorithms and machines are allowed to query this oracle. To be able to translate a model in the random oracle world into a real system, the random oracle needs to be replaced with some "random-looking" function h. Thus, there is a leap of faith involved in applying the results proved in the random oracle model to a real system. Nevertheless, we can still be sure that if the real system is insecure, then this must be caused by h not being a good approximation of R.

Now we are ready to give a computational interpretation to expressions and patterns. With each $P \in \mathbf{Pat}$ we associate a family (indexed by the security parameter) of probability distributions over bit-strings. We denote that family by $[\![P]\!]$. Figure 6.2 depicts the algorithm sampling the n-th distribution in that family. First, INITIALIZE$(1^n, P)$ is run and then CONVERT$(1^n, P)$ is invoked.

Note that if $P_1 \neq P_2$ then if we sample $\langle x, y, \text{"pair"} \rangle \leftarrow [\![(P_1, P_2)]\!]_n$, then the probability for $x = y$ is negligible.

In fact, the existence of the random oracle is itself sufficient for the existence of type-0 encryption systems. In particular, we could have fixed the encryption system, for example, to

algorithm INITIALIZE$(1^n, P)$
  for all $K \in keys(P)$ do $\tau(K) \leftarrow \mathcal{G}(1^n)$
  $\tau_{bb} \leftarrow \mathcal{G}(1^n)$
  for all $r \in rns(P)$ do $\tau(r) \stackrel{R}{\in} \{0,1\}^*$

algorithm CONVERT$(1^n, P)$
  if P is $K \in \mathbf{Keys}$
    return $\langle \tau(K), \text{“key”} \rangle$
  else if P is $b \in \mathbf{Bool}$
    return $\langle b, \text{“bit”} \rangle$
  else if P is (P_1, P_2)
    let $x = \text{CONVERT}(1^n, P_1)$
    let $y = \text{CONVERT}(1^n, P_2)$
    return $\langle x, y, \text{“pair”} \rangle$
  else if P is $\{P_2\}^r_{P_1}$
    let $x = \text{CONVERT}(1^n, P_1)$
    let $y = \text{CONVERT}(1^n, P_2)$
    let $z = \mathcal{E}^{\tau(r)}(1^n, R(1^{\ell(n)}, x), y)$
    return $\langle z, \text{“ciphertext”} \rangle$
  else: P is $\square^r$
    let $z = \mathcal{E}^{\tau(r)}(1^n, R(1^{\ell(n)}, \tau_{bb}), \mathbf{0})$
    return $\langle z, \text{“ciphertext”} \rangle$

Figure 6.2: Algorithm sampling $[\![P]\!]$

the one given in [54]. However, we would like to use the random oracle as little as possible and thus we have not fixed it.

Two families of probability distributions over bit-strings D and D' are *indistinguishable* (denoted $D \approx D'$) if for all PPT algorithms $\mathcal{A}$, the difference of probabilities

$$\Pr[\mathcal{A}(1^n, x) \,:\, x \leftarrow D_n] - \Pr[\mathcal{A}(1^n, x) \,:\, x \leftarrow D'_n]$$

is negligible in n. In fact, indistinguishability is the computational equivalent of sameness.

6.4 Equivalence Relation on Pat

We would like to define an equivalence relation $\cong$ over formal expressions (and more generally, over patterns), such that $M \cong N$ implies $[\![M]\!] \approx [\![N]\!]$. Similarly to Abadi and Rogaway, we define a function $pattern : \mathbf{Exp} \longrightarrow \mathbf{Pat}$ and state $M \cong N$ iff $pattern(M)$ and $pattern(N)$ can be obtained from each other by an α-conversion over keys and formal random numbers. Even though we could also define the function $pattern$ similarly to [27], that is by giving the entailment relation $\vdash$ (this relation describes which formal expressions the Dolev Yao attacker may obtain from a given expression) and replacing the

undecryptable encryptions in the expressions by the corresponding "blobs" $\Box$, we chose to give a different equivalence definition. The reason for this is that, if we followed Abadi and Rogaway, then we would have to assume that the expressions M and N do not contain *encryption cycles*. With atomic keys only, an encryption cycle in an expression M is a sequence of keys $K_1, \ldots, K_m$, where K_i is encrypted under K_{i+1} (possibly indirectly) for all $i \in \{1, \ldots, m-1\}$ and K_m is encrypted under K_1, all in the expression M. Even though the definition of type-0 security does not say anything about the security of encryption cycles; in systems where the Abadi and Rogaway results can be applied, encryption cycles cannot occur. However, when considering composed keys, the definition of encryption cycles is likely much more contrived, because the different parts of the same key have to be kept track of. Therefore, we avoid defining the encryption cycles at all, and thus our definition of $\cong$ applies to all expressions.

Let $P, Q \in \mathbf{Pat}$. The operation $\mathsf{box}_Q(P)$ replaces all encryptions of the form $\{\cdot\}^r_Q$ occurring in P with undecryptables. Formally, the operation $\mathsf{box}_Q(P)$ is defined by

$$\begin{aligned}
\mathsf{box}_Q(b) &= b \\
\mathsf{box}_Q(K) &= K \\
\mathsf{box}_Q((P_1, P_2)) &= (\mathsf{box}_Q(P_1), \mathsf{box}_Q(P_2)) \\
\mathsf{box}_Q(\{P_2\}^r_{P_1}) &= \begin{cases} \Box^r, & \text{if } P_1 = Q \\ \{\mathsf{box}_Q(P_2)\}^r_{\mathsf{box}_Q(P_1)}, & \text{if } P_1 \neq Q \end{cases} \\
\mathsf{box}_Q(\Box^r) &= \Box^r
\end{aligned}$$

We are looking for sufficient conditions for $[\![P]\!] \approx [\![\mathsf{box}_Q(P)]\!]$. In particular, we are going to prove that the following is a sufficient condition. Let T_P be the set of all atoms occurring in P, except that if P has subexpressions of the form $\{\cdot\}^r_Q$, then the keys and random numbers inside that Q do not count. The sufficient condition that we are looking for is $atoms(Q) \not\subseteq T_P$. To state this formally, we define the sets $\mathcal{B}_Q(P)$ for all $P \in \mathbf{Pat}$ in the following way:

$$\begin{aligned}
\mathcal{B}_Q(b) &= \emptyset \\
\mathcal{B}_Q(K) &= \{K\} \\
\mathcal{B}_Q((P_1, P_2)) &= \mathcal{B}_Q(P_1) \cup \mathcal{B}_Q(P_2) \\
\mathcal{B}_Q(\{P_2\}^r_{P_1}) &= \begin{cases} \{r\} \cup \mathcal{B}_Q(P_2), & \text{if } P_1 = Q \\ \{r\} \cup \mathcal{B}_Q(P_1) \cup \mathcal{B}_Q(P_2), & \text{if } P_1 \neq Q \end{cases} \\
\mathcal{B}_Q(\Box^r) &= \{r\}
\end{aligned}$$

and set $T_P := \mathcal{B}_Q(P)$.

If the above condition is fulfilled we say that $P \cong \mathsf{box}_Q(P)$. We also say $P \cong Q$ whenever Q can be obtained from P through some α-conversion applied to its formal keys and random numbers. Finally, we extend $\cong$ to an equivalence relation.

Now we are ready to define the function $pattern$. Let $P \in \mathbf{Pat}$. If there exists some $Q \in \mathbf{Pat}$, such that $P \neq \mathsf{box}_Q(P)$ (i.e. Q occurs as an encryption key somewhere in P) and $P \cong \mathsf{box}_Q(P)$ then we put $pattern(P) := pattern(\mathsf{box}_Q(P))$. Otherwise we put $pattern(P) := P$. It is easy to check that $pattern$ is well-defined. Furthermore, the function $pattern$ is efficiently computable.

6.5 Examples

Before going to the proof that equivalence implies indistinguishability of interpretations, let us see some examples. We are not going to repeat the examples given by Abadi and Rogaway [27][1]. In our examples we intend to illustrate and clarify what constitutes an "encryption cycle" in an expression and what does not.

- $\{K_1\}^r_{(K_1,K_2)} \cong \Box^r$. This is so since the atom K_2 of the encryption key (K_1, K_2) does not occur anywhere else.
- $\{(K_1, K_2)\}^r_{(K_1,K_2)} \ncong \Box^r$. This expression is a clear-cut encryption cycle. However, encryption cycles can be more subtle, as the next two examples show.
- $\{(K_2, K_1)\}^r_{(K_1,K_2)} \ncong \Box^r$.
- $(\{K_1\}^{r_1}_{(K_1,K_2)}, \{K_2\}^{r_2}_{(K_1,K_2)}) \ncong (\Box^{r_1}, \Box^{r_2})$.
- $\{\{K_2\}^{r_1}_{K_1}\}^r_{\{K_2\}^{r_1}_{K_1}} \ncong \Box^r$, but $\{\{K_2\}^{r_2}_{K_1}\}^r_{\{K_2\}^{r_1}_{K_1}} \cong \Box^r$. The first example contains an encryption cycle. The second example, however, does not, because the atom r_1 of the key does not occur anywhere else. These two examples show the importance of formal random numbers.
- $(\{K_1\}^r_{(K_1,K_2)}, K_2) \ncong (\Box^r, K_2)$. Compared to the first example, the addition of K_2 means that now all atoms of the encryption key occur somewhere else.

6.6 Correctness

We now present our main result of this chapter, which establishes the soundness of the formal model w.r.t. the computational counterpart.

Theorem 6.6.1. *Let $P, Q \in \mathbf{Pat}$, such that $atoms(Q) \not\subseteq \mathcal{B}_Q(P)$. Then $[\![P]\!] \approx [\![\mathsf{box}_Q(P)]\!]$.*

Proof. See Appendix A.3. □

6.7 Conclusions

In this chapter we have considered an extension of the work of Abadi and Rogaway [27]. This extension is mainly constituted by considering the use of composed, non-atomic keys in the encryption operator of the formal language. Briefly, we proceeded as follows: First, we related formal expressions in our language with an equivalence relation $\cong$. By providing an intuitive computational interpretation, and then showing that each time two formal expressions that are equivalent according to $\cong$ are also indistinguishable in the computational world, we have lifted the work of Abadi and Rogaway [27] to the case of composed keys.

[1] The reader checking out these examples should keep in mind that:

- [27] uses no formal random numbers; each occurrence of the encryption constructor is assumed to have a different formal random number attached to it;
- in [27], $M \cong N$ does not imply $[\![M]\!] \approx [\![N]\!]$, if M or N contains encryption cycles.

As we already mentioned, support for encryption with composed keys is important since many cryptographic protocols use them [100, 19, 95]. Thus, having the soundness result for the case of formal encryption with composed keys provides further faithfulness in the verification results of formal approaches that support composed keys (such as the constraint solving approach of Chapter 2 and the works [149, 5, 38]).

While giving the computational interpretation, we needed to use the random oracle. Thus, our approach gives less security guarantees than the original work of Abadi and Rogaway, based on standard security assumptions. However, we believe the use of the random oracle is necessary to guarantee the goodness and independence of the constructed keys. Usage of the random oracle allow us to model the situation in which a user generates keys in a completely secure manner, which is in accordance with the existing definitions in the computational model. However, in some situations (e.g. when considering composed keys), the key generation process may *not* be a so private activity. In this new setting, an adversary might have some knowledge about the randomness used during the key generation. Furthermore, a stronger and *active* adversary may even have some control over the key generation process. We believe it would be interesting to study such a new scenario, where new and proper definitions (and constructions) would be needed.

CHAPTER 7

Concluding Remarks

In this thesis we study the analysis of security protocols. Our extended analysis models instantiate (and relate) the Dolev Yao analysis model of the Introduction in several new, realistic settings, as illustrated in Figure 7.1.

Our primary aim is to develop methods that provide practical, efficient and effective means to analyse security protocols, that can be readily applied by protocol designers to help in the development of real-life security protocols.

In this thesis we achieve this aim successfully, allowing us to state, as the main conclusion, that the development of effective analysis models for security protocols is both feasible in theory and useful in practice. We support this conclusion by describing our explorations, which span a range of different activities:

- We develop novel automatic procedures to analyse security protocols (e.g. constraint solving);
- We create novel definitions that model security properties (e.g. definition of guessing attacks in constraint solving and in the Applied Pi Calculus);
- We show how to cast realistic analysis models within generic formal frameworks (e.g. the modelling of security protocols in timed automata);
- We formally model industrial specifications (e.g. formalization of session based web service specifications in TulaFale, and modelling of the OSA/Parlay framework in constraint solving);
- We formally relate analysis models and their security proofs (e.g. relations between the formal and computational models of security).

Each of our methods supports the analysis of existing protocols (both from the literature and industrial), in some cases uncovering potential vulnerabilities. Furthermore, we also design novel protocols. We comment on some of these protocols (the complete list is in the List of Protocols, see the Table of Contents):

- Analysis of guessing attacks for the EPT and EKE protocols. For EKE, we uncovered a potential vulnerability and proposed a new improved version.

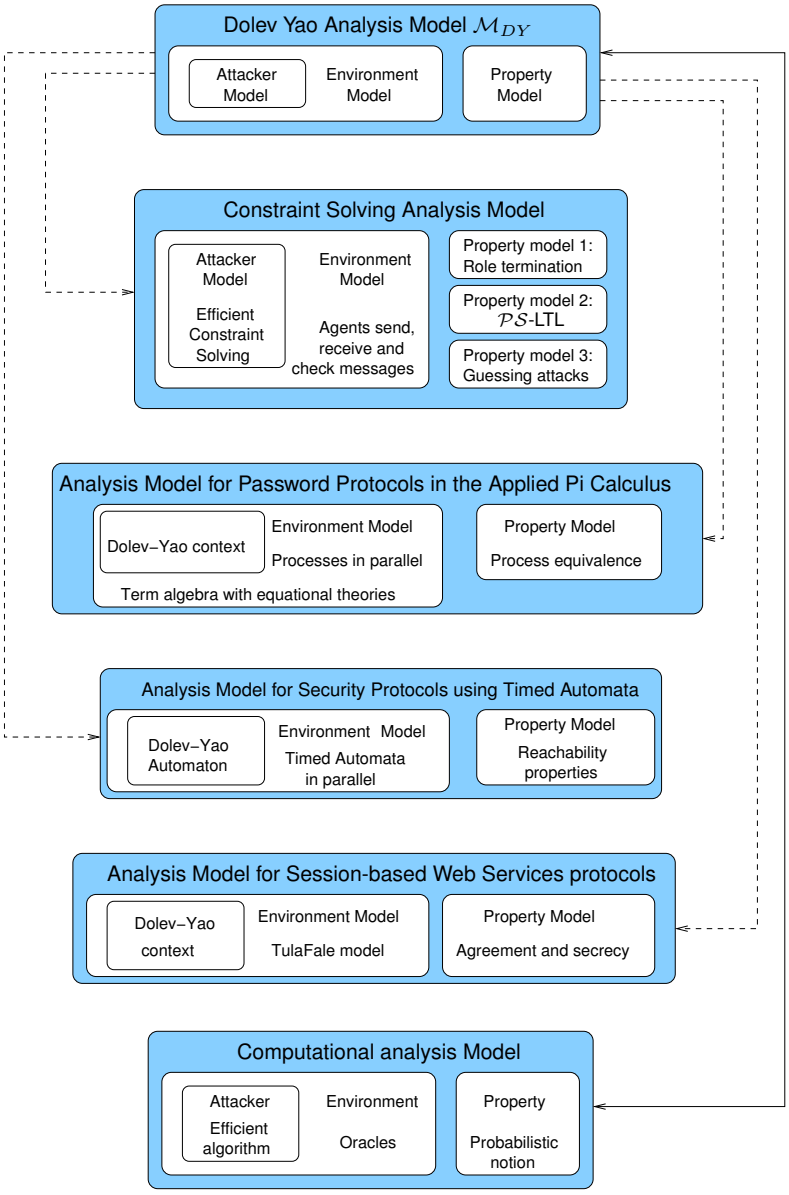


Figure 7.1: The Dolev Yao analysis model and its extensions (each dashed arrow is an instantiation) and relation (solid line)

Contribution	Practical	Theoretical
Chapter 2 (Constraint Solving)		
Section 2.3 (Improved Verifier)	Procedure 2.3.4	Theorem 2.3.6
Section 2.5 ($\mathcal{PS}$-LTL)	Procedure 2.5.13	Theorem 2.5.15
Section 2.7 (Guessing Attacks)	Definitions 2.7.2 and 2.7.4	Proposition 2.7.5
Chapter 3 (Timed Analysis)		
Sections 3.2-3.4	TA model checking	
Section 3.5	Literature Taxonomy	(n.a.)
Section 3.6	New issues with time	
Chapter 4 (GA in APC)		
Section 4.2		Definition 4.2.5
Sections 4.3 and 4.4	Analysis of EPT and EKE	
Chapter 5 (Session-based WS)		
Section 5.3	WS-Trust specification	
Section 5.4	WS-SC specification	
Section 5.4.2	Open-ended sessions	Theorem 5.4.2
Section 5.5	Interop analysis	
Chapter 6 (Relation)		
Section 6.6 (Soundness)	(n.a.)	Theorem 6.6.1

Table 7.1: Practical and Theoretical contributions of this thesis (here, 'TA' stands for 'Timed Automata', 'GA in APC' for 'Guessing Attacks in the Applied Pi Calculus', 'WS' stands for 'Web Services' and 'SC' stands for 'SecureConversation')

- Analysis and establishment of authentication properties of a web service protocol used in the Interoperability (industrial) workshop to test web services implementations of the WS-Trust and WS-SecureConversation specifications.
- Analysis of Abadi's private authentication protocol, exposing a potential timing attack if the protocol were to be implemented naively;
- Design of a novel authentication protocol that uses timed challenges as opposed to nonces for efficiency, by exploiting timeliness of messages;
- Analysis of case studies: protocols for wireless sensor networks, content management and the OSA/Parlay authentication protocol.

While developing the practical contributions, several theoretical results were also developed to analyse the correctness of the proposed practical approaches. The practical and theoretical contributions for each chapter are shown in Table 7.1.

Future Work Just like security protocols use security constructions (e.g. cryptographic primitives like message encryption) as building blocks, *security languages* are higher level constructions that use security protocols as building blocks. Table 7.2 shows example constructions for security primitives, protocols (studied in this thesis) and languages. A typical security primitive is message encryption; likewise, a typical security protocol is an

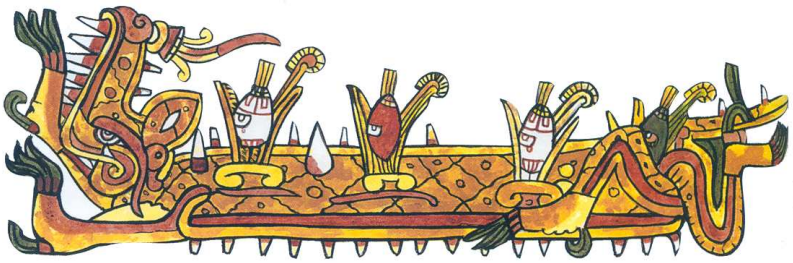
Kind	Example Construction	Example Property
Security Primitive	Message Encryption	Attacker does not glean information from ciphertext
Security Protocol	Authentication protocol	Attacker does not impersonate other participants
Security Language	Policy language	Attacker does not perform action disallowed by policy

Table 7.2: Security Primitives, Protocols and Languages

authentication protocol. An example of a higher level security language is a *policy* language. In this case, the security language is designed to specify *policies* that regulate access to particular data. There, logical languages can be designed to specify such access policies, and proof systems can be developed to reason about whether certain actions are allowed to be executed from particular policies [13, 14]. The secure transfer of these policies may be then deployed using security protocols.

We conclude by suggesting that, as possible future work, it would certainly be interesting to formally relate high level models for security languages to the (underlying) lower level models for security protocols, in a similar way in which security protocols are related to their underlying security primitives.

List of Protocols



Bibliography

Author References

Security Protocols

[1] K. Bhargavan, R. Corin, C. Fournet, and A. D. Gordon. Secure sessions for web services. In *ACM Workshop on Secure Web Services (SWS)*, Fairfax, Virginia, Oct 2004. ACM Press, New York. **(Subsumed by Chapter 5 of this thesis).**

[2] J. Cederquist, R. Corin, and M. T. Dashti. On the quest for impartiality: Design and analysis of a fair non-repudiation protocol. In *7th Int. Conf. on Information and Communications Security (ICICS)*, Beijing, China, Dec 2005. Springer-Verlag, Berlin.

