PhD-FSTM-2020-38
The Faculty of Sciences, Technology and Medicine

DISSERTATION

Defense held on 10/09/2020 in Esch-sur-Alzette

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Zach SMITH

Born on 22 April 1990 in Peterborough, (United Kingdom)

Design and Verification of Specialised Security Goals for Protocol Families

Dissertation defense committee

Dr Sjouke Mauw, dissertation supervisor
Professor, Université du Luxembourg

Dr Peter Y A Ryan, Chair
Professor, Université du Luxembourg

Dr Rolando Trujillo-Rasúa, Vice Chair
Deakin University

Dr Cas Cremers
Professor, CISPA Heimholtz Center for Information Security

Dr Steve Kremer
Inria Nancy

"The case for my life, then, or for that of any one else who has been a mathematician in the same sense in which I have been one, is this: that I have added something to knowledge, and helped others to add more; and that these somethings have a value which differs in degree only, and not in kind, from that of the creations of the great mathematicians or of any of the other artists, great or small, who have left some kind of memorial behind them."

G. H. Hardy

Abstract

Design and Verification of Specialised Security Goals for Protocol Families

by Zach SMITH

Communication Protocols form a fundamental backbone of our modern information networks. These protocols provide a framework to describe how agents - Computers, Smartphones, RFID Tags and more - should structure their communication. As a result, the security of these protocols is implicitly trusted to protect our personal data.

In 1997, Lowe presented 'A Hierarchy of Authentication Specifications', formalising a set of security requirements that might be expected of communication protocols. The value of these requirements is that they can be formally tested and verified against a protocol specification. This allows a user to have confidence that their communications are protected in ways that are uniformly defined and universally agreed upon.

Since that time, the range of objectives and applications of real-world protocols has grown. Novel requirements - such as checking the physical distance between participants, or evolving trust assumptions of intermediate nodes on the network - mean that new attack vectors are found on a frequent basis. The challenge, then, is to define security goals which will guarantee security, *even when the nature of these attacks is not known*.

In this thesis, a methodology for the design of security goals is created. It is used to define a collection of specialised security goals for protocols in multiple different families, by considering tailor-made models for these specific scenarios. For complex requirements, theorems are proved that simplify analysis, allowing the verification of security goals to be efficiently modelled in automated prover tools.

Acknowledgements

The acknowledgements section of this thesis could easily take up more space than the thesis itself, were I to give everyone the thanks that they truly deserve.

Thanks to Sjouke, my supervisor, for supporting me these years. I have learned a lot from him, and some of it related to research, too. His advice and patience as I fumbled my way through the various problems faced during my stay is greatly appreciated.

Thanks to the people who worked closely with me. Most notable is Jorge, my officemate for a majority of my PhD. I think my most productive days were spent with him, discussing the details of our research. Thanks also to Hyunwoo and Taekyoung Kwon at Seoul National University, who took a chance inviting me to work with them after I stumbled into their office, searching for a desk to work at as I raced to meet a deadline while travelling (the submission in question was later rejected). Finally, thanks to Ioana, Steve and Helen at the University of Surrey. One of the best parts of my PhD has been feeling what it is like to work in multiple different roles in these different groups, and I will treasure this experience greatly. Thanks also to my other coathors - Rolando, Hugo, Ross, Ihor, Selin, Junghwan, Gyeongjae and Taejoong.

Thanks to the members of the SaToSS group at the University of Luxembourg, especially the other PhDs, with whom I have formed some close friendships. I am fortunate to have been in a successful group that continues to grow, to the point where the list of names is somewhat extensive. Thanks in particular to Cui and Ninghan, who helped distract me by helping me learn Mandarin Chinese, and to Olga, Alex and Stas, who have given me useful support and advice.

Thanks to my advisory team - Cas Cremers and Rolando Trujillo Rasua - who gave me assistance and advice throughout my research. Their guidance in starting my journey in research was of great help.

Finally, thanks to my mom and my friends, who have given me the support I needed to make it through these years.

Contents

List of Figures

List of Tables

Layout Notes

Where relevant, boxes such as this one give some context about the research papers written over the course of preparing this thesis. They include information about my personal contributions to the conference publications which form the foundation of each chapter.

This information may be useful to reviewers of the thesis, but may not be important for other readers.

Key Statements

At certain locations in this document, key definitions, statements or results will be emphasised in boxes such as this one.

Code Snippets

```
Important code examples are provided in code boxes.

Usually these are references to Tamarin code, although
several python and bash utilities were developed in the
course of writing this thesis.
```

Some expressions, tables, formulae are presented in boxes such as this one, to separate them from the text body.

Chapter 1

Introduction

1.1 Security Protocols

Electronic communications form the backbone of our global networks. These communications are in turn made up of *protocols*: standardised instructions for how messages should be constructed, packaged and sent.

The nature of communication networks means that they are constantly at risk of adversarial attacks. An attacker might be able to attack the integrity of the cryptographic devices used, or take advantage of minor implementation errors. In many cases, the consequences of such attacks can be severe - for example, the Heartbleed exploit [DLK+14] manipulated a message-handling bug in SSL to allow for a malicious user to read sensitive data on websites including Google, Netflix and Wikipedia.

However, even when we ignore such possibilities, there is still a very large margin for error. That is, even if we assume that cryptography is perfect, and that agents always perfectly follow the instructions given to them, it is still not trivial to design protocols that allow for effective communication while guaranteeing security.

As such, the design and verification of security protocols has been an active area of research for over 40 years. The field is continually developing, in order to consider new potential attack vectors and precisely define the requirements to be placed on different communication systems.

One such domain of interest is in RFID (Radio Frequency Identification) devices. RFID tags are used in billions of devices worldwide, both for consumer-grade and industrial applications. Their uses are pervasive in everyday life: appearing in payment cards, key fobs, and even electronic passports. Meanwhile, on the industrial side, RFID devices provide a convenient and cheap way to aid the logistics process.

RFID Tags also serve a role in the greater “Internet of Things”. Billions of small-scale sensors, control devices and automated systems are connected to global networks to achieve a variety of goals. Low-power devices such as these present new challenges to protocol designers, as they often have reduced cryptographic capabilities and rely on radio signals, which can be easily intercepted or disrupted.

The growing scale of the internet also presents new problems in protocol design. Internet communication (such as the TLS protocol over the World Wide Web) is often idealised as a two-party setting, where a client and server pass encrypted packets between each other. However, in modern systems, such communications are in fact dependent on a series of intermediate agents - proxies responsible for establishing connections with servers around the world, or network appliances that read or modify content in order to grant new functionality to users. In these situations, a designer must consider ways to allow for such advancements without weakening the security of the overall process.

1.2 Security Goals

In order to guarantee that a communication protocol can be both effective and secure, formal methods present several approaches. By considering a protocol - and the underlying environment - as a mathematical construction, we can encode not only its structure but also its intended goals.

Such approaches are naturally limited by the creativity of the mathematical interpretation. The most famous (and oft-repeated) example is that of the protocol of Needham and Schroeder [NS78], who provided a proof that their design was secure under a set of assumptions. This proof stood for over 15 years until Lowe [Low96] demonstrated that a more reasonable set of assumptions lead to a novel vulnerability.

As a result, the verification community has generally moved towards attempts at standardisation - establishing foundations from which trustworthy analysis can be performed. This manifests itself in three main forms:

- A choice of language for describing the intended execution of a protocol
- A choice of *metalanguage*, which describes the security goals of protocols
- An encoding of the capabilities of the adversary seeking to undermine these goals

There are many influences behind the different models created. These generally form a tradeoff between several factors:

- Faithfully modelling real implementations of protocols
- Producing languages which can be easily written, interpreted, and reasoned about
- Integration with computer-aided proof techniques

To this end, there are two main approaches in the domain of formal protocol analysis. The *computational* approach, likely originating from techniques used to model security of cryptographic primitives, attempts to show that an attacker can violate some security goals with only negligible probability. This is often done by comparing with an ideal functionality [Gol98], or by “game-hopping”: demonstrating that breaking the protocol necessarily entails a solution to a hard math problem such as integer factorisation or the discrete logarithm [Sho04]. To this end, computational approaches involve modelling specific parameters, such as the length of bitstrings. By comparison, in the *symbolic* approach, terms are fully abstracted, and the existence of attacks is seen as a strict binary possibility (excluding uncertainty). Despite this increased coarseness, the symbolic model is still extremely powerful, and has been used to demonstrate vulnerabilities of multiple real-world protocols such as TLS 1.3 [CHH+17] and the 5G infrastructure [BDH+18]. It is for this reason that symbolic models are the focus of this research.