[3] R. Corin, J. M. Doumen, and S. Etalle. Analysing password protocol security against off-line dictionary attacks. In N. Busi, R. Gorrieri, and F. Martinelli, editors, *2nd Int. Workshop on Security Issues with Petri Nets and other Computational Models (WISP)*, volume 121, pages 47–63, Bologna, Italy, Jun 2004. Electronic Notes in Theoretical Computer Science. **(Subsumed by Chapter 4 of this thesis).**

[4] R. Corin, A. Durante, S. Etalle, and P. H. Hartel. A trace logic for local security properties. In *Int. Workshop on Software Verification and Validation (SVV)*, volume 118, pages 129–143, Mumbai, India, Dec 2003. Elsevier Science in Electronic Notes in Theoretical Computer Science. **(Subsumed by Chapter 2, Section 2.5 of this thesis).**

[5] R. Corin and S. Etalle. An improved constraint-based system for the verification of security protocols. In M. V. Hermenegildo and G. Puebla, editors, *9th Int. Static Analysis Symp. (SAS)*, volume LNCS 2477, pages 326–341, Madrid, Spain, Sep 2002. Springer-Verlag, Berlin. **(Subsumed by Chapter 2, Sections 2.3 and 2.6 of this thesis).**

[6] R. Corin, S. Etalle, P. H. Hartel, and A. Mader. Timed analysis of security protocols. *Journal of Computer Security (JCS)*, 2006. (This papers subsumes the paper: *Timed Model Checking of Security Protocols*, ACM FMSE2004.) **(Subsumed by Chapter 3 of this thesis).**

[7] R. Corin, S. Etalle, and A. Saptawijaya. Online demos. Improved Constraint-based procedure at `http://130.89.144.15/cgi-bin/show.cgi`. $\mathcal{PS}$-LTL demo at `http://130.89.144.15/cgi-bin/psltl/show.cgi`, June 2005.

[8] R. Corin, S. Malladi, J. Alves-Foss, and S. Etalle. Guess what? here is a new tool that finds some new guessing attacks (extended abstract). In R. Gorrieri and R. Lucchi, editors, *IFIP WG 1.7 and ACM SIGPLAN Workshop on Issues in the Theory of Security (WITS)*, pages 62–71, Warsaw, Poland, Apr 2003. Dipartamento di Scienze

dell'Informazione Universita di Bologna, Italy. **(Subsumed by Chapter 2, Section 2.7 of this thesis).**

[9] P. Laud and R. Corin. Sound computational interpretation of formal encryption with composed keys. In Jong In Lim and Dong Hoon Lee, editors, *6th Annual Int. Conf. on Information Security and Cryptology (ICISC)*, volume LNCS 2971, pages 55–66, Seoul, Korea, Nov 2003. Springer-Verlag, Berlin. **(Subsumed by Chapter 6 of this thesis).**

Case Studies (Described in Section 2.9)

[10] C. N. Chong, B. Ren, J. M. Doumen, S. Etalle, P. H. Hartel, and R. Corin. License protection with a tamper-resistant token. In C.-H. Lim and M. Yung, editors, *5th Workshop on Information Security Applications (WISA)*, volume LNCS 3325, pages 223–237, Jeju Island, Korea, Aug 2004. Springer-Verlag, Berlin.

[11] R. Corin, G. Di Caprio, S. Etalle, S. Gnesi, G. Lenzini, and C. Moiso. Security analysis of parlay/OSA framework II. In M. Steffen and G. Zavattaro, editors, *7th IFIP Int. Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS)*, Athens, Greece, Jun 2005. Kluwer Academic Publishers, Boston.

[12] Y. W. Law, R. Corin, S. Etalle, and P. H. Hartel. A formally verified decentralized key management architecture for wireless sensor networks. In M. Conti, S. Giordano, E. Gregori, and S. Olariu, editors, *4th IFIP TC6/WG6.8 Int. Conf on Personal Wireless Communications (PWC)*, volume LNCS 2775, pages 27–39, Venice, Italy, Sep 2003. Springer-Verlag, Berlin.

Security Languages

[13] J. Cederquist, R. Corin, M. A. C. Dekker, S. Etalle, and J. I. den Hartog. An audit logic for accountability. In A. Sahai and W. Winsborough, editors, *6th Int. Workshop on Policies for Distributed Systems & Networks (POLICY)*, Stockholm, Sweden, Jun 2005. IEEE Computer Society Press, Los Alamitos, California. (This paper subsumes *A Logic for Auditing Accountability in Decentralized Systems*, FAST2004, published by Springer).

[14] C. N. Chong, R. Corin, J. M. Doumen, S. Etalle, P. H. Hartel, Y. W. Law, and A. Tokmakoff. LicenseScript: A logical language for digital rights management. *Annals of telecommunications special issue on Information systems security*, 2006. (This paper subsumes *LicenseScript: A Novel Digital Rights Language and its Semantics*, Wedelmusic 2003, published by IEEE Computer Society Press).

Other References

[15] Security Protocols Open Repository (SPORE). At `http://www.lsv.ens-cachan.fr/spore/`, INRIA, March 2005.

[16] AVISPA: Automated Validation of Internet Security Protocols and Applications, 2003. At `http://www.avispa-project.org`.

[17] *Open Service Access (OSA) - Application Programming Interface (API) Mapping for OSA*. http://www.3gpp.org/ftp/Specs/archive/29_series. Release 5.

[18] M. Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 46(5):749–786, 1999.

[19] M. Abadi. Private authentication. In *Proceedings of the 2002 Workshop on Privacy Enhancing Technologies*, LNCS, pages 27–40. Springer-Verlag, 2002.

[20] M. Abadi and B. Blanchet. Analyzing Security Protocols with Secrecy Types and Logic Programs. In *29th ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages (POPL 2002)*, pages 33–44, Portland, Oregon, 2002. ACM Press.

[21] M. Abadi and B. Blanchet. Computer-assisted verification of a protocol for certified email. In Radhia Cousot, editor, *Static Analysis: 10th International Symposium, SAS 2003, San Diego, CA, USA, June 11-13, 2003. Proceedings*, volume 2694 of *Lecture Notes in Computer Science*, pages 316–335, Heidelberg, 2003. Springer-Verlag.

[22] M. Abadi, B. Blanchet, and C. Fournet. Just fast keying in the pi calculus. In *Proceedings of the 13th European Symposium on Programming (ESOP'04)*, volume 2986 of *LNCS*, pages 340–354. Springer, 2004.

[23] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115. ACM, January 2001.

[24] M. Abadi and C. Fournet. Private authentication. *Theor. Comput. Sci.*, 322(3):427–476, 2004.

[25] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148:1–70, 1999.

[26] M. Abadi and J. Jurjens. Formal eavesdropping and its computational interpretation. In *Fourth International Symposium on Theoretical Aspects of Computer Software (TACS2001)*, LNCS. Springer-Verlag, 2001.

[27] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *Journal of Cryptology*, number 15, pages 103–127. Springer-Verlag, 2000.

[28] M. Abadi and B. Warinschi. Password-based encryption analyzed. In L. Caires, G. Italiano, L. Monteiro, C. Palamidessi, and M. Young, editors, *International Colloqium on Automata, Languages and Programming – ICALP'05*, volume 3580 of *LNCS*, pages 664–676. Springer, July 2005.

[29] M. Abdalla and M. Bellare. Increasing the lifetime of a key: A comparative analysis of the security of re-keying techniques. In *ASIACRYPT '00: Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security*, pages 546–559, London, UK, 2000. Springer-Verlag.

[30] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, (138):183–335, 1994.

[31] R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *CONCUR*, pages 380–394, 2000.

[32] R. Amadio, D. Lugiez, and V. Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, To appear:–, 2002.

[33] T. Amnell, G. Behrmann, J. Bengtsson, P. R. D'Argenio, A. David, A. Fehnker, T. Hune, B. Jeannet, K. G. Larsen, M. O. Möller, P. Pettersson, C. Weise, and W. Yi. UPPAAL - Now, Next, and Future. In F. Cassez, C. Jard, B. Rozoy, and M. D. Ryand, editors, *4th Summer School on Modeling and Verification of Parallel Processes (MOVEP)*, volume LNCS 2067, pages 99–124, Nantes, France, Jun 2000. Springer-Verlag, Berlin. `http://www.springerlink.com/link.asp?id=8km1eyyvfqfm740f`.

[34] K. R. Apt. *From Logic Programming to Prolog*. International Series in Computer Science. Prentice Hall, 1997.

[35] N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 86–99, Oakland, CA, 1998. IEEE Computer Society, Technical Committee on Security and Privacy, IEEE Computer Society Press. A minor bug in the proceedings version was fixed. An errata sheet, distributed at the conference, is available at `http://www.zurich.ibm.com/Technology/Security/publications/1998/ASW98-errata.ps.gz`.

[36] T. Aura, A. Nagarajan, and A. Gurtov. Analysis of the hip base exchange protocol. In *ACISP*, pages 481–493, 2005.

[37] D. Basin. Lazy infinite-state analysis of security protocols. In *Secure Networking — CQRE [Secure] '99*, number 1740 in Lecture Notes in Computer Science, pages 30–42, Düsseldorf, Germany, November 1999. Springer-Verlag.

[38] D. Basin, S. Mödersheim, and L. Viganò. Constraint differentiation: A new reduction technique for constraint-based analysis of security protocols. In *Workshop on Security Protocol Verification. CONCUR 2003*, September 2003.

[39] D. Basin, S. Mödersheim, and L. Viganò. Ofmc: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, June 2005. Published online December 2004.

[40] M. Baudet. Deciding security of protocols against off-line guessing attacks. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS'05)*, Alexandria, Virginia, USA, November 2005. ACM Press.

[41] M. Baudet, V. Cortier, and S. Kremer. Computationally sound implementations of equational theories against passive adversaries. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, volume 3580 of *Lecture Notes in Computer Science*, pages 652–663, Lisboa, Portugal, July 2005. Springer.

[42] G. Bella and L. C. Paulson. Kerberos version IV: Inductive analysis of the secrecy goals. In J.-J. Quisquater, editor, *Proc. 5th European Symposium on Research in Computer Security*, volume 1485 of *LNCS*, pages 361–375, Louvain-la-Neuve, Belgium, 1998. Springer-Verlag.

[43] M. Bellare. Practice-oriented provable security. In *Information Security, First International Workshop, ISW '97*, LNCS 1396, pages 221–231, 1997.

[44] M. Bellare and T. Kohno. A theoretical treatment of related-key attacks: Rka-prps, rka-prfs, and applications. In *EUROCRYPT 2003*, LNCS 2656, pages 491–506, 2003.

[45] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[46] S. M. Bellovin and M. Merritt. Limitations of the kerberos authentication system. *SIGCOMM Comput. Commun. Rev.*, 20(5):119–132, 1990.

[47] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the 1992 IEEE Computer Security Conference on Research in Security and Privacy*, pages 72–84, 1992.

[48] B. Bérard, B. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and Ph. Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.

[49] K. Bhargavan, R. Corin, C. Fournet, and A. D. Gordon. Secure sessions for web services. Technical Report MSR-TR-2004-114, Microsoft Research, 2004.

[50] K. Bhargavan, C. Fournet, and A. D. Gordon. A semantics for web services authentication. In *31st ACM Symposium on Principles of Programming Languages (POPL'04)*, pages 198–209, 2004. An extended version appears as Microsoft Research Technical Report MSR–TR–2003–83.

[51] K. Bhargavan, C. Fournet, and A. D. Gordon. Verifying policy-based security for web services. In *11th ACM Conference on Computer and Communications Security (CCS'04)*, pages 268–277, 2004.

[52] K. Bhargavan, C. Fournet, A. D. Gordon, and R. Pucella. TulaFale: A security tool for web services. In *International Symposium on Formal Methods for Components and Objects (FMCO'03)*, volume 3188 of *LNCS*. Springer, 2004.

[53] S. Bistarelli, I. Cervesato, G. Lenzini, and F. Martinelli. Relating multiset rewriting and process algebras for security protocol analysis. *Journal of Computer Security*, 13(1):3–47, Feb 2005.

[54] J. Black, P. Rogaway, and T. Shrimpton. Encryption scheme security in the presence of key-dependent messages. In *Selected Areas in Cryptography — SAC'02*, LNCS. Springer-Verlag, 2002.

[55] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In S. Schneider, editor, *Proc. 14th IEEE Computer Security Foundations Workshop*, 2001.

[56] B. Blanchet. Automatic proof of strong secrecy for security protocols. Research Report MPI-I-2004-NWG1-001, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, July 2004.

[57] B. Blanchet, M. Abadi, and C. Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005)*, pages 331–340, Chicago, IL, June 2005. IEEE Computer Society.

[58] D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, pages 213–229, London, UK, 2001. Springer-Verlag.

[59] M. Boreale. Symbolic trace analysis of cryptographic protocols. In *28th Colloquium on Automata, Languages and Programming (ICALP)*, LNCS, pages 667–681. Springer-Verlag, 2001.

[60] D. Box, F. Curbera, et al. *Web Services Addressing (WS-Addressing)*, August 2004. At `http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/`.

[61] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*. Springer-Verlag, 2003.

[62] V. Boyko, P. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. *Lecture Notes in Computer Science*, 1807:156–171, 2000.

[63] P. J. Broadfoot and G. Lowe. Analysing a stream authentication protocol using model checking. In *ESORICS '02: Proceedings of the 7th European Symposium on Research in Computer Security*, pages 146–161, London, UK, 2002. Springer-Verlag.

[64] J. A. Bull and D. J. Otway. A nested mutual authentication protocol. *Operating Systems Review*, 33(4):42–47, 1999.

[65] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[66] J. Cederquist and M. T. Dashti. An intruder model for verifying termination in security protocols. Technical report TR-CTIT-05-29, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, Jul 2005.

[67] Y. Chevalier. A simple constraint solving procedure for protocols with exclusive or. In *Proc. 18th Int. Workshop on Unification*, Cork, Ireland, 2004.

[68] Y. Chevalier and L. Vigneron. Automated unbounded verification of security protocols. In *CAV '02: Proceedings of the 14th International Conference on Computer Aided Verification*, pages 324–337, London, UK, 2002. Springer-Verlag.

[69] C. N. Chong. *Experiments in rights control: Expression and Enforcement*. PhD thesis, Telematica Instituut Fundamental Research Series TI/FRS/013, Feb 2005.

[70] C. N. Chong and Et al. Licensescript project. At `http://wwwes.cs.utwente.nl/licensescript/`., 2002-2004.

[71] J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. `http://www.cs.york.ac.uk/ jac/papers/drareview.ps.gz`, 1997.

[72] D. Cohen. *Basic Techniques of Combinatorial Theory*. John Wiley & Sons, 1978.

[73] H. Comon-Lundh and V. Shmatikov. Is it possible to decide whether a cryptographic protocol is secure or not? *Journal of Telecommunications and Information Technology*, 2002.

[74] V. Cortier, S. Delaune, and P. Lafourcade. A survey of algebraic properties used in cryptographic protocols. *Journal of Computer Security*, 2005. To appear.

[75] C. J. F. Cremers, S. Mauw, and E. P. de Vink. Defining authentication in a trace model. In T. Dimitrakos and F. Martinelli, editors, *Fast 2003*, Proceedings of the first international Workshop on Formal Aspects in Security and Trust, pages 131–145, Pisa, September 2003. IITT-CNR technical report.

[76] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Securing SOAP e-services. *International Journal of Information Security*, 1(2):100–115, 2002.

[77] P. R. D'Argenio, H. Hermanns, J. P. Katoen, and J. Klaren. MODEST: A modelling language for stochastic timed systems. In L. de Alfaro and S. Gilmore, editors, *Joint Int. Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM-PROBMIV)*, volume LNCS 2165, pages 87–104, Aachen, Germany, Sep 2001. Springer-Verlag, Berlin. `http://www.springerlink.com/link.asp?id=ka9lrbfnj7g6a9dd`.

[78] S. Delaune. Intruder deduction problem in presence of guessing attacks. In *Proc. Workshop on Security Protocols Verification (SPV'2003), Marseille, France, Sep. 2003*, pages 26–30, 2003.

[79] S. Delaune and F. Jacquemard. A decision procedure for the verification of security protocols with explicit destructors. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS'04)*, pages 278–287, Washington, D.C., USA, October 2004. ACM Press.

[80] S. Delaune and F. Jacquemard. A theory of dictionary attacks and its complexity. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW'04)*, pages 2–15, Asilomar, Pacific Grove, California, USA, June 2004. IEEE Computer Society Press.

[81] R. Delicata and S. Schneider. Temporal rank functions for forward secrecy. In *Proceedings of the 18th IEEE Computer Security Foundations Workshop (CSFW'05)*, pages 126–139, 2005.

[82] G. Delzanno and P. Ganty. Automatic verification of time sensitive cryptographic protocols. In K. Jensen and A. Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems: 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004*, pages 342 – 356, Barcelona, Spain, March 2004. Springer-Verlag, Berlin.

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

[84] W. Diffie, P. C. Van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2(2):107 – 125, June 1992.

[85] D. Dolev and A.C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[86] P. Drielsma and S. Mödersheim. The asw protocol revisited: A unified view. In *Automated Reasoning for Security Protocol Analysis (ARSPA)*, pages 141–156. ENTCS, July 2004.

[87] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. Workshop on Formal Methods and Security Protocols (FMSP'99) (part of FLOC99)*, 1999. http://www.cs.bell-labs.com/who/nch/fmsp99/program.html.

[88] E. English and S. Hamilton. Network security under siege: the timing attack. *IEEE Computer*, 29(3):95–97, Mar 1996. `http://www.computer.org/computer/co1996/r3095abs.htm`.

[89] N. Evans and S. Schneider. Analysing time dependent security properties in CSP using PVS. In F. Cuppens, Y. Deswarte, D. Gollmann, and M. Waidner, editors, *ESORICS*, volume LNCS 1895, pages 222–237. Springer, 2000.

[90] E. W. Felten and M. A. Schneider. Timing attacks on web privacy. In *7th ACM conference on Computer and communications security*, pages 25–32, Athens, Greece, 2000. ACM Press, New York. `http://doi.acm.org/10.1145/352600.352606`.

[91] C. Ferris, D. Langworthy, et al. *Web Services Reliable Messaging Protocol (WS-ReliableMessaging)*, March 2004. At `http://msdn.microsoft.com/ws/2004/03/ws-reliablemessaging/`.

[92] M. Fiore and M. Abadi. Computing symbolic models for verifying cryptographic protocols. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW - 14)*, pages 160–173, 2001.

[93] R. Focardi, R. Gorrieri, R. Lanotte, A. Maggiolo-Schettini, F. Martinelli, S. Tini, and E. Tronci. Formal models of timing attacks on web privacy. In M. Lenisa and M. Miculan, editors, *Electronic Notes in Theoretical Computer Science*, volume 62. Elsevier, 2002.

[94] C. Fournet and M. Abadi. Hiding names: Private authentication in the applied pi calculus. In *Software Security – Theories and Systems. Mext-NSF-JSPS International Symposium (ISSS'02)*, volume 2609 of *Lecture Notes in Computer Science*, pages 317–338. Springer-Verlag Heidelberg, January 2003.

[95] A. Freier, P. Karlton, and P. C. Kocher. The ssl protocol. version 3.0.

[96] A. O. Freier, P. Karlton, and P. C. Kocher. The SSL protocol: Version 3.0. `http://home.netscape.com/eng/ssl3/draft302.txt`, November 1996.

[97] A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In *ASIACRYPT '92: Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques*, pages 244–251, London, UK, 1993. Springer-Verlag.

[98] O. Goldreich. On the foundations of modern cryptography. *Lecture Notes in Computer Science*, 1294:46–74, 1997.

[99] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.

[100] L. Gong. Using one-way functions for authentication. *ACM SIGCOMM Computer Communication Review*, 19(5):8–11, 1989.

[101] L. Gong, T. Mark A. Lomas, R. M.Needham, and J. Saltzer. Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, 1993.

[102] A. D. Gordon and A. S. A. Jeffrey. Types and effects for asymmetric cryptographic protocols. *J. Computer Security*, 12(3/4):435–484, 2004.

[103] A. D. Gordon and R. Pucella. Validating a web service security abstraction by typing. In *ACM Workshop on XML Security 2002*, pages 18–29, 2003. An extended version appears as Microsoft Research Technical Report MSR–TR–2002–108.

[104] R. Gorrieri, E. Locatelli, and F. Martinelli. A simple language for real-time cryptographic protocol analysis. In P. Degano, editor, *12th European Symposium on Programming, ESOP 2003*, volume LNCS 2618, pages 114–128. Springer-Verlag, Berlin, 2003.

[105] R. Gorrieri and F. Martinelli. Process algebraic frameworks for the specification and analysis of cryptographic protocols. In *MFCS*, pages 46–67, 2003.

[106] M. Gudgin. Using WS-Trust and WS-SecureConversation. *MSDN*, May 2004. At `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/ws-trustandsecureconv.asp`.

[107] S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. *ACM Transactions on Information and System Security*, 2(3):25–60, 1999.

[108] D. Harkins and D. Carrel. RFC 2409: The Internet Key Exchange (IKE). `http://www.ietf.org/rfc/rfc2409.txt`, November 1998.

[109] K. Havelund and G. Rosu. Testing linear temporal logic formulae on finite execution traces. Technical Report TR 01-08, RIACS, 2001.

[110] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *Proceedings, 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, July 2000.

[111] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 26(1):100–106, 1983.

[112] A. Huima. Efficient infinite-state analysis of security protocols. In *Proc. FLOC'99 Workshop on Formal Methods and Security Protocols*, 1999.

[113] R. Janvier, Y. Lakhnech, and L. Mazaré. Completing the picture: Soundness of formal encryption in the presence of active adversaries. In *ESOP*, pages 172–185, 2005.

[114] J. E. Johnson, D. E. Langworthy, L. Lamport, and F. H. Vogt. Formal specification of a web services protocol. In *1st International Workshop on Web Services and Formal Methods (WS-FM 2004)*, 2004. University of Pisa.

[115] D. Kähler and R. Küsters. Constraint Solving for Contract-Signing Protocols. In *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR 2005)*, 2005. To appear.

[116] C. Kaler, A. Nadalin, et al. *Web Services Secure Conversation Language (WS-SecureConversation) Version 1.1*, May 2004. At `http://msdn.microsoft.com/ws/2004/04/ws-secure-conversation/`.

[117] C. Kaler, A. Nadalin, et al. *Web Services Trust Language (WS-Trust) Version 1.1*, May 2004. At `http://msdn.microsoft.com/ws/2004/04/ws-trust/`.

[118] J. Katz, R. Ostrovsky, and M. Yung. Forward secrecy in password-only key exchange protocols. In *Security in Communication Networks (SCN)*, volume 2576, pages 29 – 44. Springer-Verlag Heidelberg, January 2003.

[119] E. Kleiner and A. W. Roscoe. Web services security: A preliminary study using Casper and FDR. In *Proceedings of Automated Reasoning for Security Protocol Analysis (ARSPA 04)*, 2004.

[120] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems. In N. Koblitz, editor, *16th Advances in Cryptology (CRYPTO)*, volume LNCS 1109, pages 104–113, Santa Barbara, California, Aug 1996. Springer-Verlag, Berlin. `http://www.cryptography.com/resources/whitepapers/TimingAttacks.pdf`.

[121] S. Kremer, O. Markowitch, and J. Zhou. An intensive survey of non-repudiation protocols. *Computer Communications*, 25(17):1606–1621, November 2002.

[122] S. Kremer and M.D. Ryan. Analysis of an electronic voting protocol in the applied pi-calculus. In Mooly Sagiv, editor, *Programming Languages and Systems — Proceedings of the 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 186–200, Edinburgh, U.K., April 2005. Springer.

[123] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. In J. P. Katoen and P. Stevens, editors, *Proc. 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, volume LNCS 2280, pages 52–66. Springer-Verlag, Berlin, 2002.

[124] L. Lamport. Using time instead of timeout for fault-tolerant distributed systems. *ACM Transactions on Programming Languages and Systems*, 6(2):254–280, April 1984.

[125] P. Laud. Semantics and program analysis of computationally secure information flow. In *Programming Languages and Systems: 10th European Symposium on Programming, ESOP 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genoa, Italy, April 2-6, 2001*, volume 2028 of *LNCS*, pages 77–91. Springer-Verlag, 2001.

[126] P. Laud. Encryption cycles and two views of cryptography. In *NORDSEC 2002 - Proceedings of the 7th Nordic Workshop on Secure IT Systems (Karlstad University Studies 2002:31)*, pages 85–100, 2002.

[127] Y. W. Law. *Key Management and Link-layer Security of Wireless Sensor Networks: Energy-efficient Attack and Defense*. PhD thesis, Telematica Instituut Fundamental Research Series TI/FRS/013, Dec 2005.

[128] Y. W. Law and Et al. Eyes project. At `http://www.eyes.eu.org/`., 2002-2005.

[129] G. Leduc. Verification of two versions of the challenge handshake authentication protocol (CHAP). *Annals of Telecommunications*, 55(1-2):18–30, 2000. Hermes-Lavoisier.

[130] G. Lenzini. *Integration of Analysis Techniques in Security and Fault-Tolerance*. PhD thesis, CTIT Ph.D.-thesis series No. 05-70, Jun 2005.

[131] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, November 1995.

[132] G. Lowe. Some new attacks upon security protocols. In *CSFW: Proceedings of The 9th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1996.

[133] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of 10th IEEE Computer Security Foundations Workshop, 1997*, pages 31–44. IEEE Computer Society Press, 1997.

[134] G. Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6:53–84, 1998.

[135] G. Lowe. Analyzing protocols subject to guessing attacks. *Workshop on Issues in the Theory of Security (WITS'02)*, January 2002.

[136] S. Lucks. Open key exchange: How to defeat dictionary attacks without encrypting public keys. In *Security Protocols, 5th International Workshop*, volume 1361 of *Lecture Notes in Computer Science*, pages 79–90. Springer, April 1997.

[137] T. Mark, A. Lomas, L. Gong, J. H. Saltzer, and R. M. Needham. Reducing risks from poorly chosen keys. *Operating Systems Review*, 23(5):14–18, 1989.

[138] F. Martinelli. Analysis of security protocols as open systems. *Theor. Comput. Sci.*, 290(1):1057–1106, 2003.

[139] F. Martinelli, M. Petrocchi, and A. Vaccarelli. Analysing emms with compositional proof rules for non interference. In R. Gorrieri and R. Lucchi, editors, *IFIP WG 1.7 and ACM SIGPLAN Workshop on Issues in the Theory of Security (WITS)*, pages 62–71, Warsaw, Poland, Apr 2003. Dipartamento di Scienze dell'Informazione Universita di Bologna, Italy.

[140] R. J. McEliece. A public–key cryptosystem based on algebraic coding theory. In *DSN Progress Report 42–44*, pages 114–116. Jet Propulsion Laboratory, Pasadenssa, 1978.

[141] C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[142] C. Meadows. A formal framework and evaluation method for network denial of service. In *CSFW '99: Proceedings of the 1999 IEEE Computer Security Foundations Workshop*, page 4, Washington, DC, USA, 1999. IEEE Computer Society.

[143] C. Meadows. Ordering from satan's menu: a survey of requirements specification for formal analysis of cryptographic protocols. *Sci. Comput. Program.*, 50(1-3):3–22, 2004.

[144] C. Meadows, P. F. Syverson, , and I. Cervesato. Formal specification and analysis of the group domain of interpretation protocol using NPATRL and the NRL protocol analyzer. *Journal of Computer Security*, 12(6):893–931, 2004.

[145] D. Micciancio and B. Warinschi. Completeness theorems for the abadi-rogaway language of encrypted expressions. *Journal of Computer Security*, To appear.

[146] Microsoft Corporation. *Web Services Enhancements (WSE) 2.0 SP1*, July 2004. At `http://msdn.microsoft.com/webservices/building/wse/default.aspx`.

[147] J. Millen. Constraint solver webpage, at `http://www.csl.sri.com/users/millen/capsl/constraints.html`.

[148] J. Millen. On the freedom of decryption. *Information Processing Letters*, 86(6):329–333, June 2003.

[149] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *8th ACM Conference on Computer and Communication Security*, pages 166–175. ACM SIGSAC, November 2001.

[150] J. Millen and V. Shmatikov. Symbolic protocol analysis with an abelian group operator or diffie-hellman exponentiation. *Journal of Computer Security, special issue on selected papers of CSFW-16 (ed. R. Foccardi)*, 2004.

[151] D. Mills. Cryptographic authentication for real-time network protocols. *AMS DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 45:135–144, 1999.

[152] R. Milner. *Communication and concurrency*. Prentice Hall International (UK) Ltd., Hertfordshire, UK, 1995.

[153] A. Nadalin, C. Kaler, P. Hallam-Baker, and R. Monzillo. *OASIS Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)*, March 2004. At `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf`.

[154] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

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

[156] A. Perrig, J. D. Tygar, D. Song, and R. Canetti. Efficient authentication and signing of multicast streams over lossy channels. In *SP '00: Proceedings of the 2000 IEEE Symposium on Security and Privacy*, page 56, Washington, DC, USA, 2000. IEEE Computer Society.

[157] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is np-complete. In S. Schneider, editor, *Proc. 14th IEEE Computer Security Foundations Workshop*, 2001.

[158] V. Shmatikov. Protocol analysis tools for CS 395T course. At `http://www.cs.utexas.edu/~shmat/courses/cs395t_fall04/cs395t_tools.html`.

[159] D. X. Song, S. Berezin, and A. Perrig. Athena: A novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1/2):47–74, 2001.

[160] P. Syverson. A taxonomy of replay attacks. In *Proceedings of the Computer Security Foundations Workshop (CSFW97)*, pages 187–191, June 1994.

[161] P. Syverson and C. Meadows. A formal language for cryptographic protocol requirements. *Designs, Codes and Cryptography*, 7:27 – 59, 1996.

[162] F. J. Thayer Fábrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7:191–230, 1999.

[163] W3C. *SOAP Version 1.2*, 2003. W3C Recommendation, at `http://www.w3.org/TR/soap12`.

[164] T. Y. C. Woo and S. S. Lam. A semantic model for authentication protocols. In *IEEE Computer Society Symposium on Research in Security and Privacy*, pages 178–194, 1993.

[165] T. Y. C. Woo and S. S. Lam. A lesson on authenticated protocol design. *Operating Systems Review*, 28(3):24–37, 1994.

[166] WS-SecureConversation/WS-Trust Interop Workshop, October 2004. At `http://msdn.microsoft.com/webservices/community/workshops/TrustWorkshopOct2004.aspx`.

[167] S. Yovine. KRONOS: a verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 1(1/2):123–133, 1997.

[168] J. Zhou and D. Gollmann. An efficient non-repudiation protocol. In *Proceedings of the 10th Computer Security Foundations Workshop (CSFW '97)*, pages 126–132, 1997.

APPENDIX A

Proofs

A.1 From Chapter 2

From Section 2.3

Definition A.1.1. *Let r be a protocol role, and let s be a suffix of r. Then we let $r - s$ denote the prefix protocol role of r up to s, i.e. $r - s$ is a role t s.t. $r = \langle t\ s \rangle$. Also, given two system scenarios $R = \{r_1, \ldots, r_n\}$ and $S = \{s_1, \ldots, s_n\}$ s.t. s_i is a suffix of r_i for $i \in [1 \ldots n]$, we let $R - S$ denote the system scenario $\{r_1 - s_1, \ldots, r_n - s_n\}$.*

Lemma A.1.2. *Let Sc_0 be a system scenario and let IK be an initial intruder knowledge. In Procedure 2.3.4, let $\langle Sc, IK, CS, tr \rangle$ be a state in a run for Sc_0 with initial intruder knowledge IK. Then there exists σ s.t. tr is an interleaving of $Sc_0\sigma - Sc$.*

Proof. We proceed by induction on the number of steps in the run for Sc_0. For the base case, we have $\langle Sc, IK, CS, tr \rangle$ equal to the initial state $\langle Sc_0, IK, \emptyset, \langle\rangle \rangle$, and thus $tr = \langle\rangle$. Let $\sigma = \varepsilon$ (recall that ε denotes the empty substitution). Then $tr = \langle\rangle$ is an interleaving of $Sc_0\sigma - Sc = Sc_0 - Sc_0 = \emptyset$.

Now consider a state $\langle Sc_n, IK, CS_n, tr_n \rangle$, with $\langle Sc_{n-1}, IK, CS_{n-1}, tr_{n-1} \rangle$ the previous step in Procedure 2.3.4. We know that $Sc_n = (Sc_{n-1} \setminus \{r\} \cup \{r'\})\gamma$ for some $r \in Sc_{n-1}$ with $r = \langle ev\ r' \rangle$. We consider two cases depending on whether ev is a send or recieve event.

- When $ev = \langle a : m \triangleright b \rangle$, Procedure 2.3.4 dictates that $\gamma = \emptyset$, $CS_n = CS_{n-1}$ and $tr_n = \langle\ tr_{n-1}\ ev\ \rangle$. By the inductive hypothesis, there exists σ s.t. tr_{n-1} is an interleaving of $Sc_0\sigma - Sc_{n-1}$. Hence, $tr_n = \langle\ tr_{n-1}\ ev \rangle$ is an interleaving of $Sc_0\sigma - (Sc_{n-1} \setminus \{\langle ev\ r' \rangle\} \cup \{r'\}) = Sc_0\sigma - Sc_n$.

- When $ev = \langle a : m \triangleleft b \rangle$, then Procedure 2.3.4 dictates that (CS_n, γ) is the output of running **P** on $CS_{n-1} \cup \{m : K(tr) \cup IK\}$, and $tr_n = \langle\ tr_{n-1}\gamma\ ev\gamma \rangle$.

  In this case, by the inductive hypothesis, there exist σ' s.t. tr_{n-1} is an interleaving of $Sc_0\sigma' - Sc_{n-1}$. Let $\sigma = \sigma'\gamma$. Then, $tr_n = \langle\ tr_{n-1}\gamma\ ev\gamma \rangle = \langle tr_{n-1}\ ev \rangle\gamma$ is an interleaving of $(Sc_0\sigma' - (Sc_{n-1} \setminus \{\langle ev\ r' \rangle\} \cup \{r'\}))\gamma = (Sc_0\sigma'\gamma - (Sc_{n-1} \setminus \{\langle ev\ r' \rangle\} \cup \{r'\})\gamma) = Sc_0\sigma'\gamma - Sc_n = Sc_0\sigma - Sc_n$.

□

Remark A.1.3. *If tr is an interleaving of $R - S$, then tr is a prefix interleaving of R.*

Corollary A.1.4. *Let Sc_0 be a system scenario and let IK be an initial intruder knowledge. In Procedure 2.3.4, let $\langle Sc, IK, CS, tr\rangle$ be a state in a run for Sc_0 with initial intruder knowledge IK. Then tr is derived from Sc_0.*

Proof. Recall that tr is derived from Sc_0 if tr is an instance of a prefix interleaving of Sc_0 (see Section 2.3).

By Lemma A.1.2, tr is an interleaving of $Sc_0\sigma - Sc$ for some σ. By Remark A.1.3, tr is a prefix interleaving of $Sc_0\sigma$. Hence tr is an instance of a prefix interleaving of Sc_0, obtaining the claim. □

Theorem 2.3.6. *Let Sc_0 be a system scenario and let IK be an initial intruder knowledge. For Procedure 2.3.4, it holds:*

1. *Soundness: Let $\langle Sc, IK, CS, tr\rangle$ be a state in a run for Sc_0 with initial intruder knowledge IK. Then for every solution σ of CS, (i) $tr\sigma$ is derived from Sc_0 and (ii) $tr\sigma$ is valid wrt IK.*

2. *Completeness: Let tr be a valid trace wrt IK derived from Sc_0. Then there exists a state $\langle Sc, IK, CS, tr'\rangle$ in a run for Sc_0 wrt IK and a substitution σ s.t. σ is a solution of CS and $tr = tr'\sigma$.*

Proof. (1). (i) follows from Corollary A.1.4, since $tr\sigma$ is an instance of tr. (ii) Recall that $tr\sigma$ is valid if for each $i \in [0 \dots length(tr) - 1]$, $last(tr_{i+1}) = \langle a : m \triangleleft b\rangle$ implies that $m \in \mathcal{F}(K(tr_i) \cup IK)$ (see Definition 2.3.5).

Now, let σ be a solution of CS. We proceed by induction on the number of steps in the run for Sc_0. For the base case, we have $\langle Sc, IK, CS, tr\rangle$ equal to the initial state $\langle Sc_0, IK, \emptyset, \langle\rangle\rangle$. (ii) holds since trace $tr\sigma = \langle\rangle\sigma = \langle\rangle$ is trivially valid.

Now consider a state $\langle Sc_n, IK, CS_n, tr_n\rangle$, with $\langle Sc_{n-1}, IK, CS_{n-1}, tr_{n-1}\rangle$ being the previous step in Procedure 2.3.4. We know that $Sc_n = (Sc_{n-1} \setminus \{r\} \cup \{r'\})\gamma$ for some $r \in Sc_{n-1}$ with $r = \langle ev\ r'\rangle$. We have to consider two cases for ev, as dictated by Procedure 2.3.4:

- Case $ev = \langle a : m \triangleright b\rangle$. Then $CS_n = CS_{n-1}$, $\gamma = \emptyset$ and $tr_n = \langle tr_{n-1}\ ev\rangle$. By the inductive hypothesis, $tr_{n-1}\sigma$ is valid wrt IK.Then $tr_n\sigma$ is also valid wrt IK since ev is a send communication event, and Definition 2.3.5 only regards receive event.

- Case $ev = \langle a : m \triangleleft b\rangle$. Then (CS_n, γ) is the output of running $\mathbf{P}$ on $CS_{n-1} \cup \{m : K(tr) \cup IK\}$, and $tr_n = \langle tr_{n-1}\gamma\ ev\gamma\rangle$. Recall that σ is a solution of CS_n. Then by soundness in Theorem 2.2.15, σ is also a solution of $(CS_{n-1} \cup \{m : K(tr) \cup IK\})\gamma$, and so $\gamma\sigma$ is a solution of CS_{n-1}. By inductive hypothesis, $tr_{n-1}\gamma\sigma$ is valid wrt IK. Since $\gamma\sigma$ is solution of $m : K(tr) \cup IK$, $m\gamma\sigma \in \mathcal{F}(K(tr)\gamma\sigma \cup IK)$, and then $ev\gamma\sigma$ is valid. Thus, $tr_n\sigma = \langle tr_{n-1}\gamma\ ev\gamma\rangle\sigma = \langle tr_{n-1}\gamma\sigma\ ev\gamma\sigma\rangle$ is valid.

(2). We proceed by induction on the length of tr. If $tr = \langle\rangle$, then the initial state $\langle Sc_0, IK, \emptyset, \langle\rangle\rangle$ is our desired state. Now let tr be a trace with n events recorded. Since tr is derived from Sc_0, we know that there is ρ s.t. $tr = tr'\rho$, with tr' prefix interleaving of Sc_0. Now let tr'' and ev be a trace and an event respectively s.t. $tr' = \langle tr''\ ev\rangle$. Since $tr = tr'\rho$ is valid wrt IK and derived from Sc_0, then also $tr''\rho$ is valid wrt IK and derived from Sc_0. Moreover, $tr''\rho$ has $n - 1$ events recorded. Thus by inductive hypothesis there must exist a state

$\langle Sc'', IK, CS'', tr^0\rangle$ and a substitution σ s.t. σ is a solution of CS'' and $tr''\rho = tr^0\sigma$. Since tr^0 is derived from Sc_0 by Corollary A.1.4, then $tr^0 = tr''\gamma$ for some γ and hence $\rho = \gamma\sigma$. Consider cases for ev:

- Case $ev = \langle a : m \triangleright b\rangle$. From state $\langle Sc'', IK, CS'', tr''\gamma\rangle$, Procedure 2.3.4 dictates that there must exist a subsequent state $\langle Sc', IK, CS'', \langle tr''\ ev\rangle\gamma\rangle$, where $Sc' = Sc'' \setminus \{\langle ev\gamma\ r\rangle\} \cup \{r\}$ for some $r \in Sc''$. Then $tr = tr'\rho = \langle tr''\ ev\rangle\rho = \langle tr''\rho\ ev\rho\rangle = \langle tr''\gamma\sigma\ ev\gamma\sigma\rangle = \langle tr''\gamma\ ev\gamma\rangle\sigma$, establishing the claim.