A notable landmark this line of research is that of Lowe in 1997 [Low97], in which protocols are modelled using a process algebra known as CSP [Hoa78]. The use of process algebra to model protocols has since developed greatly thanks to specialised languages such as the Applied Pi Calculus [RS11]. Meanwhile, alternative approaches such as those used to reason about Strand Spaces [FJHG99] have been generalised and applied to more expressive settings, such as Multiset Rewriting Theory [CDM+00].

Despite the multiple models available, the security verification community usually accepts the Dolev-Yao adversary [DY83] as the de-facto standard. The Dolev-Yao adversary is assumed to have full control over the communication network - able to intercept, redirect or modify messages. In most cases, the adversary is also assumed to be able to *corrupt* agents, gaining full access to their encryption keys. There are several reasons why this adversarial model is so commonly used. These include its relative strength, as well as being able to break down its capabilities into a concise set of rules (that also describe common network environments), with limited to no special cases. Alongside this adversarial model, most symbolic models avoid probability-related problems by making the *Perfect Cryptography Assumption*: that the adversary is unable to perform or undo encryption without having the associated keys, with similar assumptions for other primitives such as hash functions.

Categorising Security Goals

A common theme in all of these modelling scenarios is the use of consistent security goals - even between different settings. Security goals can be broadly categorised as being either *privacy* or *authentication* goals.

Authentication claims refer to an agent's belief about how a protocol has been executed. *Aliveness*, generally the weakest of such goals, is the notion that the intended communication partner is (or was) present at some point during the execution. *Agreement* represents the idea that the two parties both have the same view of specific terms (for example, the encryption key generated at the end of a key exchange protocol). Finally, *synchronisation* goals, first introduced by Cremers et al. [CMdV06], refine agreement further by requiring that messages were indeed exchanged at the correct times and in the correct order. This distinction is significant when considering some classes of attacks (most notably, preplay attacks).

Privacy goals relate to the adversary's ability to learn specific information, or to differentiate between terms. For example, *vote privacy* refers to the attacker's ability to deduce how agents have voted in an election, whilst *untraceability* describes the ability to detect if the same (or different) agents have performed two disjoint executions of the same protocol. These goals are generally verified by examining if an attacker can differentiate between two slightly different scenarios.

Security Goals for Advanced Adversaries

The Dolev-Yao adversary is traditionally considered to be the strongest possible traditional adversary [Cer01]. However, much consideration has been put into attacker models that are stronger still. These are often formed by weakening the perfect cryptography assumption, considering the capability of the adversary to exploit side effects or eventually break cryptographic primitives.

Perhaps the first of such security goals was *forward secrecy* [Gün89], which refers to the idea that the session data of a protocol should be secure even if a long-term key is compromised. Cremers and Basin [BC10] proved that forward secrecy against the traditional Dolev-Yao adversary is equivalent to normal secrecy against an enhanced adversary (who can corrupt secret keys outside of a specific 'test' run).

Since then, several similar security goals have been investigated - following the theme of guaranteeing traditional authentication goals against empowered adversaries. This topic is becoming increasingly relevant with the advent of *post-quantum systems* [Ber09], where devices such as Shor's algorithm [Sho94] suggest that many cryptographic primitives that we rely upon will eventually become broken. Such avenues of research include post-compromise security [CGCG16], which discusses the potential for future security even after a secret key is revealed, as well as post-breakdown security [BFG19], which generalises this idea to all cryptographic primitives (including, for example, hashing).

Instead of changing assumptions about the security of cryptographic primitives, another approach considers changing assumptions about the honesty of agents. The Dolev-Yao adversarial model typically assumes that agents are either fully honest - following their specifications perfectly, or otherwise corrupt, completely under adversarial control. However, such assumptions may not be accurate. Multiparty computation schemes often consider the notion of "honest-but-curious" [KS05] agents, who will not disrupt the execution of a protocol but may attempt to misbehave in order to learn more information than intended. Even Shamir's Secret Sharing [Sha79], a seminal work from 1979, admits an attack in which several agents may collaborate to break the integrity of the system [TW89]. A common problem with describing these kinds of vulnerabilities is in dealing with ways to correctly formalise the security definitions, as well as enumerating the possibilities of how a protocol could go wrong.

Security Goals for Advanced Protocols

Despite these various adversarial models, the underlying authentication goals that system designers seek to verify are usually the same. This is in stark contrast to privacy goals - where minor changes (either in the semantics of the protocol execution model, or in the precise wording of the security goal) can give highly varying results [DKR07]. However, as the scope of security protocols grows, we are starting to see that the limited set of definitions available may not be fully sufficient.

One such domain where traditional authentication goals fail is in *distance bounding protocols*. These protocols, originally introduced by Brands and Chaum [BC93], are designed to guarantee the physical proximity of a *Prover* to a *Verifier*. This is important for devices such as payment cards or electronic keys, where an attacker must not be able to simply relay radio signals. These protocols are also becoming increasingly relevant with the advent of smart vehicles and the so-called convoy problem [LB06], where a self-controlling vehicle must choose who to receive positioning data from. Models of these protocols must consider not only traditional authentication requirements, but also a way of describing the locations of agents.

Many authentication-style properties also implicitly assume *statelessness* of the underlying system. This means that each execution of the protocol can be seen as a blank slate, using only long-term unchanging keys or freshly generated values. However, in order to achieve goals such as forward secrecy, a common practice is to use stateful data - such as updating counters or keys. Stateful protocols present new risks, in dealing with situations in which shared data could become mismatched or reset. Such scenarios are referred to as *desynchronisation attacks*. Desynchronisation resistance is a liveness property: it is required that the protocol can always eventually successfully continue, regardless of the actions of the adversary.

Finally, many protocols are traditionally considered to be two-party processes. However, modern communication networks are often more intricate. Intermediate agents may exist, either as *proxies* who are responsible for forwarding messages to the correct parties, or as *middleboxes*, who might read or modify messages in order to add functionality to a system. As such, an endpoint may have security requirements beyond those regarding the correspondence with their final partner. For example, they may wish to ensure that intermediate agents such as firewalls or content filters are not inadvertently skipped past. Further, the intermediate agents themselves may have some security requirements - this is most notable in *payment networks* such as the Bitcoin Lightning Network [PD16], in which a party may only be willing to forward a payment if they can be confident that they will receive their share of the transaction fee.

Automated Verification

A key advantage of many modelling approaches is the assistance of automated proving tools. Such tools allow for increased confidence in the thoroughness of analysis, and allow for rapid verification, often without significant user interaction.

Originally, general-purpose model checkers such as FDR (Failures-Divergences Refinement) [Ros94] were adapted to support the requirements of security protocols. However, such approaches often run into problems in modelling the network adversary. Since the adversary traditionally has the ability to intercept and inject messages, as well as the ability to gain and infer knowledge, there is a large risk of a state space explosion.

As a result, a range of modern tools have been developed, specifically designed for handling security protocols. These tools are built upon the same powerful solving techniques, but with additional considerations in order to improve the efficiency of analysis.

Although the use of tool support can make life significantly easier for analysing a range of protocols, it can also impose additional restrictions on the user. We are constrained by the language supported by the tool, both in modelling the protocol and in any associated security claims.