- Case $ev = \langle a : m \triangleleft b\rangle$. Again, from state $\langle Sc'', IK, CS'', tr''\gamma\rangle$, there exists a subsequent state $\langle Sc'\gamma', IK, CS', \langle tr''\ ev\rangle\gamma\gamma'\rangle$ with (CS', γ') output of **P** applied to $CS'' \cup \{m\gamma : K(tr'')\gamma \cup IK\}$, where $Sc' = Sc'' \setminus \{\langle ev\gamma\ r\rangle\} \cup \{r\}$ for some $r \in Sc''$. Since σ is a solution of CS'' and $ev\rho = ev\gamma\sigma$ is valid by assumption, then σ is a solution of $CS'' \cup \{m\gamma : K(tr'')\gamma \cup IK\}$. By completeness in Theorem 2.2.15, then for some σ' solution of CS', $\sigma = \gamma'\sigma'$. Then $tr = \langle tr''\ ev\rangle\rho = \langle tr''\gamma\sigma\ ev\gamma\sigma\rangle = \langle tr''\gamma\gamma'\ ev\gamma\gamma'\rangle\sigma'$, obtaining the claim.

□

From Section 2.4

Notation and terminology Recall that $C_V = \{c_X \mid X \in \mathcal{V}\}$ is a set of fresh constants generated by the intruder, and $\sigma_V = \{X \to c_X \mid X \in \mathcal{V}\}$. To establish Theorem 2.4.4, we first clone C_V and define a set of fresh constants $D_V = \{d_X \mid X \in \mathcal{V}\}$, and let $\rho_V = \{X \to d_X \mid X \in \mathcal{V}\}$.

We say that t *occurs* in s, written $t \preceq s$, if t is a subterm of s. We extend this to sets, and write $t \preceq S$ if $t \preceq s$ for some $s \in S$. We also write $S \preceq t$ if for some $s \in S$, $s \preceq t$. We write $S \preceq T$ if for some $s \in S$ and $t \in T$, $s \preceq t$. We write $t \not\preceq s$ when $t \preceq s$ does not hold (and similarly for $t \not\preceq S$, $S \not\preceq t$ and $S \not\preceq T$).

Finally, we extend substitutions and use *extended substitutions*, i.e. mappings that replace occurrences of constants by constants. We use, in particular, the extended substitution δ, that for each $X \in \mathcal{V}$ maps constant d_X into constant c_X, i.e. $\delta = \{d_X \to c_X \mid X \in \mathcal{V}\}$.

Lemma A.1.5. *Let t be a term s.t. $D_V \not\preceq t$. Then $t\sigma_V = t\rho_V\delta$.*

Proof. By (straightforward) structural induction on t. For the base case, $t = c$ where $c \neq d_X$. Then $c\sigma_V = c = c\rho_V\delta$. For the inductive case, let $t = (t_1, t_2)$. Since d_X does not occur in t_1 nor in t_2, then by inductive hypothesis, $t_1\sigma_V = t_1\rho_V\delta$ and $t_2\sigma_V = t_2\rho_V\delta$, and hence $t\sigma_V = (t_1, t_2)\sigma_V = (t_1\sigma_V, t_2\sigma_V) = (t_1\rho_V\delta, t_2\rho_V\delta) = (t_1, t_2)\rho_V\delta = t\rho_V\delta$. The remaining cases are similar. □

Corollary A.1.6. *Let T be a set of terms s.t. $D_V \not\preceq T$. Then $\mathcal{F}(T\rho_V\delta) = \mathcal{F}(T\sigma_V)$.*

The next step consists in extending the $\mathcal{F}(\cdot)$ function with a new decryption rule $cdec$, that allows symmetric decryption of $\{t_1\}_{t_2}$ using a key t_2' whenever t_2 and t_2' are equal up to δ, i.e. $t_2\delta = t_2'\delta$:

$$\{\{t_1\}_{t_2}, t_2'\} \to_{cdec} t_1 \text{ when } t_2\delta = t_2'\delta$$

Intuitively, for each variable X we use constants c_X and d_X, which represent the *same* intruder generated constant; still, by technical reasons we want to distinguish the usage of the constants coming from C_V and the instantiations given to the variables.

We define $\mathcal{F}_\delta(T)$ to be as $\mathcal{F}(T)$ plus the new rule $cdec$. More formally:

Definition A.1.7. *Let T be a ground term set and δ the extended substitution described above. Then, $\mathcal{F}_\delta(T)$ is defined as $\cup_{n\geq 0}\mathcal{F}^n_\delta(T)$, where $\mathcal{F}^n_\delta(T)$ is the set defined inductively as follows:*

$$\begin{aligned}\mathcal{F}^0_\delta(T) &= T\\ \mathcal{F}^n_\delta(T) &= \mathcal{F}^{n-1}_\delta(T) \cup \{t \mid A \to_\ell t \text{ is a DY rule or } cdec \text{ and } A \subseteq \mathcal{F}^{n-1}_\delta(T)\}\end{aligned}$$

Lemma A.1.8. $\mathcal{F}_\delta(T\rho_V)\delta = \mathcal{F}(T\rho_V\delta)$.

Proof. ($\subseteq$) By definition of $\mathcal{F}(\cdot)$ and $\mathcal{F}_\delta(\cdot)$, (Definition 2.2.1 and A.1.7), $\mathcal{F}_\delta(T\rho_V)\delta = \cup_{n\geq 0}\mathcal{F}^n_\delta(T\rho_V)\delta$ and $\mathcal{F}(T\rho_V\delta) = \cup_{n\geq 0}\mathcal{F}^n(T\rho_V\delta)$.

We show by induction on n that $\mathcal{F}^n_\delta(T\rho_V)\delta \subseteq \mathcal{F}^n(T\rho_V\delta)$. For the base case, we have that $\mathcal{F}^0_\delta(T\rho_V)\delta = T\rho_V\delta = \mathcal{F}^0(T\rho_V\delta)$.

Now consider the inductive case, and let $t \in \mathcal{F}^n_\delta(T\rho_V)\delta = \mathcal{F}^{n-1}_\delta(T\rho_V)\delta \cup \{r \mid A \to_\ell r$ is a DY rule or $cdec$ and $A \subseteq \mathcal{F}^{n-1}_\delta(T\rho_V)\}\delta$. If $t \in \mathcal{F}^{n-1}_\delta(T\rho_V)\delta$, by inductive hypothesis we are done, so consider the case in which $t \in \{r \mid A \to_\ell r$ is a DY rule or $cdec$ and $A \subseteq \mathcal{F}^{n-1}_\delta(T\rho_V)\}\delta$. So there is a rule ℓ s.t. ℓ is a DY rule or $\ell = cdec$ s.t. $A \to_\ell r$ with $t = r\delta$ and $A \subseteq \mathcal{F}^{n-1}_\delta(T\rho_V)$. Consider cases for ℓ:

- $\ell = pair$. Then $r = (t_1, t_2)$, and $\{t_1, t_2\} \subseteq \mathcal{F}^{n-1}_\delta(T\rho_V)$, hence $\{t_1\delta, t_2\delta\} \subseteq \mathcal{F}^{n-1}_\delta(T\rho_V)\delta$. By inductive hypothesis $\{t_1\delta, t_2\delta\} \subseteq \mathcal{F}^{n-1}(T\rho_V\delta)$ and hence $t = r\delta = (t_1, t_2)\delta = (t_1\delta, t_2\delta) \in \mathcal{F}^n(T\rho_V\delta)$ by applying rule $pair$ in $\mathcal{F}^{n-1}(T\rho_V\delta)$. (The remaining cases (except $\ell = cdec$, done below) are similar.)

- $\ell = cdec$. Then $A = \{\{r\}_{t_1}, t'_1\}$ for some r s.t. $t = r\delta$ and some t_1 and t'_1 s.t. $t_1\delta = t'_1\delta$. By inductive hypothesis $\{\{r\delta\}_{t_1\delta}, t'_1\delta\} \subseteq \mathcal{F}^{n-1}(T\rho_V\delta)$, and since $t_1\delta = t'_1\delta$ then $t = r\delta \in \mathcal{F}^n(T\rho_V\delta)$ applying rule $sdec$.

Case ($\supseteq$). We show by induction on n that $\mathcal{F}^n_\delta(T\rho_V)\delta \supseteq \mathcal{F}^n(T\rho_V\delta)$. The base case is similar to the previous case ($\subseteq$).

Now consider the inductive case, and let $t \in \mathcal{F}^n(T\rho_V\delta) = \mathcal{F}^{n-1}(T\rho_V\delta) \cup \{r \mid A \to_\ell r$ is a DY rule and $A \subseteq \mathcal{F}^{n-1}(T\rho_V\delta)\}$. If $t \in \mathcal{F}^{n-1}(T\rho_V\delta)$, by inductive hypothesis we are done, so consider the case in which $t \in \{r \mid A \to_\ell r$ is a DY rule and $A \subseteq \mathcal{F}^{n-1}(T\rho_V\delta)\}$. So there is a rule ℓ s.t. ℓ is a DY rule s.t. $A \to_\ell r$ and $A \subseteq \mathcal{F}^{n-1}(T\rho_V\delta)$. Consider cases for ℓ:

- $\ell = pair$. Then $t = (t_1, t_2)$, with $A = \{t_1, t_2\} \subseteq \mathcal{F}^{n-1}(T\rho_V\delta)$, so by inductive hypothesis $\{t_1, t_2\} \subseteq \mathcal{F}^{n-1}_\delta(T\rho_V)\delta$. So there is $\{r_1, r_2\} \subseteq \mathcal{F}^{n-1}_\delta(T\rho_V)$ s.t. $t_1 = r_1\delta$ and $t_2 = r_2\delta$. Hence, $(r_1, r_2) \in \mathcal{F}^n_\delta(T\rho_V)$ applying rule $pair$, which means that $t = (t_1, t_2) = (r_1\delta, r_2\delta) = (r_1, r_2)\delta \in \mathcal{F}^n_\delta(T\rho_V)\delta$. (The remaining cases (except $\ell = sdec$, done below) are similar.)

- $\ell = sdec$. Then $A = \{\{t\}_{t_1}, t_1\} \subseteq \mathcal{F}^{n-1}(T\rho_V\delta)$, so by inductive hypothesis $\{\{t\}_{t_1}, t_1\} \subseteq \mathcal{F}^{n-1}_\delta(T\rho_V)\delta$. So there is $\{\{r\}_{r_1}, r'_1\} \subseteq \mathcal{F}^{n-1}_\delta(T\rho_V)$ s.t. $\{t\}_{t_1} = \{r\}_{r_1}\delta$ and $t_1 = r'_1\delta$. Now, since $r_1\delta = t_1 = r'_1\delta$, we can apply rule $cdec$ in $\{\{r\}_{r_1}, r'_1\} \subseteq \mathcal{F}^{n-1}_\delta(T\rho_V)$ to obtain $r \in \mathcal{F}^n_\delta(T\rho_V)$, hence $t = r\delta \in \mathcal{F}^n_\delta(T\rho_V)\delta$.

□

Lemma A.1.9. *Let T' be a term set in which $(D_V \cup C_V) \not\preceq T'$, and let $T = T' \cup C_V$. If $\{t\}_r \in \mathcal{F}(T)$ and for some $X_0 \in \mathcal{V}$, $c_{X_0} \preceq t$ or $c_{X_0} \preceq r$, then $\{t, r\} \subseteq \mathcal{F}(T)$. The same holds for the terms (t, r),$h(t)$, $pk(t)$, $\{t\}_r^{\rightarrow}$ and $sig_t(r)$.*

Proof. We show by induction on n that if $\{t\}_r \in \mathcal{F}^n(T)$ and c_{X_0} occurs in either t or r for some $X_0 \in \mathcal{V}$, then $\{t, r\} \subseteq \mathcal{F}^n(T)$. The base case holds trivially, since by assumption there is no term $\{t\}_r \in T = T' \cup C_V$ s.t. c_{X_0} occurs in t or r for some $X_0 \in \mathcal{V}$. For the inductive step, let $\{t\}_r \in \mathcal{F}^n(T) = \mathcal{F}^{n-1}(T) \cup \{s \mid A \rightarrow_\ell s \text{ is a DY rule and } A \subseteq \mathcal{F}^{n-1}(T)\}$. If $\{t\}_r \in \mathcal{F}^{n-1}(T)$ we apply the inductive hypothesis and we are done, so consider the case in which there is some DY rule ℓ s.t. $A \rightarrow_\ell \{t\}_r$ with $A \subseteq \mathcal{F}^{n-1}(T)$. Take cases for ℓ:

- $\ell = senc$: Then $A = \{t, r\} \subseteq \mathcal{F}^{n-1}(T) \subseteq \mathcal{F}^n(T)$.
- $\ell = sdec$: $A = \{\{\{t\}_r\}_u, u\}$. Since c_{X_0} occurs in either t or r, then c_{X_0} occurs in $\{\{t\}_r\}_u$, and hence by inductive hypothesis $\{t\}_r \in \mathcal{F}^{n-1}(T)$. Applying the inductive hypothesis again gives us the claim. (The cases of $\ell = first$ and $\ell = second$ are similar to this case).

□

Lemma A.1.10. *Let T' be a term set in which $(D_V \cup C_V) \not\preceq T'$, and let $T = T' \cup C_V$. Let σ be a ground substitution s.t. for each $X \in \mathcal{V}$ s.t. $X \preceq T$, $X\sigma \in \mathcal{F}(T\sigma)$ and $(C_V \cup D_V) \not\preceq X\sigma$. Let σ' be the extended substitution $\{d_X \rightarrow X\sigma \mid X \in \mathcal{V}\}$ and $\delta = \{d_X \rightarrow c_X \mid X \in \mathcal{V}\}$. Let t be a term s.t. $t\sigma' \in \mathcal{F}(T\sigma)$, and let r be a ground term for which:*

1. *$C_V \not\preceq r$,*
2. *for every d_Y with $Y \in \mathcal{V}$ s.t. $d_Y \preceq r$, $Y \preceq T'$, and*
3. *$t\delta = r\delta$.*

Then $r\sigma' \in \mathcal{F}(T\sigma)$.

Proof. We proceed by structural induction on r. The base case is when r is a constant:

- $r = d_{X_0}$ for some $X_0 \in \mathcal{V}$. By (2), X_0 occurs in T', and hence $r\sigma' = X_0\sigma \in \mathcal{F}(T\sigma)$ by hypothesis.
- Otherwise, $r = c \neq d_X$ (and also $c \neq c_X$ by assumption (1)) for each $X \in \mathcal{V}$. Here, $r\delta = c\delta = c = t\delta$ (by assumption (3)) and hence $t\sigma' = c\sigma' = c$, thus $r\sigma' = c\sigma' = c = t\sigma' \in \mathcal{F}(T\sigma)$.

For the inductive case, we take cases on the structure of r (we show here only the case for pair, but the remaining cases are similar):

- $r = (u, v)$. We know that $t = (w, x)$ for some w and x. Thus, $r\delta = (u, v)\delta = (u\delta, v\delta) = (w\delta, x\delta) = (w, x)\delta = t\delta$. So $u\delta = w\delta$ and $v\delta = x\delta$. Take cases on whether c_{X_0} for some $X_0 \in \mathcal{V}$ occurs in w or x:

– Suppose that c_{X_0} does not occur in w nor x for all $X_0 \in \mathcal{V}$. Then since $r\delta = t\delta$ and no c_X occurs in r for any $X \in \mathcal{V}$ by assumption (1), then $r = t$ and we are done.

– Now assume that c_{X_0} occurs in either w or x, for some $X_0 \in \mathcal{V}$. Then c_{X_0} occurs in $w\sigma'$ or in $x\sigma'$, since σ' only replaces occurrences of d_Y for every $Y \in \mathcal{V}$. Since $(w, x)\sigma' \in \mathcal{F}(T\sigma)$, then by Lemma A.1.9 $\{w\sigma', x\sigma'\} \subseteq \mathcal{F}(T\sigma)$. By inductive hypothesis, then $\{u\sigma', v\sigma'\} \subseteq \mathcal{F}(T\sigma)$, and so $r\sigma' \in \mathcal{F}(T\sigma)$.

□

Lemma A.1.11. *Let T' be a term set in which $(D_V \cup C_V) \not\preceq T'$, and let $T = T' \cup C_V$. Let σ be a substitution s.t. for each variable $X \in \mathcal{V}$ s.t. $X \preceq T$, $X\sigma \in \mathcal{F}(T\sigma)$ and$(C_V \cup D_V) \not\preceq X\sigma$. Also let σ' be the extended substitution $\{d_X \rightarrow X\sigma \mid X \in \mathcal{V}\}$. Then $\mathcal{F}_\delta(T\rho_V)\sigma' \subseteq \mathcal{F}(T\sigma)$.*

Proof. We show by induction on n that $\mathcal{F}^n_\delta(T\rho_V)\sigma' \subseteq \mathcal{F}(T\sigma)$. The base case: $\mathcal{F}^0_\delta(T\rho_V)\sigma' = T\rho_V\sigma' = T\sigma \subseteq \mathcal{F}(T\sigma)$, the last equality following from the fact that there are no occurrences of d_X in T for any $X \in \mathcal{V}$.

For the inductive case, let $t \in \mathcal{F}^n_\delta(T\rho_V)\sigma' = \mathcal{F}^{n-1}_\delta(T\rho_V)\sigma' \cup \{r \mid A \rightarrow_\ell r$ is a DY rule or $cdec$ and $A \subseteq \mathcal{F}^{n-1}_\delta(T\rho_V)\}\sigma'$. If $t \in \mathcal{F}^{n-1}_\delta(T\rho_V)\sigma'$, by inductive hypothesis we are done, so consider the case in which $t \in \{r \mid A \rightarrow_\ell r$ is a DY rule or $cdec$ and $A \subseteq \mathcal{F}^{n-1}_\delta(T\rho_V)\}\sigma'$. So there is a rule ℓ s.t. ℓ is a DY rule or $\ell = cdec$ s.t. $A \rightarrow_\ell r$ with $t = r\sigma'$ and $A \subseteq \mathcal{F}^{n-1}_\delta(T\rho_V)$. Consider cases for ℓ:

- $\ell = pair$. Then $r = (t_1, t_2)$, and $\{t_1, t_2\} \subseteq \mathcal{F}^{n-1}_\delta(T\rho_V)$, hence $\{t_1\sigma', t_2\sigma'\} \subseteq \mathcal{F}^{n-1}_\delta(T\rho_V)\sigma'$. By inductive hypothesis $\{t_1\sigma', t_2\sigma'\} \subseteq \mathcal{F}(T\sigma)$ and hence $t = r\sigma' = (t_1, t_2)\sigma' = (t_1\sigma', t_2\sigma') \in \mathcal{F}(T\sigma)$ by applying rule $pair$ in $\mathcal{F}(T\sigma)$. (The remaining cases (except $\ell = cdec$, done below) are similar.)

- $\ell = cdec$. Then $A = \{\{r\}_{t_1}, t'_1\} \subseteq \mathcal{F}^{n-1}_\delta(T\rho_V)$, with $t_1\delta = t'_1\delta$. Hence, it holds that $\{\{r\}_{t_1}\sigma', t'_1\sigma'\} \subseteq \mathcal{F}^{n-1}_\delta(T\rho_V)\sigma'$. By inductive hypothesis, $\{\{r\}_{t_1}\sigma', t'_1\sigma'\} \subseteq \mathcal{F}(T\sigma)$.

  Take cases in which c_{X_0} for some $X_0 \in \mathcal{V}$ occurs in either $r\sigma'$ or $t_1\sigma'$:

  – Consider the case in which c_{X_0} for some $X_0 \in \mathcal{V}$ occurs in either $r\sigma'$ or $t_1\sigma'$.

    We can apply Lemma A.1.9, since $T\sigma = T'\sigma \cup C_V$ where no c_Y nor d_Z occurring in $T'\sigma$ for any $Y, Z \in \mathcal{V}$, since by assumption no c_Y and d_Z occur in $X\sigma$ for all $X, Y, Z \in \mathcal{V}$ s.t. X occurs in T').

    Thus since $\{r\}_{t_1}\sigma' \in \mathcal{F}(T\sigma)$, by Lemma A.1.9, $\{t_1\sigma', r\sigma'\} \subseteq \mathcal{F}(T\sigma)$, and since $t = r\sigma'$ we are done.

  – If no c_X occurs in $r\sigma'$ nor in $t_1\sigma'$ for all $X \in \mathcal{V}$. Then no c_X occurs in t_1 either, for all $X \in \mathcal{V}$. Then we can apply Lemma A.1.10 and obtain that $t_1\sigma' \in \mathcal{F}(T\sigma)$, and hence $t = r\sigma' \in \mathcal{F}(T\sigma)$.

□

Lemma A.1.12. *Let m be a ground term s.t. $(C_V \cup D_V) \not\preceq m$. Let σ' be the extended substitution $\{d_X \rightarrow X\sigma \mid X \in dom(\sigma)\}$, given a substitution σ, and let δ be the extended substitution $\{d_X \rightarrow c_X \mid X \in \mathcal{V}\}$. Let T be a ground term set. Then $m \notin \mathcal{F}(T)\sigma'$ implies that $m \notin \mathcal{F}(T)\delta$.*

Proof. We proceed by structural induction on m. For $m = c$ for some constant c s.t. $C_V \cup D_V \not\preceq c$, $c \notin \mathcal{F}(T)\sigma'$ implies that $c \notin \mathcal{F}(T)$ since $D_V \not\preceq c$. Hence $c \notin \mathcal{F}(T)\delta$ since $C_V \not\preceq c$. Now consider cases for m ((We show here only the case for $pair$, the other cases are similar):

- $m = (t_1, t_2)$. Hence $t_1 \notin \mathcal{F}(T)\sigma'$ and $t_2 \notin \mathcal{F}(T)\sigma'$, so by inductive hypothesis $\{t_1, t_2\} \notin \mathcal{F}(T)\delta$. Hence $m = (t_1, t_2) \notin \mathcal{F}(T)\delta$.

□

We now quote two needed results from Millen and Shmatikov's paper [149]:

Theorem A.1.13. *Let $\langle Sc, IK, CS, tr\rangle$ be a state of Procedure 2.3.4, for initial scenario Sc_0, and let $\neg(m : K(tr') \cup IK)$ be a negated constraint where tr' is a prefix trace of tr. Recall also that S satisfies the origination assumption (see Section 2.2.2). Then:*

1. **Theorem (Invariance of origination)** *in [149]: For each variable $X \preceq K(tr') \cup IK$, there is $(X : K(tr'') \cup IK) \in CS$ with tr'' prefix trace of tr', and $X \not\preceq K(tr'')$.*
2. **Theorem (Invariance of monotonicity)** *in [149]: Let tr' and tr'' be the two traces mentioned above, satisfying that tr'' is a prefix trace of tr'. Let σ be a substitution s.t. $X\sigma \in \mathcal{F}((K(tr'') \cup IK)\sigma)$ for each $X \in \mathcal{V}$ s.t. $X : T_X \in CS$ for some T_X. Then $X\sigma \in \mathcal{F}((K(tr') \cup IK)\sigma)$.*

Theorem A.1.13(1) says that whenever we have a constraint mentioning a variable X, we can find another constraint in CS of the form $X : T$, where X does not occur in T. So, variable X always originates in a receive action. Theorem A.1.13(2) says that the intruder never forgets information: If one solution is possible in an early step of the run, then that solution is still possible afterwards (this is evidenced by the fact that since tr'' is a prefix trace of tr', then $K(tr'') \subseteq K(tr')$).

We are ready now to prove the main result, Theorem 2.4.4.

Theorem 2.4.4. *Let $\langle Sc, IK, CS, tr\rangle$ be a state from Procedure 2.3.4 where for each $X : T_X \in CS$ for $X \in \mathcal{V}$, $T_X = T'_X \cup C_V$ where $C_V \not\preceq T'_X$. Let $\neg(m : K(tr') \cup IK)$ be a negated constraint for tr' prefix trace of tr,* where m is ground and $C_V \not\preceq m$. *Let σ be a substitution s.t. for all $X \in \mathcal{V}$ with $X : T_X \in CS$, $C_V \not\preceq X\sigma$. Then σ is a solution of both CS and $\neg(m : K(tr') \cup IK)$ iff σ_V is a solution of both CS and $\neg(m : K(tr') \cup IK)$.*

Proof. Recall the set $D_V = \{d_X \mid X \in \mathcal{V}\}$ is chosen fresh, i.e. $D_V \not\preceq T_X$ for each $X \in \mathcal{V}$ with $X : T_X \in CS$ for some T_X, and also $D_V \not\preceq IK \cup K(tr') \cup \{m\}$.

The non trivial case is $(\Rightarrow)$, that is, when we know that σ is a solution of CS and $\neg(m : K(tr') \cup IK)$ then $\sigma_V = \{X \rightarrow c_X \mid X \in \mathcal{V}\}$ (i.e. the substitution assigning a fresh constant c_X to each variable $X \in \mathcal{V}$) is also a solution of CS and $\neg(m : K(tr') \cup IK)$. We first note that since $C_V \subseteq IK$, then $C_V \subseteq T_X$ for every $X : T_X \in CS$, and hence $X\sigma_V = c_X \in C_V$ by definition, so the positive part CS is always solvable by σ_V. To show

that $\neg(m : K(tr') \cup IK)$ is solvable by σ_V, we need to show that $m \notin \mathcal{F}((K(tr') \cup IK)\sigma_V)$ assuming $m \notin \mathcal{F}((K(tr') \cup IK)\sigma)$ (since m is ground, $m\sigma = \sigma_V = m$).

We first check that it is possible to apply Lemma A.1.11 to $\mathcal{F}((K(tr') \cup IK)\sigma)$. Let X be a variable occurring in $K(tr') \cup IK$. We need to see that $X\sigma \in \mathcal{F}((K(tr') \cup IK)\sigma)$.

By Theorem A.1.13(1), there is a constraint $(X : K(tr'') \cup IK) \in CS$ s.t. tr'' is a prefix trace of tr'. Since σ is a solution of CS, $X\sigma \in \mathcal{F}((K(tr'') \cup IK)\sigma)$. By Theorem A.1.13(2), $X\sigma \in \mathcal{F}((K(tr') \cup IK)\sigma)$. Hence we apply Lemma A.1.11, and obtain that $m \notin \mathcal{F}((K(tr') \cup IK)\sigma)$ implies $m \notin \mathcal{F}_\delta((K(tr') \cup IK)\rho_V)\sigma'$. Then by Lemma A.1.12, $m \notin \mathcal{F}_\delta((K(tr') \cup IK)\rho_V)\delta$. By Lemma A.1.8, $m \notin \mathcal{F}((K(tr') \cup IK)\rho_V\delta)$, and finally by Corollary A.1.6 $m \notin \mathcal{F}((K(tr') \cup IK)\sigma_V)$. □

From Section 2.5

Lemma 2.5.12. *Let ϕ be a closed $\mathcal{PS}$-LTL formula, tr be a trace and IK be an initial intruder knowledge, and let σ be a ground substitution such that $var(tr) \subseteq dom(\sigma)$. Then $\langle tr\sigma, IK\rangle \models \phi$ iff $\sigma \models' \mathbf{T}(\phi, tr, IK)$.*

Proof. Correctness of Step 1. We first show that Step 1 is correct: $\langle tr\sigma, IK\rangle \models \phi$ iff $\sigma \models' \lfloor\phi\rfloor tr$. We proceed by induction on the number of rewriting steps of $\lfloor\cdot\rfloor\cdot$. Base cases:

- $\phi = \mathtt{Y}\psi$ and $tr = \langle\rangle$. By definition of $\models'$, $\sigma \not\models' \lfloor\mathtt{Y}\psi\rfloor\langle\rangle$, since $\lfloor\mathtt{Y}\psi\rfloor\langle\rangle = \mathtt{false}$. Also by definition of $\models$, $\langle\langle\rangle, IK\rangle \not\models \mathtt{Y}\psi$.
- Cases $\phi = \mathtt{true}$ and $\phi = \mathtt{false}$ are immediate.
- $\phi = \mathtt{learn}(m)$. Since ϕ is closed, then m is ground. Now, $\sigma \models' \lfloor learn(m)\rfloor tr$ iff $m \in \mathcal{F}((K(tr) \cup IK)\sigma)$, since by definition $\lfloor learn(m)\rfloor tr = m : K(tr) \cup IK$. Also by definition of $\models$, $\langle tr\sigma, IK\rangle \models \mathtt{learn}(m)$ iff $m \in \mathcal{F}(K(tr\sigma) \cup IK)$.
- $\phi = p(d_1, \ldots, d_n)$, $tr = \langle\rangle$ immediate.
- $\phi = p(d_1, \ldots, d_n)$, $tr = \langle tr'\ q(e_1, \ldots, e_m)\rangle$, with $p \neq q$ or $n \neq m$. Here, $\sigma \not\models' \lfloor p(d_1, \ldots, d_n)\rfloor tr$, since by definition $\lfloor p(d_1, \ldots, d_n)\rfloor tr = \mathtt{false}$. Also $\langle tr\sigma, IK\rangle \not\models \phi$, since $p(d_1, \ldots, d_n) \neq q(e_1, \ldots, e_n)\sigma$.
- $\phi = p(d_1, \ldots, d_n)$, $tr = \langle tr'\ p(e_1, \ldots, e_n)\rangle$. Here, $\sigma \models' \lfloor p(d_1, \ldots, d_n)\rfloor tr$ iff $\sigma \models' d_1 = e_1 \wedge \cdots \wedge d_n = e_n$ iff by definition of $\models'$, $\sigma \models' d_i = e_i$ for all $i \in [1 \ldots n]$, iff again by definition of $\models'$, $d_i = e_i\sigma$, for all $i \in [1 \ldots n]$, iff $p(d_1, \ldots, d_n) = p(e_1, \ldots, e_n)\sigma$, iff by definition of $\models$, $\langle tr\sigma, IK\rangle \models \phi$.

Inductive cases:

- $\phi = \neg\psi$. $\sigma \models' \lfloor\phi\rfloor tr$ iff $\sigma \models' \neg\lfloor\psi\rfloor tr$ iff $\sigma \not\models' \lfloor\psi\rfloor tr$ iff by inductive hypothesis (since ϕ is closed, ψ is closed as well) $\langle tr\sigma, IK\rangle \not\models \psi$ iff $\langle tr\sigma, IK\rangle \models \phi$. (Similarly for cases $\wedge$ and $\vee$.)
- $\phi = \mathtt{Y}\psi$ and $tr = \langle tr'\ e\rangle$. Notice that ψ is closed. $\sigma \models' \lfloor\phi\rfloor tr$ iff $\sigma \models' \lfloor\psi\rfloor tr'$ iff by inductive hypothesis $\langle tr'\sigma, IK\rangle \models \psi$ iff by definition, $\langle tr\sigma, IK\rangle \models \phi$.

- $\phi = \phi_1 \mathtt{S} \phi_2$ and $tr = \langle\rangle$. $\sigma \models' \lfloor \phi \rfloor \langle\rangle$ iff $\sigma \models' \lfloor \phi_2 \rfloor \langle\rangle$ iff by inductive hypothesis $\langle\langle\rangle, IK\rangle \models \phi_2$ iff $\langle\langle\rangle, IK\rangle \models \phi$.

- $\phi = \phi_1 \mathtt{S} \phi_2$ and $tr = \langle tr'\ e\rangle$.

  $\sigma \models' \lfloor \phi_1 \mathtt{S} \phi_2 \rfloor tr$ iff by definition of $\lfloor \cdot \rfloor$, $\sigma \models' \lfloor \phi_1 \mathtt{S} \phi_2 \rfloor tr'$ and $\sigma \models' \lfloor \phi_1 \rfloor tr$, or $\sigma \models' \lfloor \phi_2 \rfloor tr$. Iff, by inductive hypothesis:
  $\langle tr'\sigma, IK\rangle \models (\phi_1 \mathtt{S} \phi_2)$ and $\langle tr\sigma, IK\rangle \models \phi_1$, or $\langle tr\sigma, IK\rangle \models \phi_2$ Iff
  $\exists i \in [0, length(tr'\sigma)] : (\langle tr'_i\sigma, IK\rangle \models \phi_2 \wedge \forall j \in [i{+}1, length(tr'\sigma)] : \langle tr'_j\sigma, IK\rangle \models \phi_1)$ and $\langle tr\sigma, IK\rangle \models \phi_1$, or $\langle tr\sigma, IK\rangle \models \phi_2$ Iff
  $\exists i \in [0, length(tr\sigma){-}1] : (tr_i\sigma \models \phi_2 \wedge \forall j \in [i{+}1, length(tr\sigma){-}1] : \langle tr_j\sigma, IK\rangle \models \phi_1)$ and $\langle tr\sigma, IK\rangle \models \phi_1$, or $\langle tr\sigma, IK\rangle \models \phi_2$. iff
  $\exists i \in [0, length(tr\sigma)] : (\langle tr_i\sigma, IK\rangle \models \phi_2 \wedge \forall j \in [i+1, length(tr\sigma)] : \langle tr_j\sigma, IK\rangle \models \phi_1)$ iff
  $\langle tr\sigma, IK\rangle \models (\phi_1 \mathtt{S} \phi_2) = \phi$.

- Cases $\phi = \exists\psi$ and $\phi = \forall\psi$ are immediate.

The correctness of Step 2 follows from the correctness of standard formula manipulations.

Correctness of Step 3.
Let π be the result of one rewrite of Step 3 for a formula ϕ. Then $\sigma \models' \phi$ iff $\sigma \models' \pi$.
Cases:

- $\phi = \forall v.\psi$ and $\pi = \psi$, for v not free in ψ. Then $\sigma \models' \phi$ iff for all $t \in \mathcal{T}^+ : \sigma \models' \psi[^t/_v]$ iff $\sigma \models' \psi$, since v is not free in ψ, and then $\sigma \models' \pi$.

- $\phi = \forall v.\neg(v = t)$ and $\pi = \mathtt{false}$. Then $\sigma \models' \phi$ iff for all $t' \in \mathcal{T}^+ : \sigma \models' \neg(t' = t)$ iff for all $t' \in \mathcal{T}^+ : \sigma \not\models' (t' = t)$ iff for all $t' \in \mathcal{T}^+ : t' \neq t\sigma$, which is false (take $t' = t\sigma$); likewise, $\sigma \not\models' \mathtt{false}$, which is $\sigma \not\models' \pi$.

- $\phi = \forall v.(v = t)$ and $\pi = \mathtt{false}$. Then $\sigma \models' \phi$ iff for all $t' \in \mathcal{T}^+ : \sigma \models' (t' = t)$ (since v does not occur in t by assumption, see last paragraph of Section 2.5.1) iff for all $t' \in \mathcal{T}^+ : \sigma \models' (t' = t)$ iff for all $t' \in \mathcal{T}^+ : t' = t\sigma$, which is false (take $t' \neq t\sigma$, for example letting $t' = h(t\sigma)$). Likewise, $\sigma \not\models' \mathtt{false}$, which is $\sigma \not\models' \pi$.

- $\phi = \forall v.(v : K)$ and $\pi = \mathtt{false}$. Then $\sigma \models' \phi$ iff for all $t' \in \mathcal{T}^+ : \sigma \models' (t' : K)$ iff for all $t' \in \mathcal{T}^+ : t' \in \mathcal{F}(K\sigma)$, which is false by letting t' be any constant (e.g., an agent identity) not occurring in $K\sigma$, since the universe of constants is enumerable and $K\sigma$ finite. Likewise, $\sigma \not\models' \mathtt{false}$, which is $\sigma \not\models' \pi$.

- $\phi = \forall v.\neg(v : K)$ and $\pi = \mathtt{false}$. Then $\sigma \models' \phi$ iff for all $t' \in \mathcal{T}^+ : \sigma \not\models' (t' : K)$ iff for all $t' \in \mathcal{T}^+ : t' \notin \mathcal{F}(K\sigma)$, which is false by letting t' be $e \in IK \subseteq K$ (see Section 2.2.4). Likewise, $\sigma \not\models' \mathtt{false}$, which is $\sigma \not\models' \pi$.

- $\phi = \forall v.(\psi_1 \wedge \psi_2)$, $\pi = \forall v.\psi_1 \wedge \forall v.\psi_2$. Then $\sigma \models' \phi$ iff for all $t \in \mathcal{T}^+ : \sigma \models' (\psi_1 \wedge \psi_2)[^t/_v]$ iff for all $t \in \mathcal{T}^+ : \sigma \models' (\psi_1[^t/_v] \wedge \psi_2[^t/_v])$ iff for all $t \in \mathcal{T}^+ : \sigma \models' \psi_1[^t/_v]$ and for all $t \in \mathcal{T}^+ : \sigma \models' \psi_2[^t/_v]$ iff $\sigma \models' \forall v.\psi_1$ and $\sigma \models' \forall v.\psi_2$, which is $\sigma \models' \pi$.

- $\phi = \forall v.(\psi_1 \vee \psi_2)$, $\pi = \forall v.\psi_1 \vee \forall v.\psi_2$ for v not free in ϕ_1 or v not free in ϕ_2.
  We see the case in which v not free in ϕ_1 (the other two cases are analogous): $\sigma \models' \phi$ iff for all $t \in \mathcal{T}^+ : \sigma \models' (\psi_1 \vee \psi_2)[^t/v]$ iff for all $t \in \mathcal{T}^+ : \sigma \models' \psi_1[^t/v]$ or $\sigma \models' \psi_2[^t/v]$ iff for all $t \in \mathcal{T}^+ : \sigma \models' \psi_1$ or $\sigma \models' \psi_2[^t/v]$ iff $\sigma \models' \psi_1$ or for all$t \in \mathcal{T}^+ : \sigma \models' \psi_2[^t/v]$ iff $\sigma \models' \forall v.\psi_1 \vee \forall v.\psi_2$.

□

Lemma 2.5.14. *Let ϕ be a closed $\mathcal{PS}$-LTL formula, and $\langle Sc, IK, CS, tr\rangle$ a state from Procedure 2.3.4. Assume that $\pi = \mathbf{T}(\phi, tr, IK)$ is a well-behaving EF formula, $\pi = \exists v_1 \ldots \exists v_n.\psi$ with $\psi = \bigvee_j \psi_j$ and $\psi_j = \bigwedge_i \pi_{j,i}$. Then:*

1. *$\mathbf{D}(\pi, CS)$ succeeds and returns a substitution σ implies that $\sigma \models' \pi$, with $tr\sigma$ valid w.r.t. IK; and*
2. *$\sigma \models' \pi$, with $tr\sigma$ valid w.r.t. IK implies that there exists a substitution γ s.t. $\mathbf{D}(\pi, CS)$ succeeds and returns γ.*

Proof. (1): We know that there exists some disjunct ψ_j s.t. $D(\pi, CS)$ succeeds and returns $\sigma = \rho\rho_k\sigma_V$.

As in the definition of Procedure 2.5.13, we assume that $\psi_j = A_j \wedge B_j \wedge C_j \wedge D_j$, as follows (recall that $\equiv$ denotes syntactic equality):

$$A_j \equiv (L_j = R_j) \quad (1)$$
$$B_j \equiv \neg(L_j^\neg = R_j^\neg) \quad (2)$$
$$C_j \equiv (m_j : K_j) \quad (3)$$
$$D_j \equiv \neg(m_j^\neg : K_j^\neg) \quad (4)$$

This simplifies notation and can be generalized in a straightforward way. We now show that $\sigma \models' \pi$. By definition, $\sigma \models' \pi$ holds iff there exists $t_1 \in \mathcal{T}^+ \ldots t_n \in \mathcal{T}^+$ s.t. $\sigma \models' \psi_j[^{t_1,\ldots,t_n}/v_1, \ldots, v_n]$. Let $t_i = v_i\sigma$ for $i \in [1 \ldots n]$. Then by (1)-(4) we have to show that:

1. $\sigma \models' A_j[^{v_1\sigma,\ldots,v_n\sigma}/v_1, \ldots, v_n]$ and
2. $\sigma \models' B_j[^{v_1\sigma,\ldots,v_n\sigma}/v_1, \ldots, v_n]$ and
3. $\sigma \models' C_j[^{v_1\sigma,\ldots,v_n\sigma}/v_1, \ldots, v_n]$ and
4. $\sigma \models' D_j[^{v_1\sigma,\ldots,v_n\sigma}/v_1, \ldots, v_n]$

This will imply the thesis.