Especially in the symbolic setting, a significant concern is in over-abstracting a protocol's description, or in failing to model the algebraic properties of the primitives used. For example, the Scyther [Cre08] tool supports only a fixed equational theory. Much effort has been put into adding faithful representation of operators such as exclusive-or [DHRS18] into verification tools, and modern work demonstrates that even some of our assumptions about how cryptographic primitives such as signature schemes operate can be innacurate [JCCGS19].

Often these restrictions are a necessary measure in order to ensure that the tool behaves well. The ideal functionality of a verifier is that it should be *sound*: it will not report incorrectly report attacks - as well as *complete*: if there is an attack, the tool will find it. A corollary of the Halting problem [BM82] tells us that no proof tool can be simultaneously *sound, complete* and *terminating*. That is, in order to guarantee that a tool will terminate on all inputs, the designer must be willing either for it to report false vulnerabilities, or for it to miss some attacks.

Different tools handle this problem in different ways. ProVerif [Bla01], a tool based on the Applied Pi Calculus, overapproximates security goals, meaning that it sometimes cannot be sure if an attack is legitimate. Many tools are only able to consider bounded environments (e.g. a maximum number of sessions or messages), especially those focusing on privacy goals, such as DeepSec [CKR18]. This can affect completeness - indeed, a result of Millen shows that for any n it is possible to construct a protocol that is secure for n sessions and insecure for $n + 1$. The Tamarin Prover [MSCB13] instead opts to ensure both soundness and completeness by sacrificing termination - meaning that results can be trusted, but a user has no guarantee that the analysis of their protocol will ever finish.

In many of these cases it is the responsibility of the person modelling a protocol to justify the implicit assumptions made in this translation. As such, a focus of this work is to evaluate *families* of protocols, rather than individual protocols at a time. By building a common framework within which to perform analysis, we can be sure that protocols are handled equally, under a shared set of assumptions.

1.3 Research Questions

The relatively recent wave of verification tools has granted opportunities for ground-breaking progress in the symbolic analysis of security protocols. These new tools allow for symbolic implementations of protocols which follow the original specification more closely than was previously possible.

However, although we can model the behaviour of these protocols, it is not always immediately clear how to define their intended security goals. A common approach is to look at historical attacks on related protocols and attempt to actively describe these. However, this can lead to scenarios in which novel attacks are missed - a good example of this is the "discovery" of the Distance Hijacking class of attacks on Distance Bounding protocols [CRSC12], which was not considered in early research.

Instead, our focus should be on attempts to translate the idealised outcomes of these protocols directly into our formal models - or to create these descriptions ourselves in situations where they are lacking. In addition, we should strive to be precise with the assumptions we make during the modelling process.

Our first objective in this direction is in the translation process: demonstrating the ability to produce results in a symbolic language that immediately relate to the physical problems of real-world settings.

> Research Question 1
>
> Can we translate problems faithfully from other domains into the language of proving tools, in order to produce accurate results?

The relevance of this question is demonstrated in Distance Bounding protocols, where the most precise security claims must consider physical attributes such as the distance between participants.

Clearly such translations require the abstraction of some details, and this is not possible without simplifying assumptions. Similarly, it is known that some goals - particularly those related to liveness properties - are significantly more difficult to directly (dis)prove for any given protocol. This motivates the next question:

> Research Question 2
>
> Can we prove results about the relationships between different security goals in a restricted model?

This direction is inspired by Lowe's hierarchy, which shows a set of *universal* relations between different authentication goals. For example, it is known that any attack trace which violates the security goal known as recent aliveness must necessarily also break weak aliveness, for all protocols. Here, our intent is to reduce the scope of these comparisons. By making precisely-defined assumptions about the structure of the protocols for analysis, complex security goals can be broken down into individual properties. In addition, we investigate the relations between security goals on a protocol-by-protocol basis.

These considerations are relevant for *Terrorist Fraud* attacks on Distance Bounding protocols, a novel scenario in which a prover temporarily deviates from their specification (for example, to issue a one-time access key to a partner). In this case, the security goal is to demonstrate that it is impossible to temporarily misbehave without causing irreversible damage to the security of the protocol.

Our last goal is to construct a methodology for defining new security claims. Examination of several modern protocols demonstrates the need not only for extensions of well-established authentication requirements, but also the formalisation of new goals.

Research Question 3

Can we define new security goals for modern protocols, that truly capture their intended purpose?

We will investigate several domains where such new definitions are needed. The first is in *Key-Updating* protocols, and the analysis of *Desynchronisation Resistance*. Such liveness properties form an active area of research for automata and other processes [AS87], but are often neglected in security analysis. Further, we look at extensions of the classic authentication goals to *proxied protocols*.

Methodology

To solve the problems presented above, the following approach is taken:

- A minimal symbolic security model is defined
- For individual application domains, a series of extensions are created for this model. These extensions introduce new syntax to the model in order to describe the specific problem, alongside semantic restrictions on how the new additions are applied
- Results about the structure of protocols inside these extended models are proven. These include results about how the model relates to implementations in other settings, or grant additional implications to security claims
- These results are used to allow us to prove advanced security claims more easily, by embedding the models inside the specification language of an automated verifier (in this case, the Tamarin prover tool)

1.4 Contributions

The main contributions of this thesis are as follows:

1. We present a faithful translation from the Distance Bounding model of Basin et al. [SSBC09] to one inside the Tamarin Prover. The *Causality-Based Distance Bounding* property is defined and proven to be equivalent to the *Secure Distance Bounding* definition of Basin et al. under a set of simplifying assumptions. As a result, the verification process is transformed from a largely manual one, using a theorem prover, to an almost entirely automatic process.

This simplified process is then put to use by performing an extensive survey of Distance Bounding protocols in the literature, identifying several vulnerabilities in industry protocols including the PayPass protocol for Mastercard [EMV18]

2. We further the work of Lowe [Low97] and Cremers et al. [CMdV06] in describing *relations* between security properties, this time arguing on the level of individual protocols rather than a universal quantifier. In particular, we introduce the notion of *irreversibility*, which represents the idea that any deviation from a protocol specification that breaks one security goal irreversibly breaks another in future executions.

   Further, a *Desynchronisation Resistance* property is defined for key-updating protocols. This property, though not directly verifiable, is shown to be implied by a combination of smaller goals in our specific setting.

3. We develop a series of security definitions for multiparty protocols. The protocols examined are those in which messages are passed through a series of intermediate agents. These include the LoRaWAN protocol suite and other proxied communications such as payment networks or onion routing systems. The main security properties introduced are those of *indirect agreement*, which states that an endpoint can be confident of their partner's execution of the protocol even if they are not directly communicating with them, and *path integrity*, which guarantees that messages flow correctly along a fixed route.

4. We design, implement and verify the *maTLS* protocol, an extension to the TLS protocol suite. This protocol is designed to actively support middlebox participation during TLS, allowing them to contribute functionality to endpoints in a way which guarantees the novel security goals defined above.

1.5 Layout

In Chapter 2 the main model which is used throughout the thesis is introduced, as well as more insight into the modelling approaches used. The rest of the thesis is split into two main parts. Part I, RFID Protocols, contains the first two contributions, addressing the first two research questions. Part II, Multiparty Protocols, contains the second two contributions, addressing the last research question. Throughout this document, several indepth case studies have been highlighted, in order to demonstrate the applicability of the underlying theory. This includes several "real-world" protocols that are in active deployment, such as the PaySafe contactless payment protocol, the Bitcoin Lightning Network, and the LoRaWAN IoT protocol suite.

Part I

Chapter 3 examines the problem of distance bounding protocols. This necessitates an extension of our model that considers agents' physical position. The seminal work of Basin et al. is translated into this model in a provably faithful way, while abstracting away the notions of time and location.