- We first show that $\sigma \models' A_j[^{v_1\sigma,\ldots,v_n\sigma}/v_1, \ldots, v_n]$. Since Step 2 of the procedure succeeded with unifier ρ, we have that $L_j\rho = R_j\rho$. Thus $L_j(\rho\rho_k\sigma_V) = R_j(\rho\rho_k\sigma_V)$, and since $\sigma = \rho\rho_k\sigma_V$, $\sigma \models' (L_j\sigma = R_j)$ by the definition of $\models'$. We know that $var(L_j) \subseteq \{v_1, \ldots, v_n\}$. Recall that we assume that variables v_i, $i \in [1 \ldots n]$ are disjoint from the variables in $var(tr)$, which implies that $\{v_1, \ldots, v_n\} \cap var(R_j) = \emptyset$. Then $\sigma \models' (L_j\sigma = R_j)$ is equivalent to $\sigma \models' (L_j = R_j)[^{v_1\sigma,\ldots,v_n\sigma}/v_1, \ldots, v_n]$, establishing the claim.

- Now we show that $\sigma \models' C_j[^{v_1\sigma,\ldots,v_n\sigma}/v_1,\ldots,v_n]$. Since Step 3 of the procedure succeeded, we know that $CS\rho \cup \{m_j\rho : K_j\rho\}$ is solvable with partial solution ρ_k. By Soundness in Theorem 2.2.15, $m_j\rho\rho_k\sigma_V \in \mathcal{F}(K_j\rho\rho_k\sigma_V)$, since σ_V is always a solution of any simple constraint set $\{X : T_X \mid X \in \mathcal{V}\}$ with $C_V \subseteq T_X$ (see Section 2.4). Hence by the definition of $\models'$, $\sigma \models' (m_j\sigma : K_j)$, since no v_i occurs in K_j, $i \in [1\ldots n]$, and $var(m_j) \subseteq \{v_1,\ldots,v_n\}$ we obtain again that $\sigma \models' C_j[^{v_1\sigma,\ldots,v_n\sigma}/v_1,\ldots,v_n]$.

- We now show that $\sigma \models' D_j[^{v_1\sigma,\ldots,v_n\sigma}/v_1,\ldots,v_n]$ holds. Since Step 5 of the procedure holds, we know by Corollary 2.4.2 that $m_j^\neg\sigma \notin \mathcal{F}(K_j^\neg\sigma)$, which by definition of $\models'$ implies that $\sigma \models' \neg(m_j^\neg\sigma : K_j^\neg)$. Since no variable v_i occurs in $K_j^\neg$, $i \in [1\ldots n]$, and $var(m_j^\neg) \subseteq \{v_1,\ldots,v_n\}$, we finally obtain that $\sigma \models' \neg(m_j^\neg : K_j^\neg)[^{v_1\sigma,\ldots,v_n\sigma}/v_1,\ldots,v_n]$.

- We finally establish $\sigma \models' B_j[^{v_1\sigma,\ldots,v_n\sigma}/v_1,\ldots,v_n]$. Since Step 6 succeeded, we know that $L_j^\neg(\rho\rho_k\sigma_V) \neq R_j(\rho\rho_k\sigma_V)$. By the definition of $\models'$, this implies that $\sigma \models' \neg(L_j^\neg\sigma = R_j^\neg)$. We finally obtain that $\sigma \models \neg(L_j^\neg = R_j^\neg)[^{v_1\sigma,\ldots,v_n\sigma}/v_1,\ldots,v_n]$ since no v_i occurs in $R_j^\neg$, $i \in [1\ldots n]$ and $var(L_j^\neg) \subseteq \{v_1,\ldots,v_n\}$.

Finally, to see that $tr\sigma$ is valid, it is sufficient to observe that $\rho\rho_k$ is a partial solution of CS and σ_V is a solution of $CS\rho\rho_k$.

(2) We know that $\sigma \models' \pi$. By the definition of $\models'$, this holds iff there exist an index j and $t_1 \in \mathcal{T}^+\ldots t_n \in \mathcal{T}^+$ s.t. $\sigma \models' \psi_j[^{t_1,\ldots,t_n}/v_1,\ldots,v_n]$. Here we can assume (by alpha conversion) that $dom(\sigma) \cap \{v_1,\ldots,v_n\} = \emptyset$.

As done in the previous case and in the definition of Procedure 2.5.13, we assume that $\psi_j = A_j \wedge B_j \wedge C_j \wedge D_j$, as follows (recall that $\equiv$ denotes syntactic equality):

$$A_j \equiv (L_j = R_j) \quad (1)$$
$$B_j \equiv \neg(L_j^\neg = R_j^\neg) \quad (2)$$
$$C_j \equiv (m_j : K_j) \quad (3)$$
$$D_j \equiv \neg(m_j^\neg : K_j^\neg) \quad (4)$$

This simplifies notation and does not result in a loss of generality. By assumption, we know that $\sigma \models' \pi$, hence by the definition of $\models'$ we know that there exists $t_1 \in \mathcal{T}^+\ldots t_n \in \mathcal{T}^+$ s.t. $\sigma \models' \psi_j[^{t_1,\ldots,t_n}/v_1,\ldots,v_n]$. Let $\sigma' = [^{t_1,\ldots,t_n}/v_1,\ldots,v_n]$. This implies, by (1)-(4), that the following items hold:

1. $\sigma \models' A_j\sigma'$ and
2. $\sigma \models' B_j\sigma'$ and
3. $\sigma \models' C_j\sigma'$ and
4. $\sigma \models' D_j\sigma'$

We need to show that there exist appropriate ρ and ρ_k s.t. Steps 2,3,5 and 6 of the procedure succeed. We thus let γ be $\rho\rho_k\sigma_V$.

- We first see that there exists ρ s.t. Step 2 succeeds. Recall we assume that variables from v_i, $i \in [1 \dots n]$ are disjoint from the variables in $var(tr)$ (see last paragraph of Section 2.5.1), which implies that $\{v_1, \dots, v_n\} \cap var(R_j) = \emptyset$. Thus $\sigma \models' A_j\sigma'$ is equivalent to $\sigma \models' (L_j\sigma' = R_j)$, which then by definition of $\models'$ implies that

$$L_j\sigma' = R_j\sigma \tag{5}$$

  Hence $\sigma\sigma'$ is a unifier. Now, recall that Step 2 succeeds iff L_j and R_j unify. Therefore by (5) Step 2 succeeds. In addition, Step 2 returns a m.g.u. ρ of L_j and R_j. By well-known results on unification (e.g. [34]) we have that for some ρ_2, $\sigma\sigma' = \rho\rho_2$.

- We now show that there exists ρ_k s.t. Step 3 succeeds. We know that $\sigma \models' C_j\sigma'$, that is, $\sigma \models' (m_j\sigma' : K_j)$ which implies that $m_j\sigma' \in \mathcal{F}(K_j\sigma)$, and hence that $m_j\rho\rho_2 \in \mathcal{F}(K_j\rho\rho_2)$. By Completeness in Theorem 2.2.15, $m_j\rho : K_j\rho$ is solvable with a partial solution ρ_k and there exists a substitution ξ s.t. $\sigma\sigma' = \rho\rho_k\xi$. Hence, Step 3 succeeds returning partial solution ρ_k.

- We now see that Step 5 succeeds. We know that $\sigma \models' D_j\sigma'$, hence that $\sigma \models' \neg(m_j^{\neg}\sigma' : K_j^{\neg})$, which by the definition of $\models'$ implies that $m_j^{\neg}\sigma' \notin \mathcal{F}(K_j^{\neg}\sigma)$, which is $m_j^{\neg}\rho\rho_k\xi \notin \mathcal{F}(K_j^{\neg}\rho\rho_k\xi)$. Hence ξ is a solution of $\neg(m_j^{\neg}\rho\rho_k : K_j^{\neg}\rho\rho_k)$. By Theorem 2.4.4, σ_V is also a solution of $\neg(m_j^{\neg}\rho\rho_k : K_j^{\neg}\rho\rho_k)$, and thus $\neg(m_j^{\neg}\rho\rho_k\sigma_V : K_j^{\neg}\rho\rho_k\sigma_V)$ is solvable. Thus by Corollary 2.4.2 we conclude that applying $\mathbf{P}$ to $m_j^{\neg}\rho\rho_k\sigma_V : K_j^{\neg}\rho\rho_k\sigma_V$ must fail, and hence Step 5 must succeed.

- We finally see that Step 6 succeeds. We know that $\sigma \models' B_{j,i}\sigma$, i.e. that $\sigma \models' \neg(L_{j,i}^{\neg}\sigma' = R_{j,i}^{\neg})$. By the definition of $\models'$, the latter implies that $L_{j,i}^{\neg}\sigma' \neq R_{j,i}^{\neg}\sigma$. Thus $L_{j,i}^{\neg}\rho\rho_k\xi \neq R_{j,i}^{\neg}\rho\rho_k\xi$. Hence $L_{j,i}^{\neg}\rho\rho_k \neq R_{j,i}^{\neg}\rho\rho_k$, and so $L_{j,i}^{\neg}\rho\rho_k\sigma_V \neq R_{j,i}^{\neg}\rho\rho_k\sigma_V$, ensuring that Step 6 succeeds.

□

A.2 From Chapter 5

This appendix provides a proof of Theorem 5.4.2. The target properties involve events that record the whole session history, hence they cannot be automatically handled via TulaFale and ProVerif. Instead, we automatically establish a series of intermediate properties, and we compose these properties using standard (manual) proof techniques for the pi calculus.

Before detailing the proof, we set up auxiliary notations for processes and present an intermediate script that models our protocol more abstractly.

Additional definitions for processes We reason about TulaFale scripts as processes in the applied pi calculus [23, 50]. We use the concrete syntax of TulaFale for all processes. We let P, Q, T, C, S, $\mathcal{S}$ range over processes. We write $P \equiv Q$ when P and Q are structurally equivalent, $P \rightarrow Q$ when P reduces to Q in a single reduction step (that is, a communication on a channel, a predicate evaluation, or a test), and $P \rightarrow^* Q$ when P reduces to Q in $n \geq 0$ steps.

A context is a process with holes ($[\cdot]$) instead of some of its subprocesses. We consider several kinds of contexts: an *evaluation context* is a context with a single hole occurring at top level (that is, under restrictions and parallel compositions, but not under guards); for instance, an arbitrary active attacker is modeled as an evaluation context with no events. A *guarded context* is a context where all occurrences of the holes are guarded. A *linear context* is either the hole, or an evaluation context applied to a (simpler) linear context under a linear guard: an input, an output, a filter, or an event. We always assume that restricted names are distinct from free names in their enclosing contexts—this can be systematically enforced using local renaming. A process P *records the events* **begin** C, ..., **end** C′, ... when these events occur in evaluation contexts in P, that is, $P \equiv E[\ \textbf{begin}\ \text{C} \mid \ldots \mid \textbf{end}\ \text{C}' \mid \ldots\ ]$ for some evaluation context E.

We let $\mathcal{S}^{\mathcal{N}}$ be the script presented in Section 5.4.2 and defined in `ssws-secrm.tf`. With these definitions, we arrive at the following restatement of the theorem:

RESTATEMENT OF THEOREM 5.4.2.

Let E be an evaluation context with no events and where the name ***secret*** *does not occur. If $E[\mathcal{S}^{\mathcal{N}}] \rightarrow^* P$, then*

1. *If P records* ***end*** *C3n(sc,sessionId,H,n), then P records either* ***begin*** *C3n(sc,sessionId, H,n) or* ***begin*** *Leak(sc).*
2. *If P records* ***end*** *C4n(sc,sessionId,H,n), then P records either* ***begin****C4n(sc,sessionId, H,n) or* ***begin*** *Leak(sc).*
3. *If P outputs* ***secret*** *on a free channel, then P records* ***begin*** *Leak(sc) and either event* ***begin*** *C3n(sc,sessionId,H,n) or event* ***begin*** *C4n(sc,sessionId,H,n) where* ***secret*** *is the body of the last envelope of H.*

In addition to $\mathcal{S}^{\mathcal{N}}$, used in the statement of Theorem 5.4.2, our proof relies on $\mathcal{S}^{\mathcal{A}}$, a similar but more abstract script defined in `ssws-secrm-a.tf`. TulaFale terminates rapidly on $\mathcal{S}^{\mathcal{A}}$ (within about twenty minutes) but apparently diverges on $\mathcal{S}^{\mathcal{N}}$. To complete the proof of Theorem 5.4.2, we manually relate the behaviors of $\mathcal{S}^{\mathcal{N}}$ and $\mathcal{S}^{\mathcal{A}}$. Next, we outline the differences between $\mathcal{S}^{\mathcal{N}}$ (on the left) and $\mathcal{S}^{\mathcal{A}}$ (on the right). The client processes have the following structures:

```
private channel dc(item,item,item,item).
C[
  new sessionId:string;
  out dc(sc,sessionId,[],zero))
][
  !in dc(sc,sessionId,H,n);
  L3C[
    begin C3n (sc,sessionId,[Req @ H],n);
    L4C[
      end C4n (sc,sessionId,[Resp Req @ H],n);
      out dc(sc,sessionId,
             [Resp Req @ H],succ(n)). ]]]
```

```
private channel dc(item,item).
C[
  new sessionId:string;
  out dc(sc,sessionId)
][
  !in dc(sc,sessionId); in env(n);
  L3C[
    begin C3a (sc,sessionId,[Req],n);
    L4C[
      end C4a (sc,sessionId,[Resp Req],n);
      out dc(sc,sessionId).
                              ]]]
```

Channel dc—named sessionDbC in the scripts—appears in $\mathcal{S}^{\mathcal{N}}$ and $\mathcal{S}^{\mathcal{A}}$ only as explicited above. The rest of the scripts is abstracted as a binary context $C[\cdot][\cdot]$ and two linear contexts, $L_{3C}[\cdot]$ and $L_{4C}[\cdot]$.

The client uses a message on private channel dc to record the state for every running session. To initiate a session, the client creates a fresh identifier sessionId and sends a message on dc representing the session state: its identifier and its security context sc (bound by the context $C[\cdot][\cdot]$). To extend an existing session, the client reads the session state on dc, performs the exchange, then puts back an updated state on dc. In $\mathcal{S}^{\mathcal{N}}$, the message also records the session history and the number for the next SOAP message, starting with an empty history and zero; the history is included in all events. In $\mathcal{S}^{\mathcal{A}}$, only sc and sessionId are recorded, the history is not used, and the next-message number is an arbitrary value provided by the environment on channel env; the simplified events are called C3a and C4a. (In the scripts, the message number n is named msgNumber.)

Similarly, the server processes in $\mathcal{S}^{\mathcal{N}}$ and $\mathcal{S}^{\mathcal{A}}$ have the following structures:

```
private channel ds(item,item,item,item).
S[
  in public(sessionId);
  L_ARC[out ds(sc,sessionId,[],zero)]
][
  !in ds(sc,sessionId,H,n);
  L_3S[
    end C3n (sc,sessionId,[Req @ H],n);
    L_4S[
      begin C4n (sc,sessionId,[Resp Req @ H],n);
      out ds(sc,sessionId,
             [Resp Req @ H],succ(n)). ]]]
```

```
private channel ds(item,item).
S[
  in public(sessionId);
  L_ARC[out ds(sc,sessionId)]
][
  !in ds(sc,sessionId); in env(n);
  L_3S[
    end C3a (sc,sessionId,[Req],n);
    L_4S[
      begin C4a (sc,sessionId,[Resp Req],n);
      out ds(sc,sessionId).
                                  ]]]
```

Channel ds—named sessionDbS in the scripts—appears in $\mathcal{S}^{\mathcal{N}}$ and $\mathcal{S}^{\mathcal{A}}$ only as explicited above, and the rest of the scripts is abstracted using a binary context $S[\cdot][\cdot]$ and three linear contexts, L_{ARC}, $L_{3S}[\cdot]$, and $L_{4S}[\cdot]$. The context L_{ARC} is part of the implementation of an anti-replay cache; before sending a message a message on ds, it checks that this is the first session with this identifier, and otherwise does nothing. (See Lemma A.2.1 for our pi calculus implementation of the anti-replay cache.)

As a simplification, our server processes input new session identifiers sessionId directly from the environment, on channel public—named ContextProvideSessionId in the scripts—instead of reading them from received first messages.

Reachable states of $\mathcal{S}^{\mathcal{N}}$ The following lemma gives an explicit representation for the reachable states of the protocol $\mathcal{S}^{\mathcal{N}}$. This representation is used later to relate the recorded events of $\mathcal{S}^{\mathcal{N}}$ to those of $\mathcal{S}^{\mathcal{A}}$. Crucially, the lemma concerns only the management of state on three channels; its correctness does not depend on the rest of the protocol.

Lemma A.2.1 (Reachable States for $\mathcal{S}^{\mathcal{N}}$)**.** *We let*

$$
\begin{aligned}
C'_{sc} &= \textbf{\textit{new x:string; out dc(sc,x,[],zero)}} \\
S'_{y,sc} &= \textbf{\textit{in cache(y°);(out cache([y@y°]) |if not (y in y°) then out ds(sc,y,[],zero))}} \\
C_{succ} &= \textbf{\textit{!in dc(sc,x,H,n);LC[out dc(sc,x,H',succ(n))]}} \\
S_{succ} &= \textbf{\textit{!in ds(sc,y,H,n);LS[out ds(sc,y,H',succ(n))]}} \\
T^{\mathcal{N}} &= \textbf{\textit{private channel dc(item,item,item,item).}} \\
&\quad \textbf{\textit{private channel ds(item,item,item,item).}} \\
&\quad \textbf{\textit{private channel cache(items).}} \\
&\quad E\big[C_{succ} \mid S_{succ} \mid \\
&\qquad C_{X'}[C'_{sc}]_{sc\in X'} \mid S_{Y'}[S'_{y,sc}]_{(y,sc)\in Y'} \mid \textbf{\textit{out cache(Y°)}} \mid \\
&\qquad \textstyle\prod_{(x,n)\in X} C_{x,n}[\ \textbf{\textit{out dc(sc,x,H}}_{x,n}\textbf{\textit{,n)}}\ ] \mid \\
&\qquad \textstyle\prod_{(y,n)\in Y} S_{y,n}[\ \textbf{\textit{out ds(sc,y,H}}_{y,n}\textbf{\textit{,n)}}]\big]
\end{aligned}
$$

where X' range over finite multisets of terms; X, Y, and Y' range over finite sets of pairs of terms and (term-coded) integers; Y° is a list collecting all first terms of pairs in Y; E ranges over evaluation contexts; $C_{X'}$ range over contexts with a hole for each element of X'; $S_{Y'}$ range over contexts with a hole for each element of Y'; $C_{x,n}$ and $S_{y,n}$ range over linear contexts indexed by X and Y; LC and LS are linear contexts appearing in $\mathcal{S}^{\mathcal{N}}$ such that $\mathcal{S}^{\mathcal{N}} \equiv T^{\mathcal{N}}_{\emptyset,\emptyset}$. Let E' be an evaluation context, and assume that dc, ds, and cache do not occur in any of those contexts.

If $E'[\mathcal{S}^{\mathcal{N}}] \to^* P$, then

1. *there exists $T^{\mathcal{N}}$ such that $P \equiv T^{\mathcal{N}}$;*
2. *reduction steps on dc communicate messages $(sc, x, H_{x,i}, i)$ for pairwise-distinct (x, i)s such that $(x, n) \in X$ and $i < n$.*
3. *reduction steps on ds communicate messages $(sc, y, H_{y,i}, i)$ for pairwise-distinct (y, i)s such that $(y, n) \in Y$ and $i < n$.*

In the lemma, the process $T^{\mathcal{N}}$ represents a reachable state of $\mathcal{S}^{\mathcal{N}}$, parameterized by four index sets: the sets X and Y keep track of ongoing sessions for the client and the server, respectively, with initially $X = Y = \emptyset$; the sets X' and Y' keep track of sessions that have not started yet.