In Chapter 4, we take a step back, investigating relations between security goals on protocols. The idea of *protocol deviations* is introduced, representing (sometimes temporary) changes in the structure of transition rules. This motivates the definition of *irreversibility* of a protocol: that temporary deviations may lead to long-term consequences. Using this new theory, we return to Distance Bounding protocols, to look at a special class of attacks known as Terrorist Fraud. This allows for an extensive and thorough survey - not only of protocols from the literature, but also several industrial standards.

Chapter 5 builds on the idea of relations on security properties. Our focus is swapped to *key-updating protocols*, stateful systems in which agents periodically update their encryption keys. Such techniques are used in *grouping* protocols, which attempt to simultaneously authenticate multiple RFID tags. Desynchronisation Resistance - the notion that keys can never become out-of-sync - is defined. Instead of proving such a challenging liveness property directly, Desynchronisation Resistance is shown to be an implication of a collection of more readily verifiable goals.

Part I is written using the content of three published conference papers, all co-authored with Sjouke Mauw, Jorge Toro-Pozo and Rolando Trujillo-Rasua.
Chapter 3 is based on the IEEE S&P 2018 Paper "Distance-bounding protocols: Verification without time and location" [MSTPTR18b], for which Jorge Toro-Pozo was the main author. The chapter is re-structured to highlight the Tamarin code implementations, and the related work is extended to discuss recent results from the literature.
Chapter 4 is based on the ACM CCS 2019 Paper "Post-collusion security and distance bounding" [MSTPTR19], for which Jorge Toro-Pozo was the main author. The theory section of the chapter is expanded, formalising and expanding the notion of collusion resistance as an irreversibility property, as well as a discussion of the feasibility of brute-force style approaches for analysing protocol deviations.
Chapter 5 is based on the ESORICS 2018 Paper "Automated Identification of Desynchronisation Attacks on Shared Secrets" [MSTPTR18a], for which I was the main author.

Part II

Chapter 6 introduces the notions of *path-based* protocols and gives definitions used as a foundation for the following work. *Path integrity* - the idea that messages must flow between multiple agents in a specific order - is formally defined, and a survey of several protocols from multiple different domains is made.

Chapter 7 and Chapter 8 form two extended case studies into specific multiparty settings. Chapter 7 considers the case of TLS extended with *middleboxes*, introducing a new protocol named *maTLS* alongside a set of verified security goals. Chapter 8 looks instead at the LoRaWAN protocol family for IoT devices, where an extremely fine-grained model is used to highlight vulnerabilities.

Part II is written using the content of three research papers.
Chapter 6 is based on work with Hugo Jonker, Hyunwoo Lee and Sjouke Mauw, for which I am the main author.
Chapter 7 is based on the publication "maTLS: How to Make TLS middlebox-aware" [LSL+19], coauthored with Hyunwoo Lee, Junghwan Lim, Gyeongjae Choi, Selin Chun, Taejoong Chung and Taekyoung Kwon at the NDSS Symposium 2019.
Chapter 8 is based on the EuroS&P 2020 paper "Extensive Security Verification of the LoRaWAN Key-Establishment: Insecurities & Patches". This work was a result of a collaboration with Ioana Boreanu, Steve Wesemeyer and Helen Trehearne.

Finally, Chapter 9 contains some conclusions relating to the overall thesis and the work contained therein, including a discussion of future avenues of research.

In addition to the papers listed above, I was also a contributing author to the ESORICS 2019 Paper "Breaking Unlinkability of the ICAO 9303 Standard for e-Passports Using Bisimularity" [FHMS19], with Ihor Filimonov, Ross Horne and Sjouke Mauw. The content of the aforementioned paper is not included in this thesis, as it does not fit with the underlying discussion of authentication properties.
I was also a contributor to several poster sessions at workshops. The content of these posters is subsumed by the papers that followed them, and so they are not mentioned here.

Chapter 2

A Multiset Rewriting Model

At the core of any symbolic analysis is a transition system, which mimics the true execution of the program. In this chapter we introduce the model that serves as a foundation for the work throughout this thesis.

Here, the transition system of choice is based upon multiset rewriting theory. Informally, at any time, the state of a protocol's execution is stored as a collection of facts. A set of rules - describing both the protocol's intended behaviour, and the actions of the adversary - change these facts over each run of the protocol.

The language we use can be considered as a subset of that supported by the Tamarin automatic verifier [MSCB13][The18]. Indeed, this is by design, allowing us to directly encode all of the protocols we will consider. However, we intentionally use only a small part of the full expressiveness of the tool at any given time. This will enable us to prove results about the behaviour of protocols built in each of the extended models we will consider over the course of the thesis.

This chapter is organised as follows. Section 2.1 contains an introduction to multiset rewriting theory and its application to modelling communication protocols. Section 2.2 introduces the core methodology used throughout the thesis: the concept of *extensions* to the base model, and reasoning about protocols built in them.

2.1 Fundamentals

In this section we give an overview of the multiset-rewriting model that will be used throughout this work. We introduce the use of rules to transition between different potential execution states of a protocol, and describe how these transitions can be used to describe security goals. In later sections we will augment this model: adding additional notation to describe how transitions may occur, or new ways in which a protocol might be specified or analysed.

We begin with an order-sorted algebra. We define two top-level sorts *msg* and *Fact*, and subsorts *pub, fresh, const* such that

$$const < pub < msg$$
$$fresh < msg$$

We write $x : y$ to indicate that the term x is of type y. We will usually use the following symbols for terms of various types:

- 'a' : *const*
- A : *pub*
- x : *fresh*
- m : *msg*

We allow for collections of function symbols $\Sigma_{msg^*,msg}$ and $\Sigma_{msg^*,Fact}$, which map a sequence of type *msg* (or its subsorts) to either another *msg* or a *Fact*. For simplicity we denote $\Sigma_{msg^*,msg}$ by Σ and $\Sigma_{msg^*,Fact}$ by F.

As we are working in an abstract model, function symbols are not evaluated, and instead the application of functions can lead to terms of containing several components. For example, we may have a term $m = f(x,y)$, in which case we call the symbols x and y *subterms* of m.

Atoms (i.e. undecomposable terms) can represent names (unassigned expressions), or variables (assigned values). A term is said to be *ground* if it contains no variables. A *substitution*, σ, is a (partial) function from variables to ground terms of the same sort (or a subsort). A substitution is applied to a term by applying it to each subterm. Given a term t and a substitution σ, we say that the substitution σ *grounds* t if the resultant term $t\sigma$ is a ground term.

We allow for an equational theory E over terms of (sub)type *msg*. E is a collection of equations $lhs = rhs$. For convenience we assume that E is *subterm-convergent*: for each equation we have that *rhs* is either a subterm of E or a constant. Two terms s and t are said to be equivalent modulo E if a series of equations can be applied to s or t (or both) such that the resulting terms s' and t' are equal. We reserve the following collection of function symbols Σ, with equational theory E:

- $\Sigma = \langle\ \rangle/2, fst/1, snd/1, h/1,$
  $aenc/2, adec/2, sdec/2, senc/2, pk/1$
- $fst(\langle x,y \rangle) = x$
- $snd(\langle x,y \rangle) = y$
- $sdec(senc(m,k),k) = m$
- $adec(aenc(m,pk(k),k)) = m$

Intuitively, the *aenc* and *adec* functions model asymmetric encryption and decryption, with *senc* and *sdec* corresponding to the symmetric case. We will write $\{x\}_k$ to denote encryption when the kind is apparent from the context. Similarly, when possible, we will often omit the pair operators $\langle$ and $\rangle$ when terms are clearly paired, for readability reasons. h denotes a hash function with no attached equational theory.

We reserve the following fact symbols with corresponding intuition:

- $\mathrm{Net}\,/1$: A message on the communication network
- $\mathrm{K}\,/1$: Adversary knowledge of a term
- $\mathrm{Pk}\,/2, \mathrm{Ltk}\,/2$: A public/private keypair
- $\mathrm{ShKey}\,/3$: A shared encryption key

Protocol Specification

Thusfar, our discussion has been restricted to terms of type *msg*. We now divert our attention to *Facts*. From now on we assume that we are working over multisets where all terms are of type *Fact*. A *State*, S, is a multiset where all of the terms are ground, and models the current execution state of a protocol. Each run of a protocol will begin with an empty state, which then transitions into future states through a series of *rules*.

A *rule* r is defined by a triplet of multisets $r : L \xrightarrow{E} R$. Given a state S, and a substitution σ, we can apply the rule r if:

- σ is a grounding substitution for L
- $L\sigma \subset S$

In this case, the state S' is produced by removing the submultiset of S equal to $L\sigma$, and replacing it with $R\sigma$.

The elements of $E\sigma$ are known as the *event facts* of the rule instance. By convention, we assume that the set of event fact symbols is disjoint to the set of other fact symbols.

For convenience, we will sometimes use the prefix ! in fact symbols to indicate that the fact is *persistent*. Persistent facts are never removed from a state as a result of rule execution (if they appear on the left hand side of a rule, we assume they also appear on the right hand side). They model reusable assets, such as encryption keys or adversary knowledge.

A simple example of a rule is given in `Dec_Fwd` below, in which an agent (whose name will instantiate the variable A), detects a message on the network encrypted with their public key, and decrypts it before forwarding on the result.

$$\texttt{Dec\_Fwd} := \begin{bmatrix} \mathrm{Net}(\{m\}_{pk(k)}) \\ !\,\mathrm{Ltk}(A, k) \end{bmatrix} \rightarrow \begin{bmatrix} \mathrm{Net}(m) \end{bmatrix}$$

Notationally, event facts will be written on top of the arrow between the premises and conclusions of a rule. In this case, `Dec_Fwd` is defined to have no event facts – Figure 2.1 contains some examples of rules containing event facts.

When a rule is applied, the terms $E\sigma$ are appended to the *trace*, τ, an indelible ordered history of event markers. At the start of any execution, $\tau_0 = \phi$, the empty sequence. After the execution of a rule r, the resulting event facts are added along with a discrete time marker t_i, e.g. $\text{Event}(x)@t_1$. Time markers are assumed to be increasing - the event facts generated by two separate rule executions must have two different time markers. We will freely quantify over time markers e.g. $\forall t_i$ when there is no ambiguity, and will make use of ordering of time markers (e.g. $t_i < t_j$).

We reserve the following special rule `Fresh`:

$$\texttt{Fresh} := \Big[\quad - \quad \Big] \longrightarrow \Big[\text{Fr}(x : \mathit{fresh}) \Big]$$

This rule allows for the creation of freshly generated random variables, for example for use in creating encryption keys. We further require that the `Fresh` rule is the **only** way in which a Fr fact can be created. Note that in general this means that the adversary cannot know the value of any term generated by the `Fresh` rule, unless it is later released as part of a message.

A protocol, P, is given by $P = (R, \mathsf{F}, \Sigma, E)$, a tuple of rules, facts, functions and an equational theory. We will assume that R, F, Σ and E contain all the reserved elements indicated in this section (including the adversary rules below), except when stated otherwise. We define $\mathit{Traces}(P)$ as the set of all (valid) traces that can be constructed as a result of executing the rules from R along with the associated material from the other protocol components. Similarly, we define $\mathbb{U}(\Sigma)$ to be the set of all states that can be created as the consequence of a set of rule executions on P. Finally, given a specific execution of a protocol, we define $\text{lastState}(\tau)$ to be the state at the end of this execution.

Adversary Model

The network adversary is also defined in terms of multiset rewriting rules. The basis of these is the Dolev-Yao [DY83] adversary. The set of these rules, *Adv*, is given in Figure 2.1.

The Dolev-Yao adversary is capable of blocking or modifying any messages received on the network, modelled by the `Block` and `Inject` rules. The $\texttt{Fun}_\texttt{f}$ rule allows the adversary to derive new terms by applying function symbols to known terms, and `Adv_Pub` and `Adv_Fr` allow the adversary to deduce public and (previously unused) fresh terms. Finally, `Corrupt` rule models agents who are fully under the control of the adversary – we use the Corrupt fact to additionally indicate that a specific agent is corrupted. Typically, security claims will be made modulo corruption (i.e. we are only concerned with traces in which the fundamental agents are honest).

$$\texttt{Inject} := [\ !\mathrm{K}(x)\ ]\rightarrow[\ \mathrm{Net}(x)\ ]$$

$$\texttt{Block} := [\ \mathrm{Net}(x)\ ]\xrightarrow{\mathrm{K}(x)}[\ !\mathrm{K}(x)\ ]$$

$$\texttt{Adv\_Pub} := [\ \ -\ \ ]\xrightarrow{\mathrm{K}(A)}[\ !\mathrm{K}(A : pub)\ ]$$

$$\texttt{Adv\_Fr} := [\ \mathrm{Fr}(x)\ ]\xrightarrow{\mathrm{K}(x)}[\ !\mathrm{K}(x)\ ]$$

$$\texttt{Fun}_{\texttt{f}} := \begin{bmatrix} !\mathrm{K}(x_1) \\ !\mathrm{K}(x_2) \\ \cdots \end{bmatrix}\xrightarrow{\mathrm{K}(f(x_1,x_2,\ldots))}[\ !\mathrm{K}(f(x_1,x_2,\ldots))\ ]$$

$$\texttt{Corrupt\_Ltk} := [\ !\mathrm{Ltk}(A,k)\ ]\xrightarrow{\mathrm{Corrupt}(A)}[\ !\mathrm{K}(k)\ ]$$

$$\texttt{Corrupt\_L} := [\ !\mathrm{ShKey}(A,B,k)\ ]\xrightarrow{\mathrm{Corrupt}(A)}[\ !\mathrm{K}(k)\ ]$$

$$\texttt{Corrupt\_R} := [\ !\mathrm{ShKey}(A,B,k)\ ]\xrightarrow{\mathrm{Corrupt}(B)}[\ !\mathrm{K}(k)\ ]$$

FIGURE 2.1: Rules which define the Dolev-Yao adversary.

Security Goals

Security goals of a protocol are given in terms of first-order logic formulae on the set of traces of the protocol. Intuitively, they indicate that certain events can or cannot happen, or that they occur in a certain order. These goals are defined in terms of event facts.

As a simple example, consider the following protocol in Alice-Bob notation:

$$\begin{array}{ll} A \rightarrow B: & \{x, y\}_{k_{AB}} \\ B \rightarrow A: & \{y, x\}_{k_{AB}} \end{array}$$

In this protocol, A sends a pair of encrypted terms to B, who reverses their order and sends them back. A reasonable security claim for this protocol might be "The value x is either unknown to the adversary, or one of A or B is corrupt". This could be expressed as:

$$\begin{aligned} &\forall \tau \in \mathit{Traces}(P), \forall t_i, \forall A, B, x : \mathit{msg} : \\ &\quad \mathrm{Secret}(A, B, x)@t_i \implies \\ &\quad\ \ !\exists t_a \mid \mathrm{K}(x)@t_a \vee \\ &\quad\ \ \exists t_b \mid \mathrm{Corrupt}(A)@t_b \vee \\ &\quad\ \ \exists t_c \mid \mathrm{Corrupt}(B)@t_c \end{aligned}$$

Often the event facts in security claims contain unassigned names, and therefore can be instantiated with a range of values. The faithfulness of security claims to the protocol they are modelling is thus dependent on the correct placement and usage of such event facts within the protocol specification.

2.2 Modelling Methodology

One potential risk with encoding protocols in this model is that important event markers or rules may not always be correctly applied. This may not be an issue for the analysis of individual protocols - where the person performing the implementation can ensure that the model is faithful to the spirit of the specification. However, as much as possible, we would like to reason about *all* protocols within a specific setting.