Proof. The proof is by induction on the number of steps in $E'[\mathcal{S}^{\mathcal{N}}] \to^k P$. The base case holds by construction for $T^{\mathcal{N}}_{\emptyset,\emptyset}$. For the inductive case, the inductive hypothesis yields $T^{\mathcal{N}}$ such that $E'[\mathcal{S}^{\mathcal{N}}] \to^k \equiv T^{\mathcal{N}} \to P$. We perform a case analysis on the final reduction step using the structure of $T^{\mathcal{N}}$, with the following cases:

- The step is a communication on private channel dc. By construction, the input is the replicated input C_{succ}, and there are two subcases for the output:
  - The output is in C'_{sc} for some $sc \in X'$. Using structural equivalence, we rename the restricted name sessionId to some globally-fresh name x and lift the restriction on x to E. We remove sc from X', add (x, zero) to X, and define $H_{x,0}$ and $C_{x,0}[\_]$ to match the triggered process $LC[\dots]$.

  - The output is in $C_{x,n}$ for some $(x, n) \in X$. Thus, $C_{x,n}$ is an evaluation context. Using structural equivalence, we merge this context with E, we replace (x, n) by $(x, n + 1)$ in X, and define $H_{x,n+1}$ and $C_{x,n+1}$ to match the triggered process $LC[\ldots]$.

- The step is a communication on private channel ds. By construction, the input is the replicated input S_{succ} and the output is in $S_{y,n}$ for some $(y, n) \in Y$. Thus, $S_{y,n}$ is an evaluation context. As in the case above, we merge $S_{y,n}$ with E, replace (y, n) by $(y, n+1)$ in Y, and define $H_{y,n+1}$ and $S_{y,n+1}$ to match the triggered process $LS[\ldots]$.
- The step is a communication on private channel cache. By construction, the only output on this channel is **out** cache(Y°); the input is $S'_{y,sc}$ for some $(y, sc) \in Y'$. We distinguish two subcases:
  - If $y \in Y^\circ$, then the triggered process is an output on cache of a list with the same elements as Y° in parallel with an inert process (since the inclusion test succeeds). To conclude, we let Y' become Y' minus (y, sc).
  - Otherwise, we let Y' become Y' minus (y, sc), let Y become Y plus $(y,$ zero$)$, let $H_{y,0}$ be $[]$, and let $S_{y,0}$ be $[\cdot]$, so that the triggered process is an updated message on cache in parallel with the new element in the product on Y.
- All other communication steps preserve the structure of $T^{\mathcal{N}}$, for some updated sets X', Y' and contexts E, $C_{X'}$, $C_{x,n}$, $S_{Y'}$, $S_{y,n}$.

□

Relating events of $\mathcal{S}^{\mathcal{N}}$ and $\mathcal{S}^{\mathcal{A}}$ In $\mathcal{S}^{\mathcal{N}}$, the correspondences C3n and C4n record similar session information, namely a tuple of the form (sc,sessionId, [M @ H],n). We define their projection to events that may occur in $\mathcal{S}^{\mathcal{A}}$ as follows:

event C3n (sc,sessionId, [Req @ H],n)$^{\#}$ = event C3a (sc,sessionId,[Req],n)

event C4n (sc,sessionId, [Resp Req @ H],n)$^{\#}$ = event C4a (sc,sessionId,[Resp Req],n)

Other events are left unchanged: Leak(sc)$^{\#}$ = Leak(sc). We lift this projection to processes: $P^{\#}$ is obtained from P by projecting any event occurring in P. Our next lemma shows that, for any run of $\mathcal{S}^{\mathcal{N}}$, there is a corresponding run of $\mathcal{S}^{\mathcal{A}}$ that records the same Leak events, and an event event C$^{\#}$ for each event event C. (On the other hand, $\mathcal{S}^{\mathcal{A}}$ has runs that cannot be simulated by $\mathcal{S}^{\mathcal{N}}$.)

Lemma A.2.2 ($\mathcal{S}^{\mathcal{A}}$ simulates $\mathcal{S}^{\mathcal{N}}$)**.** *For any evaluation context $E^{\mathcal{N}}$ with no events and where the name **secret** does not occur, for any reductions $E^{\mathcal{N}}[\mathcal{S}^{\mathcal{N}}] \rightarrow^* P$, there exist an evaluation context $E^{\mathcal{A}}$ and reductions $E^{\mathcal{A}}[\mathcal{S}^{\mathcal{A}}] \rightarrow^* T^{\mathcal{A}}$ such that*

1. *if P records e, then $T^{\mathcal{A}}$ records $e^{\#}$;*
2. *if $T^{\mathcal{A}}$ records f, then P records some e such that $f = e^{\#}$;*
3. *If P outputs **secret** on a free channel, then $T^{\mathcal{A}}$ also outputs **secret** on a free channel.*

Proof. By Lemma A.2.1(1), we have a process of the form $T^{\mathcal{N}}$ with $P \equiv T^{\mathcal{N}}$ (and in particular with the same events and outputs of **secret** on free channels). We obtain $T^{\mathcal{A}}$ from $T^{\mathcal{N}}$ by applying $\cdot^{\#}$ to every event; erasing the message number and the history from every input and output on dc and ds; inserting an input **in** env(n) after every input on dc and ds; and adding outputs $\prod_{i=0}^{n}$**!out** env($succ^i$(zero)) at top-level, where $n = max\{i \mid (z,i) \in Y\}$. Thus, we define $T^{\mathcal{A}}$ and $E^{\mathcal{A}}[\_]$ as follows:

$$
\begin{array}{rcl}
C'^{\mathcal{A}}_{sc} & = & \textbf{new}\ \text{x:}\textbf{string};\textbf{out}\ \text{dc}(sc,\text{x}) \\
S'^{\mathcal{A}}_{y,sc}(Y) & = & \textbf{in}\ \text{cache(y°)};(\textbf{out}\ \text{cache}([y@\ \text{y°}])\ |\textbf{if}\ \text{not}\ (y\ \textbf{in}\ \text{y°})\ \textbf{then out}\ \text{ds}(sc,y))] \\
C^{\mathcal{A}}_{succ} & = & \textbf{!in}\ \text{dc(sc,x)};\textbf{in}\ \text{env(n)};LC^{\#}[\textbf{out}\ \text{dc(sc,x)}] \\
S^{\mathcal{A}}_{succ} & = & \textbf{!in}\ \text{ds(sc,y)};LS^{\#}[\textbf{out}\ \text{ds(sc,y)}] \\
T^{\mathcal{A}} & = & \prod_{i=0}^{n}\ \textbf{!out}\ \text{env}(succ^i(\text{zero}))\ | \\
 & & \textbf{private channel}\ \text{dc}(\textbf{item},\textbf{item}). \\
 & & \textbf{private channel}\ \text{ds}(\textbf{item},\textbf{item}). \\
 & & \textbf{private channel}\ \text{cache}(\textbf{items}). \\
 & & E^{\#}\big[C^{\mathcal{A}}_{succ}\ |S^{\mathcal{A}}_{succ}\ | \\
 & & \quad C^{\#}_{X'}[\ C'^{\mathcal{A}}_{sc}\ ]_{sc\in X'}\ |S^{\#}_{Y'}[\ S'^{\mathcal{A}}_{y,sc}\ ]_{(y,sc)\in Y'}\ |\textbf{out}\ \text{cache}(Y^{\circ})\ | \\
 & & \quad \prod_{(x,n)\in X} C^{\#}_{x,n}[\ \textbf{out}\ \text{dc}(sc,x)\ ]\ | \\
 & & \quad \prod_{(y,n)\in Y} S^{\#}_{y,n}[\ \textbf{out}\ \text{ds}(sc,y)]\big] \\
E^{\mathcal{A}}[\_] & = & \prod_{i=0}^{n}\textbf{!out}\ \text{env}(succ^i(\text{zero}))\ |\ E^{\mathcal{N}\#}[\_]
\end{array}
$$

We show that, with these definitions, if $E^{\mathcal{N}}[\mathcal{S}^{\mathcal{N}}] \rightarrow^* P \equiv T^{\mathcal{N}}$, then $E^{\mathcal{A}}[\mathcal{S}^{\mathcal{A}}] \rightarrow^* T^{\mathcal{A}}$. The processes $T^{\mathcal{N}}$ and $T^{\mathcal{A}}$ differ only in the content of their events (but this content does not affect reductions) and on the messages communicated on dc and ds. Each communication step on dc is simulated by a matching communication step on dc immediately followed by a communication step on env to input a matching integer. Similarly, each communication step on ds is simulated by a matching communication step on ds and a communication step on env. Any other reduction step in $T^{\mathcal{N}}$ immediately carries over to $T^{\mathcal{A}}$, with matching effects. □

The following lemmas state properties verified by TulaFale and ProVerif: some structural properties for $\mathcal{S}^{\mathcal{N}}$, and the main correspondence for $\mathcal{S}^{\mathcal{A}}$. They are established by running TulaFale on scripts `ssws-secrm.tf` and `ssws-secrm-a.tf`, respectively.

Lemma A.2.3. *For any evaluation context E with no events, if $E[\mathcal{S}^{\mathcal{N}}] \rightarrow^* P$, then:*

1. *If P records event **end** C4n (sc,sessionId,[Resp Req @ H],n), then P also records event **begin** C3n (sc,sessionId,[Req @ H],n).*

2. *If P records event **begin** C4n (sc,sessionId,[Resp Req @ H],n), then P also records event **end** C3n(sc,sessionId,[Req @ H],n).*

3. *If P records event **begin** C3n(sc,sessionId,[Req @ H],succ(n)), then P also records event **end** C4n (sc,sessionId,H,n).*

4. *If P records event* ***end*** *C3n(sc,sessionId,[Req @ H],succ(n)), then P also records event* ***begin*** *C4n(sc,sessionId,H,n).*

Lemma A.2.4 (Agreement and Secrecy in $\mathcal{S}^{\mathcal{A}}$)**.** *For any evaluation context E with no events and where the name* **secret** *does not occur, if $E[\mathcal{S}^{\mathcal{A}}] \rightarrow^* P$, then:*

1. *If P records event* ***end*** *C3a(sc,sessionId,[Req],n), then P also records either event* ***begin*** *C3a(sc,sessionId,[Req],n) or event Leak(sc).*
2. *If P records* ***end*** *C4a (sc,sessionId,[Resp Req],n), then P also records either* ***begin*** *C4a (sc,sessionId,[Resp Req],n) or Leak(sc).*
3. *If P outputs* ***secret*** *on a free channel, then P records event* ***begin*** *Leak(sc) and either event* ***begin*** *C3a(sc,sessionId,[Req],n) or event* ***begin*** *C4a(sc,sessionId,[Resp Req],n) where* ***secret*** *is the body of Req or Resp respectively.*

We are now ready to prove the main result.

Proof of Theorem 5.4.2: Property (3) of the theorem follows from Lemma A.2.2(3) and Lemma A.2.4(3). For Properties (1) and (2), we first show that x, n and y, n uniquely index each kind of events **begin** C3n, **end** C3n, **begin** C4n, and **end** C4n recorded in $\mathcal{S}^{\mathcal{N}}$, with n always representing an integer; this follows from the construction of X and Y and the linear communications on channels dc and ds, described in Lemma A.2.1(2,3), that ensure that each event records sessionId and n at most once.

Let E be any evaluation context, with $E[\mathcal{S}^{\mathcal{N}}] \rightarrow^* P$. We define the property $(\mathcal{P}_i)_{i>0}$ as follows:

- $\mathcal{P}_{2n+1}$: if P records event **end** C3n(sc,sessionId,H,n), then P also records either event **begin** C3n(sc,sessionId,H,n) or event **begin** Leak(sc).
- $\mathcal{P}_{2n+2}$: if P records event **end** C4n(sc,sessionId,H,n), then P also records either event **begin** C4n(sc,sessionId,H,n) or event **begin** Leak(sc).

and establish this property by induction on i. For the base case $\mathcal{P}_1$, the process P records event **end** C3n(sc,sessionId,[Req],0). Applying Lemmas A.2.2(1), A.2.4(1), and A.2.2(2), we obtain that P also records either event **begin** C3n(sc,sessionId,[Req],0) or event Leak(sc), establishing the claim.

Suppose now that $\mathcal{P}_m$ holds for $m > 0$, and consider $\mathcal{P}_{m+1}$. We distinguish two cases, with a similar argument—the second case is illustrated on Figure A.1. We use a structural correspondence on $\mathcal{S}^{\mathcal{N}}$ towards the previous begin to match the old parts of the event, a correspondence on $\mathcal{S}^{\mathcal{A}}$ towards the latest begin to match the new parts, the inductive hypothesis to go from the previous end to the previous begin, and finally uniqueness of events at a given index.

- Case $m + 1 = 2n + 1$: P records **end** C3n(sc,sessionId,[Req @ H],succ(n)). By Lemma A.2.2(1), Lemma A.2.4(1), and Lemma A.2.2(2), then either **begin** C3n(sc, sessionId, [Req @ H'], succ(n)) (for some H') or Leak(sc) is also recorded in P. If event Leak(sc) is recorded then we are done, so consider the case in which event **begin** C3n(sc,sessionId,[Req @ H'],succ(n)) is recorded. By Lemma A.2.3(3), we know that **end** C4n(sessionId, H',n) is also recorded. By inductive hypothesis, we

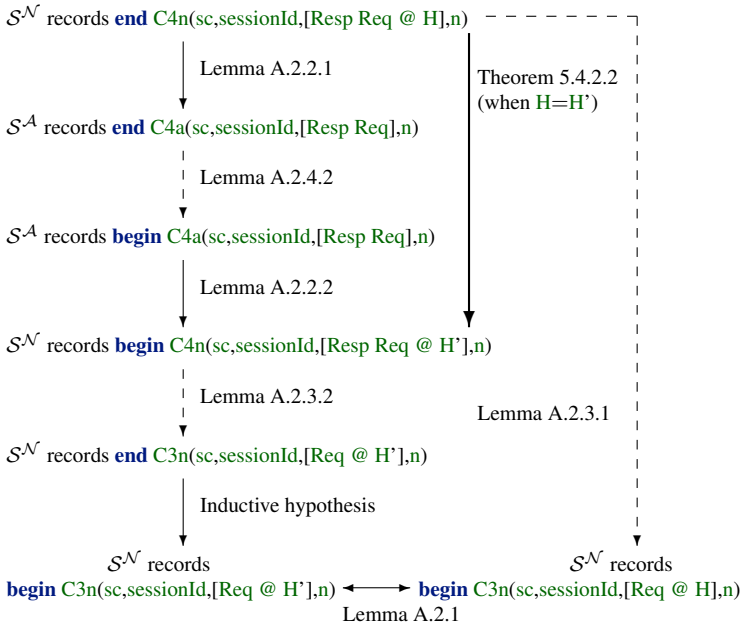


Figure A.1: Outline for the proof of Theorem 5.4.2(2) (bold arrow), using properties proved manually (solid arrows) and automatically (dashed arrows)

obtain **begin** C4n(sc,sessionId,H',n). We can then apply Lemma A.2.3(4) to event **end** C3n(sc,sessionId, [Req @ H],succ(n)) to obtain that event **begin** C4n(sc,sessionId, H,n) is also recorded in P. Since events are indexed by sessionId and n, we obtain that H = H', establishing the claim.

- Case $m + 1 = 2n + 2$: the process P records **end** C4n(sc,sessionId,[Resp Req @ H],n). By Lemma A.2.2(1), Lemma A.2.4(2), and Lemma A.2.2(2), either event **begin** C4n(sc,sessionId, [Resp Req @ H'],n), for some H' or event Leak(sc) is recorded in P. Consider the former case. By Lemma A.2.3(2), the event **end** C3n(sc,sessionId, [Req @ H'],n) is also recorded. By inductive hypothesis, we obtain that the event **begin** C3n(sc,sessionId,[Req @ H'],n). Applying Lemma A.2.3(1) to the recorded event **end** C4n(sc,sessionId,[Resp Req @ H],n), we obtain that the event **begin** C3n(sc ,sessionId,[Req@ H],n) is also recorded in P. Since events are indexed by sessionId and n, we obtain that H = H', hence the property—this case is illustrated as Figure A.1. □

A.3 From Chapter 6

Theorem 6.6.1. *Let* $P, Q \in \mathbf{Pat}$*, such that* $atoms(Q) \not\subseteq \mathcal{B}_Q(P)$*. Then* $[\![P]\!] \approx [\![\mathsf{box}_Q(P)]\!]$.

Proof. Assume the contrary, i.e. there is an algorithm $\mathcal{A}$ that can distinguish the families of probability distributions $[\![P]\!]$ and $[\![\mathsf{box}_Q(P)]\!]$. Fig. A.2 shows an algorithm (first call INI-

TIALIZE and then CONVERT) sampling either $[\![P]\!]$ or $[\![\mathsf{box}_Q(P)]\!]$, depending on the values of the oracles f and g. If f and g are $\mathcal{E}(1^n, k, \cdot)$ and $\mathcal{E}(1^n, k', \cdot)$, then the algorithm in Fig. A.2 samples $[\![P]\!]$. If f and g are both $\mathcal{E}(1^n, k, \mathbf{0})$ then this algorithm samples $[\![\mathsf{box}_Q(P)]\!]$. Composing this algorithm with algorithm $\mathcal{A}$ allows us to break the type-0 security of the encryption system.

We have to show that the algorithm $\text{CONVERT}^{f(\cdot),g(\cdot)}$ can complete its job, i.e. it does not have to access τ at a point where it is undefined. If a key K occurs in P but does not belong to $\mathcal{B}_Q(P)$ then this key only occurs as a subexpression of Q, where Q is used as an encryption key in P. But $\text{CONVERT}^{f(\cdot),g(\cdot)}(1^n, Q, Q)$ is never needed in this context, the oracle f is used instead of encrypting with it. Therefore we do not need the value of K there. Similarly, if $r \in \mathbf{Rnd}$ occurs in P but not in $\mathcal{B}_Q(P)$ then we would need it only for computing the interpretation of the encryption key Q, which is not necessary to compute at all. If r belongs to the set subtracted from $\mathcal{B}_Q(P)$ in the algorithm INITIALIZE, then $\tau(r)$ is not used, but the oracles f or g generate some random numbers of their own.