To do this, we will define extensions to our base model, which introduce new facts, event facts, and rules - describing the structure of protocols, or providing new adversarial capabilities. More significantly, for these extensions, we will give a formalisation of *well-formedness* for protocols modelled inside them. This definition of well-formedness will differ from case to case, but will aim to fulfill the following goals:

- **Intuitive Application of Event Facts**: Enforcements should be placed on the usage of event facts in rules to ensure that they match the intuition of the security goals associated with them.
- **Controlling Protocol Setup**: For protocols which require setup (e.g. where users have encryption keys established before the protocol begins), this is handled in a way that matches the assumptions of the real-world system.
- **One-to-oneness**: Given a specification written in a simple notation (e.g. Alice-and-Bob, or a message sequence chart), there should be a unique representation of this protocol in the extended model (up to simple transformations).

Generally, well-formedness will be broken down into a collection of simple assumptions about the structure of rules in a protocol. For example, in Chapter 5, we define *key uniqueness*, which intuitively requires that each encryption key generated in a trace is distinct. This manifests itself as a statement about the presence and usage of the ShKey fact in rules.

The benefits of restricting to our analysis to protocols which follow our well-formedness goals are as follows:

1. We can ensure that the semantic interpretation of a security claim actually matches its real-world intent.
2. We can prove results about security claims inside the model - either how they relate to each other, or to claims made in similar models.
3. It allows for improved automation of the verification process. Knowing that protocols must (by definition) share parts of their structure allows us to re-use parts of the specification code.

We will see each of these advantages demonstrated over the course of the rest of the thesis.

Bibliography

[AS87] Bowen Alpern and Fred B Schneider. Recognizing safety and liveness. *Distributed computing*, 2(3):117–126, 1987.

[BC93] Stefan Brands and David Chaum. Distance-bounding protocols. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 344–359. Springer, 1993.

[BC10] David Basin and Cas Cremers. Modeling and analyzing security in the presence of compromising adversaries. In *European Symposium on Research in Computer Security*, pages 340–356. Springer, 2010.

[BDH+18] David Basin, Jannik Dreier, Lucca Hirschi, Saša Radomirovic, Ralf Sasse, and Vincent Stettler. A formal analysis of 5G authentication. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1383–1396. ACM, 2018.

[Ber09] Daniel J Bernstein. Introduction to post-quantum cryptography. In *Post-quantum cryptography*, pages 1–14. Springer, 2009.

[BFG19] Jacqueline Brendel, Marc Fischlin, and Felix Günther. Breakdown resilience of key exchange protocols: NewHope, TLS 1.3, and hybrids. In *European Symposium on Research in Computer Security*, pages 521–541. Springer, 2019.

[Bla01] Bruno Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *CSF'01*, pages 82–96, 2001.

[BM82] Robert S Boyer and J Strother Moore. A mechanical proof of the unsolvability of the halting problem. Technical report, Texas Univ At Austin Inst For Computing Science And Computer Applications, 1982.

[CDM+00] Iliano Cervesato, Nancy Durgin, John Mitchell, Patrick Lincoln, and Andre Scedrov. Relating strands and multiset rewriting for security protocol analysis. In *Proceedings 13th IEEE Computer Security Foundations Workshop. CSFW-13*, pages 35–51. IEEE, 2000.

[Cer01] Iliano Cervesato. The Dolev-Yao intruder is the most powerful attacker. In *16th Annual Symposium on Logic in Computer Science—LICS*, volume 1, 2001.

[CGCG16] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. On post-compromise security. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 164–178. IEEE, 2016.

[CHH+17] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1773–1788. ACM, 2017.

[CKR18] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. Deepsec: Deciding equivalence properties in security protocols theory and practice. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 529–546. IEEE, 2018.

[CMdV06] Cas JF Cremers, Sjouke Mauw, and Erik P de Vink. Injective synchronisation: an extension of the authentication hierarchy. *Theoretical Computer Science*, 367(1-2):139–161, 2006.

[Cre08] Cas JF Cremers. The scyther tool: Verification, falsification, and analysis of security protocols. In *International Conference on Computer Aided Verification*, pages 414–418. Springer, 2008.

[CRSC12] Cas Cremers, Kasper B Rasmussen, Benedikt Schmidt, and Srdjan Capkun. Distance hijacking attacks on distance bounding protocols. In *2012 IEEE Symposium on Security and Privacy*, pages 113–127. IEEE, 2012.

[DHRS18] Jannik Dreier, Lucca Hirschi, Sasa Radomirovic, and Ralf Sasse. Automated unbounded verification of stateful cryptographic protocols with exclusive OR. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 359–373. IEEE, 2018.

[DKR07] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Symbolic bisimulation for the applied pi calculus. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 133–145. Springer, 2007.

[DLK+14] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, et al. The matter of heartbleed. In *Proceedings of the 2014 conference on internet measurement conference*, pages 475–488. ACM, 2014.

[DY83] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Trans. Information Theory*, 29(2):198–207, 1983.

[EMV18] EMVCo. *EMV Contactless Specifications for Payment Systems, Book C-2, Kernel 2 Specification, Version 2.7.* April 2018.

[FHMS19] Ihor Filimonov, Ross Horne, Sjouke Mauw, and Zach Smith. Breaking unlinkability of the icao 9303 standard for e-passports using bisimilarity. In *European Symposium on Research in Computer Security*, pages 577–594. Springer, 2019.

[FJHG99] THAYER Fábrega, F Javier, Jonathan C Herzog, and Joshua D Guttman. Strand spaces: Proving security protocols correct. *Journal of computer security*, 7(2-3):191–230, 1999.

[Gol98] Oded Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, 78, 1998.

[Gün89] Christoph G Günther. An identity-based key-exchange protocol. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 29–37. Springer, 1989.

[Hoa78] Charles Antony Richard Hoare. Communicating sequential processes. In *The origin of concurrent programming*, pages 413–443. Springer, 1978.

[JCCGS19] Dennis Jackson, Cas Cremers, Katriel Cohn-Gordon, and Ralf Sasse. Seems legit: Automated analysis of subtle attacks on protocols that use signatures. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2165–2180. ACM, 2019.

[KS05] Lea Kissner and Dawn Song. Privacy-preserving set operations. In *Annual International Cryptology Conference*, pages 241–257. Springer, 2005.

[LB06] Christine Laurendeau and Michel Barbeau. Threats to security in DSRC/WAVE. In *International Conference on Ad-Hoc Networks and Wireless*, pages 266–279. Springer, 2006.

[Low96] Gavin Lowe. Breaking and fixing the needham-schroeder public-key protocol using FDR. In *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, pages 147–166. Springer, 1996.

[Low97] Gavin Lowe. A hierarchy of authentication specification. In *CSF'97*, pages 31–44, 1997.

[LSL$^+$19] Hyunwoo Lee, Zach Smith, Junghwan Lim, Gyeongjae Choi, Selin Chun, Taejoong Chung, and Ted Taekyoung Kwon. matls: How to make TLS middlebox-aware? In *NDSS*, 2019.

[MSCB13] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *International Conference on Computer Aided Verification*, pages 696–701. Springer, 2013.

[MSTPTR18a] Sjouke Mauw, Zach Smith, Jorge Toro-Pozo, and Rolando Trujillo-Rasua. Automated identification of desynchronisation attacks on shared secrets. In *European Symposium on Research in Computer Security*, 2018.

[MSTPTR18b] Sjouke Mauw, Zach Smith, Jorge Toro-Pozo, and Rolando Trujillo-Rasua. Distance-bounding protocols: Verification without time and location. In *IEEE Symposium on Security and Privacy, S&P'18, May 21–23, 2018, San Francisco, California, USA*, may 2018.

[MSTPTR19] Sjouke Mauw, Zach Smith, Jorge Toro-Pozo, and Rolando Trujillo-Rasua. Post-collusion security and distance bounding. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2019.

[NS78] Roger M Needham and Michael D Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[PD16] Joseph Poon and Thaddeus Dryja. The Bitcoin lightning network: Scalable off-chain instant payments, 2016.

[Ros94] Bill Roscoe. Model- checking CSP. 1994.

[RS11] Mark D Ryan and Ben Smyth. Applied pi calculus. 2011.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[Sho94] Peter W Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. *IACR Cryptology ePrint Archive*, 2004:332, 2004.

[SSBC09] Patrick Schaller, Benedikt Schmidt, David Basin, and Srdjan Capkun. Modeling and verifying physical properties of security protocols for wireless networks. In *2009 22nd IEEE Computer Security Foundations Symposium*, pages 109–123. IEEE, 2009.

[The18] The Tamarin Team. *The Tamarin User Manual*, 2018.

[TW89] Martin Tompa and Heather Woll. How to share a secret with cheaters. *journal of Cryptology*, 1(3):133–138, 1989.

Part I

RFID Protocols

Chapter 3

Distance Bounding Protocols

In the first part of this thesis we focus on the application domain of communication protocols for RFID devices. Small, highly portable RFID tags have enabled a variety of novel applications for short range interactions, including access control, payment systems and tracking services.

The design of these protocols is affected by the physical properties of these tags in two main ways. Firstly, many protocols implicitly (or explicitly) carry authentication requirements conditional on physical properties of the tags - such as their location, or machine-readable data printed on the device. This presents a challenge in constructing models which attempt to capture these details. Secondly, RFID tags are often restricted in their capability for storage and computation. As a result, their protocols often involve different cryptographic primitives to those seen in traditional network communication. Although this does not affect the choice of security goals directly, it can lead to new attack vectors and enables fine-grained models which can faithfully represent these primitives.

In this chapter, we closely examine the application domain of Distance Bounding protocols. These protocols attempt to mitigate the risk of relay attacks, in which an attacker forwards a radio signal far beyond an RFID tag's intended radius.

The work in this chapter is based on the IEEE S&P 2018 Paper "Distance-bounding protocols: Verification without time and location" [MSTT18].
The content here is restructured to highlight the Tamarin code that was developed while researching this topic. In partiular, the appendices which cover modelling specifics have been pushed earlier in the chapter. In addition, some of the results from the paper are delayed until the next chapter, which extends the model presented here. The related work has been updated.
My main contribution to this paper was in translating the model (which uses an extension of the Cremers-Mauw model) into the Tamarin specification language used in our analysis of protocols in the literature.
Special thanks to Alexandre Debant and Prof. Stephanie Delaune, whose comments on both the paper and the codebase helped improve it greatly.

3.1 Introduction to Distance Bounding

The increasing availability of miniature devices capable of wireless communication allows for a vast range of new additions to our day-to-day lives. Passive RFID chips embedded in credit cards or passports, as well as transmitters in devices such as car keys, allow for authentication protocols to be run using radio signals at a distance.

However, contactless communication is known to be vulnerable to *relay attacks* [DGB87a]. In such a scenario, an adversary relays the verbatim messages that are being exchanged, for example by using a radio retransmitter.

Relay attacks are mostly used to break communication protocols with a bounded read range, such as smartcards (2-10 cm) or car keys (10-100 m). By simply relaying, an adversary is able to establish a long-range communication between two contactless tokens, which otherwise wouldn't be possible. This has been used, for example, by Francillon et al. [FDC11] to break the passive keyless entry system of various modern cars.

These attacks are not covered by standard authentication security goals. Indeed, messages are correctly exchanged by the intended parties. However, security violations occur as a result of the owner of a device either not intending to perform the protocol, or possibly not being aware that it is taking place at all. For example, in the case of passive RFID tags, an attacker who is able to get close enough to power a tag can seamlessly relay messages from it to a distant recipient.

To face relay attacks, Desmedt et al. [BBD$^+$91, BD90] introduced the notion of *distance-bounding protocols*, and the first such protocol was designed by Brands and Chaum [BC93]. Distance-bounding protocols use the round-trip time (RTT) of one or more challenge/response rounds to provide an upper bound on the distance between a prover and a verifier (see Figure 3.1a). Through this scheme, security verification translates into the validity of the actual prover-to-verifier distance in comparison with the RTTs. More precisely, in a secure distance-bounding protocol, if the prover-to-verifier distance is d and the RTT is Δt, then it must hold that $d \leq \frac{1}{2}\Delta t \cdot \mathsf{c}$, where c denotes the maximum network transmission speed (for radio-waves, this is the speed of light). This intuition is supported by the physical fact that no message can be transmitted at a speed higher than c.

In the context of distance-bounding protocols, their security has traditionally been verified over the years by accounting for their resistance to three types of attack: mafia fraud [DGB87a], distance fraud [Des88], and terrorist fraud. Resistance is measured in terms of probability of success of the adversary in a given adversary model [ABK$^+$11, DFKO11, BV14]. For example, the chance of an attacker to correctly guess the proper response in each round determines the overall likelyhood of an attack succeeding. However, this probabilistic analysis based on attack-resistance does not seem to be a promising verification scheme, as new attacks might be discovered in the future.

A clear and convincing demonstration of this style of analysis is given by Cremers et al. [CRSC12]. In this work, the authors prove several protocols to be vulnerable to *distance-hijacking* attacks while they were previously considered secure as they

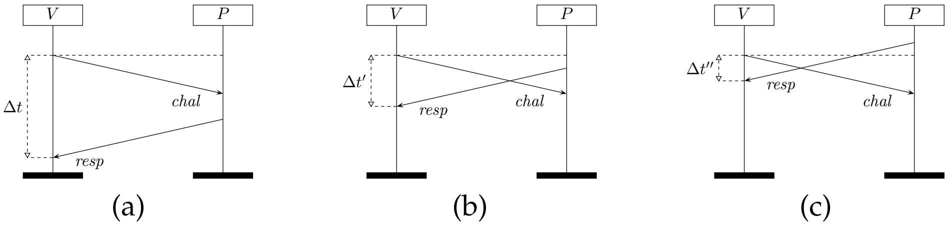


FIGURE 3.1: Three timing scenarios of a challenge/response round.

resisted the then-existing attack types (mafia, distance and terrorist frauds).

Unfortunately, although the desired properties of a distance-bounding protocol can be precisely defined in current security models, it is not necessarily straightforward to verify that a given protocol satisfies these properties. On the one hand, computational models [DFKO11, BMV13] typically lead to manual and complex security proofs. On the other hand, symbolic models [SSBC09, CRSC12] rely on using adapted versions of higher-order theorem-proving tools such as Isabelle/HOL [NPW02], which require a high degree of user intervention. This means that verifying the security of a distance-bounding protocol in the existing symbolic models requires not only a considerable amount of expertise, but also a significant time investment.

This chapter argues that the notions of time and location are indeed *not* needed to specify and verify the security of distance-bounding protocols. Surprisingly enough, such protocols can be verified by considering the causal order of events in protocol traces, similarly to authentication properties like aliveness and synchronization [CM12]. The intuition behind this observation is illustrated in Figure 3.1.

Figure 3.1 shows a regular challenge/response round, in which prover P can only respond to verifier V's challenge after having received the challenge. Therefore, $\frac{1}{2}\mathsf{c} \cdot \Delta t$ determines an upper bound on the distance d between V and P. Now, suppose that, due to a vulnerability of the protocol, P is able to predict the appropriate response before having received the challenge (Figure 3.1b). This means that he will be able to send his response "too early", leading to a shorter round-trip time $\Delta t' < \Delta t$ and thus to a smaller and incorrect distance calculated by V. Thus, if the protocol is insecure because P can preempt the response, P has sufficient knowledge to create the response before reception of the challenge. Now our main observation is that (assuming that there is no other causal relation between sending the challenge and P's knowledge), P could even have sent the response *before* V *sent the challenge* (Figure 3.1c). From a causal point of view, this means that if there is a trace in which P sends its response before P receives the challenge, there must also be a trace in which P sends the response before V sends the challenge. Hence, a flaw in the protocol translates into such a wrongly ordered trace, which can be discovered through an analysis that does not consider time.