We have to argue that the algorithm in Figure A.2 indeed samples the claimed families of distributions. Clearly, if f and g are both $\mathcal{E}(1^n, k, \mathbf{0})$, then $\text{CONVERT}^{f(\cdot),g(\cdot)}(1^n, P, Q)$ samples $[\![\mathsf{box}_Q(P)]\!]$. If f is $\mathcal{E}(1^n, k, \cdot)$ and g is $\mathcal{E}(1^n, k', \cdot)$ then we have to show that the key k used by f is indistinguishable from $R(1^{\ell(n)}, \text{CONVERT}(1^n, Q, Q))$ even when we are given the values of τ on $\mathcal{B}_Q(P)$. The key used by f is independent of all the given values of τ. Also, these values of τ do not uniquely determine $\text{CONVERT}^{f,g}(1^n, Q, Q)$ yet, because $atoms(Q) \not\subseteq \mathcal{B}_Q(P)$. Even more, with given values of τ we can guess the value of $\text{CONVERT}^{f,g}(1^n, Q, Q)$ only with negligible success probability. Therefore the application of the random oracle to this value gives us a random bit-string that is independent of the given values of τ. The key used by f and the bit-string $R(1^{\ell(n)}, \text{CONVERT}(1^n, Q, Q))$ are identically distributed, therefore they are indistinguishable under given conditions. Hence we conclude that $\text{CONVERT}^{f(\cdot),g(\cdot)}(1^n, P, Q)$ samples $[\![P]\!]$. □

algorithm INITIALIZE$(1^n, P, Q)$
  for all $K \in \mathcal{B}_Q(P)$ do $\tau(K) \leftarrow \mathcal{G}(1^n)$
  for all r in the set
    $\mathcal{B}_Q(P) \backslash \big(\{ r' \in \mathbf{Rnd} \,|\, \Box^{r'} \text{ occurs in } P \} \cup \{ r' \in \mathbf{Rnd} \,|\, \{\cdot\}_Q^{r'} \text{ occurs in } P \} \big)$
      do $\tau(r) \stackrel{R}{\in} \{0,1\}^*$

algorithm CONVERT$^{f(\cdot),g(\cdot)}(1^n, P, Q)$
  if CONVERT$^{f(\cdot),g(\cdot)}(1^n, P, Q)$ has been invoked before
    return the value returned previously
  if P is $K \in \mathbf{Keys}$
    return $\langle \tau(K), \text{“key”} \rangle$
  else if P is $b \in \mathbf{Bool}$
    return $\langle b, \text{“bit”} \rangle$
  else if P is (P_1, P_2)
    let $x =$ CONVERT$^{f,g}(1^n, P_1, Q)$
    let $y =$ CONVERT$^{f,g}(1^n, P_2, Q)$
    return $\langle x, y, \text{“pair”} \rangle$
  else if P is $\{P_2\}^r_{P_1}$
    if $P_1 = Q$
      let $y =$ CONVERT$^{f,g}(1^n, P_2, Q)$
      let $z \leftarrow f(y)$
    else
      let $x =$ CONVERT$^{f,g}(1^n, P_1, Q)$
      let $y =$ CONVERT$^{f,g}(1^n, P_2, Q)$
      let $z = \mathcal{E}^{\tau(r)}(1^n, R(1^{\ell(n)}, x), y)$
    return $\langle z, \text{“ciphertext”} \rangle$
  else: P is $\Box^r$
    let $z \leftarrow g(\mathbf{0})$
    return $\langle z, \text{“ciphertext”} \rangle$

Figure A.2: Algorithm sampling either $[\![P]\!]$ or $[\![\mathsf{box}_Q(P)]\!]$

Samenvatting

Communicatie over een gedeeld medium, zoals bijvoorbeeld het Internet, is inherent onveilig: Iedereen heeft toegang tot berichten, kan de routing beïnvloeden en kan het communicatie verkeer potentieel afluisteren of zelfs veranderen. Beveiligings protocollen zijn gedistribueerde programmas die specifiek ontworpen zijn om veilige communicatie over zulk een gedeeld medium mogelijk te maken, waarbij uitgewisselde berichten typisch gebouwd zijn met cryptografische operaties (zoals bericht versleuteling).

Omdat het correct ontwerpen van beveiligings protocollen moeilijk is, is de analyse van deze protocollen essentieel. Een buitengewoon succesvol model voor de analyse van beveiligings protocollen is het zogenaamde Dolev Yao model, waarin wordt aangenomen dat de aanvaller complete controle heeft over het netwerk. Het aanvallers model gaat uit van 'perfecte cryptografie' i.e. van alle cryptografische operaties wordt aangenomen dat ze onbreekbaar zijn. Het Dolev Yao model is aantrekkelijk omdat het eenvoudig geformaliseerd kan worden met op formele methoden gebaseerde talen en gereedschappen. Bovendien heeft het model het juiste abstractie niveau aangezien veel aanvallen onafhankelijk zijn van de details van de cryptografische operaties omdat ze alleen gebaseerd zijn op combinaties van uitgewisselde berichten en de kennis die de aanvaller verzameld tijdens de executie van het protocol.

In dit proefschrift worden vijf significante en ortogonale uitbreidingen van het Dolev Yao model beschreven. Elk van deze uitbreidingen geeft een realistischere benadering van een aspect van de werkelijkheid, en geeft hierdoor een sterkere veiligheids garantie. De uitbreidingen zijn:

1. We stellen een efficiënte beslissings procedure voor gebaseerd op "constraint solving" welke een verbetering op eerder werk door Millen en Shmatikov geeft. We introduceren vervolgens met een rijke taal voor beveiligings eigenschappen gebaseerd op lineaire temporele logica en presenteren een procedure voor het checken van gokaanvallen ("guessing attacks"). We gebruiken de procedure als een didactisch hulpmiddel en voor de analyse van voorbeeld studies.

2. We ontwikkelen een model waarin de voortgang van tijd tijdens executie van een protocol expliciet wordt gemaakt met behulp van geklokte automaten. Dit maakt het mogelijk om tijd afhankelijke punten zoals time-outs en herversturing in beveiligings protocol implementaties te bestuderen.

3. We gebruiken de Applied Pi Calculus om gok-aanvallen te bestuderen onder de aanname van een versterkte Dolev Yao aanvaller die cryptografische relaties binnen afgeluisterde berichten kan benutten.

4. We benutten de TulaFale taal om sessie gebaseerde web service beveiligings protocollen te bestuderen. We formalizeren en geven een semantiek aan twee industriele specificaties namelijk WS-Trust en WS-SecureConversation.

5. We relateren het Dolev Yao model met een realistisch compationeel model. In het bijzonder breiden we het werk van Abadi en Rogaway uit met encryptie die gebruikt maakt van samengestelde sleutels in plaats van atomaire sleutels.

Titles in the IPA Dissertation Series

J.O. Blanco. *The State Operator in Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1996-01

A.M. Geerling. *Transformational Development of Data-Parallel Algorithms*. Faculty of Mathematics and Computer Science, KUN. 1996-02

P.M. Achten. *Interactive Functional Programs: Models, Methods, and Implementation*. Faculty of Mathematics and Computer Science, KUN. 1996-03

M.G.A. Verhoeven. *Parallel Local Search*. Faculty of Mathematics and Computing Science, TUE. 1996-04

M.H.G.K. Kesseler. *The Implementation of Functional Languages on Parallel Machines with Distrib. Memory*. Faculty of Mathematics and Computer Science, KUN. 1996-05

D. Alstein. *Distributed Algorithms for Hard Real-Time Systems*. Faculty of Mathematics and Computing Science, TUE. 1996-06

J.H. Hoepman. *Communication, Synchronization, and Fault-Tolerance*. Faculty of Mathematics and Computer Science, UvA. 1996-07

H. Doornbos. *Reductivity Arguments and Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1996-08

D. Turi. *Functorial Operational Semantics and its Denotational Dual*. Faculty of Mathematics and Computer Science, VUA. 1996-09

A.M.G. Peeters. *Single-Rail Handshake Circuits*. Faculty of Mathematics and Computing Science, TUE. 1996-10

N.W.A. Arends. *A Systems Engineering Specification Formalism*. Faculty of Mechanical Engineering, TUE. 1996-11

P. Severi de Santiago. *Normalisation in Lambda Calculus and its Relation to Type Inference*. Faculty of Mathematics and Computing Science, TUE. 1996-12

D.R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. Faculty of Mathematics and Computing Science, TUE. 1996-13

M.M. Bonsangue. *Topological Dualities in Semantics*. Faculty of Mathematics and Computer Science, VUA. 1996-14

B.L.E. de Fluiter. *Algorithms for Graphs of Small Treewidth*. Faculty of Mathematics and Computer Science, UU. 1997-01

W.T.M. Kars. *Process-algebraic Transformations in Context*. Faculty of Computer Science, UT. 1997-02

P.F. Hoogendijk. *A Generic Theory of Data Types*. Faculty of Mathematics and Computing Science, TUE. 1997-03

T.D.L. Laan. *The Evolution of Type Theory in Logic and Mathematics*. Faculty of Mathematics and Computing Science, TUE. 1997-04

C.J. Bloo. *Preservation of Termination for Explicit Substitution*. Faculty of Mathematics and Computing Science, TUE. 1997-05

J.J. Vereijken. *Discrete-Time Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1997-06

F.A.M. van den Beuken. *A Functional Approach to Syntax and Typing*. Faculty of Mathematics and Informatics, KUN. 1997-07

A.W. Heerink. *Ins and Outs in Refusal Testing*. Faculty of Computer Science, UT. 1998-01

G. Naumoski and W. Alberts. *A Discrete-Event Simulator for Systems Engineering*. Faculty of Mechanical Engineering, TUE. 1998-02

J. Verriet. *Scheduling with Communication for Multiprocessor Computation*. Faculty of Mathematics and Computer Science, UU. 1998-03

J.S.H. van Gageldonk. *An Asynchronous Low-Power 80C51 Microcontroller*. Faculty of Mathematics and Computing Science, TUE. 1998-04

A.A. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1998-05

E. Voermans. *Inductive Datatypes with Laws and Subtyping – A Relational Model*. Faculty of Mathematics and Computing Science, TUE. 1999-01

H. ter Doest. *Towards Probabilistic Unification-based Parsing*. Faculty of Computer Science, UT. 1999-02

J.P.L. Segers. *Algorithms for the Simulation of Surface Processes*. Faculty of Mathematics and Computing Science, TUE. 1999-03

C.H.M. van Kemenade. *Recombinative Evolutionary Search*. Faculty of Mathematics and Natural Sciences, UL. 1999-04

E.I. Barakova. *Learning Reliability: a Study on Indecisiveness in Sample Selection*. Faculty of Mathematics and Natural Sciences, RUG. 1999-05

M.P. Bodlaender. *Scheduler Optimization in Real-Time Distributed Databases*. Faculty of Mathematics and Computing Science, TUE. 1999-06

M.A. Reniers. *Message Sequence Chart: Syntax and Semantics*. Faculty of Mathematics and Computing Science, TUE. 1999-07

J.P. Warners. *Nonlinear approaches to satisfiability problems*. Faculty of Mathematics and Computing Science, TUE. 1999-08

J.M.T. Romijn. *Analysing Industrial Protocols with Formal Methods*. Faculty of Computer Science, UT. 1999-09

P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. Faculty of Computer Science, UT. 1999-10

G. Fábián. *A Language and Simulator for Hybrid Systems*. Faculty of Mechanical Engineering, TUE. 1999-11

J. Zwanenburg. *Object-Oriented Concepts and Proof Rules*. Faculty of Mathematics and Computing Science, TUE. 1999-12

R.S. Venema. *Aspects of an Integrated Neural Prediction System*. Faculty of Mathematics and Natural Sciences, RUG. 1999-13

J. Saraiva. *A Purely Functional Implementation of Attribute Grammars*. Faculty of Mathematics and Computer Science, UU. 1999-14

R. Schiefer. *Viper, A Visualisation Tool for Parallel Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1999-15

K.M.M. de Leeuw. *Cryptology and Statecraft in the Dutch Republic*. Faculty of Mathematics and Computer Science, UvA. 2000-01

T.E.J. Vos. *UNITY in Diversity. A stratified approach to the verification of distributed algorithms*. Faculty of Mathematics and Computer Science, UU. 2000-02

W. Mallon. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes*. Faculty of Mathematics and Natural Sciences, RUG. 2000-03

W.O.D. Griffioen. *Studies in Computer Aided Verification of Protocols*. Faculty of Science, KUN. 2000-04

P.H.F.M. Verhoeven. *The Design of the MathSpad Editor*. Faculty of Mathematics and Computing Science, TUE. 2000-05

J. Fey. *Design of a Fruit Juice Blending and Packaging Plant*. Faculty of Mechanical Engineering, TUE. 2000-06

M. Franssen. *Cocktail: A Tool for Deriving Correct Programs*. Faculty of Mathematics and Computing Science, TUE. 2000-07

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

E. Saaman. *Another Formal Specification Language*. Faculty of Mathematics and Natural Sciences, RUG. 2000-10

M. Jelasity. *The Shape of Evolutionary Search Discovering and Representing Search Space Structure*. Faculty of Mathematics and Natural Sciences, UL. 2001-01

R. Ahn. *Agents, Objects and Events a computational approach to knowledge, observation and communication*. Faculty of Mathematics and Computing Science, TU/e. 2001-02

M. Huisman. *Reasoning about Java programs in higher order logic using PVS and Isabelle*. Faculty of Science, KUN. 2001-03

I.M.M.J. Reymen. *Improving Design Processes through Structured Reflection*. Faculty of Mathematics and Computing Science, TU/e. 2001-04

S.C.C. Blom. *Term Graph Rewriting: syntax and semantics*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2001-05

R. van Liere. *Studies in Interactive Visualization*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2001-06

A.G. Engels. *Languages for Analysis and Testing of Event Sequences*. Faculty of Mathematics and Computing Science, TU/e. 2001-07

J. Hage. *Structural Aspects of Switching Classes*. Faculty of Mathematics and Natural Sciences, UL. 2001-08

M.H. Lamers. *Neural Networks for Analysis of Data in Environmental Epidemiology: A Case-study into Acute Effects of Air Pollution Episodes*. Faculty of Mathematics and Natural Sciences, UL. 2001-09

T.C. Ruys. *Towards Effective Model Checking*. Faculty of Computer Science, UT. 2001-10

D. Chkliaev. *Mechanical verification of concurrency control and recovery protocols*. Faculty of Mathematics and Computing Science, TU/e. 2001-11

M.D. Oostdijk. *Generation and presentation of formal mathematical documents*. Faculty of Mathematics and Computing Science, TU/e. 2001-12

A.T. Hofkamp. *Reactive machine control: A simulation approach using χ*. Faculty of Mechanical Engineering, TU/e. 2001-13

D. Bošnački. *Enhancing state space reduction techniques for model checking*. Faculty of Mathematics and Computing Science, TU/e. 2001-14

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects*. Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity*. Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing*. Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing*. Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining*. Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL*. Faculty of Mathematics and Computer Science, TU/e. 2002-16

J.J.D. Aerts. *Random Redundant Storage for Video on Demand*. Faculty of Mathematics and Computer Science, TU/e. 2003-01

M. de Jonge. *To Reuse or To Be Reused: Techniques for component composition and construction*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-02

J.M.W. Visser. *Generic Traversal over Typed Source Code Representations*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-03

S.M. Bohte. *Spiking Neural Networks*. Faculty of Mathematics and Natural Sciences, UL. 2003-04

T.A.C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing*. Faculty of Mathematics and Computer Science, TU/e. 2003-05

S.V. Nedea. *Analysis and Simulations of Catalytic Reactions*. Faculty of Mathematics and Computer Science, TU/e. 2003-06

M.E.M. Lijding. *Real-time Scheduling of Tertiary Storage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-07

H.P. Benz. *Casual Multimedia Process Annotation – CoMPAs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08

D. Distefano. *On Modelchecking the Dynamics of Object-based Software: a Foundational Approach*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-09

M.H. ter Beek. *Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components*. Faculty of Mathematics and Natural Sciences, UL. 2003-10

D.J.P. Leijen. *The λ Abroad – A Functional Approach to Software Components*. Faculty of Mathematics and Computer Science, UU. 2003-11

W.P.A.J. Michiels. *Performance Ratios for the Differencing Method*. Faculty of Mathematics and Computer Science, TU/e. 2004-01

G.I. Jojgov. *Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving*. Faculty of Mathematics and Computer Science, TU/e. 2004-02

P. Frisco. *Theory of Molecular Computing – Splicing and Membrane systems*. Faculty of Mathematics and Natural Sciences, UL. 2004-03

S. Maneth. *Models of Tree Translation*. Faculty of Mathematics and Natural Sciences, UL. 2004-04

Y. Qian. *Data Synchronization and Browsing for Home Environments*. Faculty of Mathematics and Computer Science and Faculty of Industrial Design, TU/e. 2004-05

F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-06

L. Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications*. Faculty of Science, Mathematics and Computer Science, KUN. 2004-07

E.H. Gerding. *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications*. Faculty of Technology Management, TU/e. 2004-08

N. Goga. *Control and Selection Techniques for the Automated Testing of Reactive Systems*. Faculty of Mathematics and Computer Science, TU/e. 2004-09

M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs*. Faculty of Science, Mathematics and Computer Science, RU. 2004-10

A. Löh. *Exploring Generic Haskell*. Faculty of Mathematics and Computer Science, UU. 2004-11

I.C.M. Flinsenberg. *Route Planning Algorithms for Car Navigation*. Faculty of Mathematics and Computer Science, TU/e. 2004-12

R.J. Bril. *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets*. Faculty of Mathematics and Computer Science, TU/e. 2004-13

J. Pang. *Formal Verification of Distributed Systems*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-14

F. Alkemade. *Evolutionary Agent-Based Economics*. Faculty of Technology Management, TU/e. 2004-15

E.O. Dijk. *Indoor Ultrasonic Position Estimation Using a Single Base Station*. Faculty of Mathematics and Computer Science, TU/e. 2004-16

S.M. Orzan. *On Distributed Verification and Verified Distribution*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-17

M.M. Schrage. *Proxima - A Presentation-oriented Editor for Structured Documents*. Faculty of Mathematics and Computer Science, UU. 2004-18

E. Eskenazi and A. Fyukov. *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures*. Faculty of Mathematics and Computer Science, TU/e. 2004-19

P.J.L. Cuijpers. *Hybrid Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2004-20

N.J.M. van den Nieuwelaar. *Supervisory Machine Control by Predictive-Reactive Scheduling*. Faculty of Mechanical Engineering, TU/e. 2004-21

E. Ábrahám. *An Assertional Proof System for Multithreaded Java -Theory and Tool Support-* . Faculty of Mathematics and Natural Sciences, UL. 2005-01

R. Ruimerman. *Modeling and Remodeling in Bone Tissue*. Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. *Experiments in Rights Control - Expression and Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

H. Gao. *Design and Verification of Lock-free Parallel Algorithms*. Faculty of Mathematics and Computing Sciences, RUG. 2005-04

H.M.A. van Beek. *Specification and Analysis of Internet Applications*. Faculty of Mathematics and Computer Science, TU/e. 2005-05

M.T. Ionita. *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures*. Faculty of Mathematics and Computing Sciences, TU/e. 2005-06

G. Lenzini. *Integration of Analysis Techniques in Security and Fault-Tolerance*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-07

I. Kurtev. *Adaptability of Model Transformations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-08

T. Wolle. *Computational Aspects of Treewidth - Lower Bounds and Network Reliability*. Faculty of Science, UU. 2005-09

O. Tveretina. *Decision Procedures for Equality Logic with Uninterpreted Functions*. Faculty of Mathematics and Computer Science, TU/e. 2005-10

A.M.L. Liekens. *Evolution of Finite Populations in Dynamic Environments*. Faculty of Biomedical Engineering, TU/e. 2005-11

J. Eggermont. *Data Mining using Genetic Programming: Classification and Symbolic Regression*. Faculty of Mathematics and Natural Sciences, UL. 2005-12

B.J. Heeren. *Top Quality Type Error Messages*. Faculty of Science, UU. 2005-13

G.F. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*. Faculty of Science, Mathematics and Computer Science, RU. 2005-14

M.R. Mousavi. *Structuring Structural Operational Semantics*. Faculty of Mathematics and Computer Science, TU/e. 2005-15

A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems*. Faculty of Mathematics and Computer Science, TU/e. 2005-16

T. Gelsema. *Effective Models for the Structure of pi-Calculus Processes with Replication*. Faculty of Mathematics and Natural Sciences, UL. 2005-17

P. Zoeteweij. *Composing Constraint Solvers*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-18

J.J. Vinju. *Analysis and Transformation of Source Code by Parsing and Rewriting*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-19

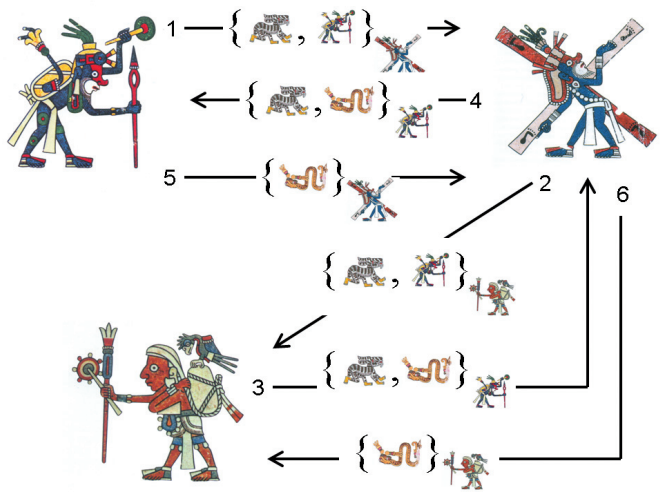
M.Valero Espada. *Modal Abstraction and Replication of Processes with Data*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2005-20

A. Dijkstra. *Stepping through Haskell*. Faculty of Science, UU. 2005-21

Y.W. Law. *Key management and link-layer security of wireless sensor networks: energy-efficient attack and defense*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-22

E. Dolstra. *The Purely Functional Software Deployment Model*. Faculty of Science, UU. 2006-01

R.J. Corin. *Analysis Models for Security Protocols*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-02


1
4
5
2
6
3