The rest of the chapter is organised as follows:

- In Section 3.2, we review the related work on modelling distance-bounding protocols. We focus our attention on an adaptation of the security model of Basin et al. [BCSS09, SSBC09], creating an equivalent formalisation in a model similar to that of Cremers-Mauw. We then formally define a security goal named *secure distance-bounding*.

- Then, in Section 3.3, we analyse the semantic domain and formulate a number of basic properties that provide a sufficient characterization of the semantics to prove our main result. These definitions are used to formulate a notion of *causality-based secure distance-bounding*, in which the notions of time and location have been removed. We then prove this definition is in fact equivalent to the previously defined notion of *secure distance-bounding*.
- In order to validate our results, we demonstrate an implementation of causality-based secure distance-bounding in Tamarin [MSCB13]. We discuss some of the modelling details required to perform the translation into this tool.
- Finally, a case study is provided on the TREAD distance-bounding protocol to demonstrate the applicability of the analysis in a more in-depth fashion, in Section 3.5.

The model developed in this chapter is extended further in Chapter 4, which also contains the results of an extensive security survey of distance-bounding protocols.

3.2 Modelling Distance Bounding

In this section we introduce distance bounding protocols. Subsection 3.2 presents a discussion of the first distance bounding protocols, as well as their security goals. In Subsection 3.2, strategies for the security analysis of distance bounding protocols are presented, including some of the most recent work. Finally, Subsection 3.2 contains an in-depth analysis specifically of the work of Basin et al. [BCSS09]. We express their model (which makes use of continuous event timings) in a syntax which will form the basis of the rest of this chapter.

Foundations

Distance-Bounding Protocols.
The first distance-bounding protocol was designed by Brands and Chaum [BC93] and it is composed of three phases. The *slow phase* (a.k.a. initial phase, setup phase) is where the parties agree on the parameters of the session, such as nonces. Then the *fast phase* (a.k.a. critical phase, distance-bounding phase, timed phase) is executed, consisting of a number of challenge/responses rounds, where the verifier measures the round-trip times. Finally, a *verification phase* (a.k.a. final phase, authentication phase) takes place, in which the verifier makes a decision on whether the prover successfully passed the protocol. This is done by checking the correctness of all round-trip times and the prover's proof of knowledge of a valid signature.

Another well-known distance-bounding protocol was proposed by Hancke and Kuhn in [HK05]. An abstraction of this protocol is shown in Figure 3.2. The first two

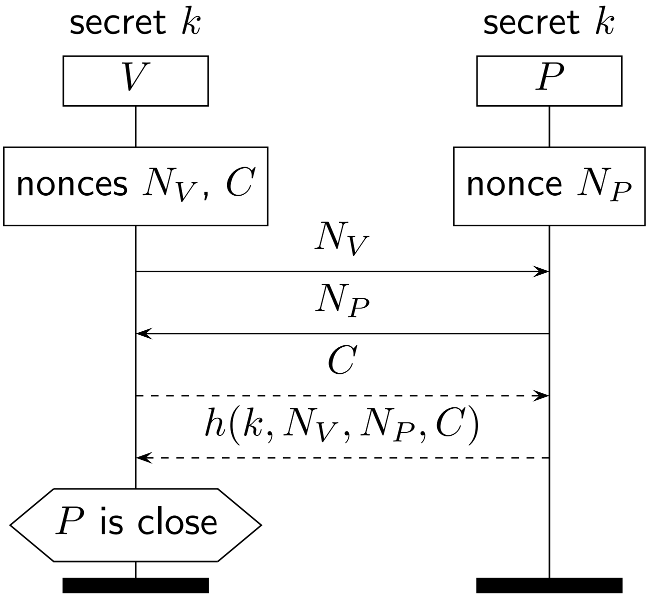


FIGURE 3.2: A representation of Hancke and Kuhn's protocol.

messages compose the initial phase of the protocol, where the verifier V sends his nonce N_V to the prover P who replies back with his nonce N_P. Then the fast phase starts (represented by dashed arrows) with V sending his challenge C to P whose response is $h(k, N_V, N_P, C)$, where k is the shared secret key between V and P and h is an irreversible cryptographic function. The verification phase is represented by V's claim that "P is close". The protocol seems to be secure, as for an attacker (who could be an untrusted prover) to pass the protocol, he must know either the verifier's challenge in advance or the shared secret key between the verifier and the intended prover. However, due to the particular choice of h, a mafia-fraud attacker successfully passes the protocol with a non-negligible probability of $(3/4)^{|C|}$ (see [HK05] for further details).

One of the main differences between Brands and Chaum's protocol and Hancke and Kuhn's protocol is as follows. In the former, the fast phase messages do not rely on long-term secret keys whereas in the latter protocol, such a reliance does exist. Various protocols have been proposed following this characteristic of Brands and Chaum's approach, e.g. [RC10, CBH03, MPP+07, ABG+17, CGdR+15] whilst others employ Hancke and Kuhn's design, such as [AT09, TMA10, MTT16a, KA09, MP08].

Attacks on Distance-Bounding Protocols.
Although distance-bounding protocols solved the problem of relay attacks to some extent, more sophisticated attacks have emerged, such as *mafia fraud*, *distance fraud*, *terrorist fraud* and *distance hijacking*.

Mafia-fraud attacks were introduced in [DGB87a], in which a dishonest agent A uses an honest prover P to provide a verifier V with a false upper bound on the distance between V and P. Some authors consider mafia-fraud attacks to be the same as relay attacks. Others, however, classify mafia-fraud attackers stronger than relay attackers by assuming that the former can manipulate/modify the messages, rather than simply relaying them.

A distance-fraud attacker [Des88] is a dishonest prover A whose goal is to provide a verifier V with a false upper bound on V's distance to A. In particular, for this type of attack, A does not use any other prover to perform his attack.

More sophisticated attacks are *terrorist fraud* and *distance hijacking*. Terrorist-fraud attacks were first discussed in [Des88] in which the attacker prover A cheats on the upper bound on the distance between a verifier V and a dishonest prover P, without learning P's secret key material. Distance hijacking was introduced by Cremers at al. in [CRSC12], in which a dishonest prover A makes use of honest provers in order to provide a verifier V with a false upper bound on the distance between V and A.

Analysing Distance Bounding Security

Probabilistic Security Analysis.
The work by Avoine et al. [ABK+11] introduces a framework that explores the adversary's capabilities and strategies and the influence of provers' abilities to tamper with their devices. New concepts in the distance-bounding field are introduced such as black-box and white-box models.

The concepts sketched in [ABK+11] were soon formulated in computational models. For example, Dürholz et al. formalized the classical frauds (except for distance hijacking) by using an adversary model that does not allow for corrupted verifiers [DFKO11]. Boureanu, Mitrokotsa, and Vaudenay introduced a more general model [BMV13] by allowing adversaries to interact with multiple provers and verifiers, hence capturing distance hijacking [CRSC12].

Mauw, Toro-Pozo, and Trujillo-Rasua [MTT16a, MTT16b] developed a probabilistic analysis of the security of a class of distance-bounding protocols in terms of mafia fraud. This class includes distance-bounding protocols that do not have a final verification phase and are based on precomputation (e.g. [HK05, AT09, GAA11, KKBD12, KA11, MTT16a, MP08, TMA10]). They proposed a set-of-automata representation of protocols that allows the analyst to generically compute the success probability of mafia-fraud attacks.

Symbolic Security Analysis.
Meadows et al. [MPP+07] propose a formal framework to verify distance-bounding protocols. Their approach does not particularly deal with multi-prover scenarios, therefore neither distance-hijacking nor terrorist-fraud attacks are detected.

The first formal framework for distance-bounding protocols with multi-prover scenarios was proposed by Malladi et al. [MBK10], along with a software tool. They analyse the signature-based Brands and Chaum's protocol and find an attack in which an adversary who is not in the vicinity of the verifier still passes the protocol. They call this attack the *farther adversary* scenario. Moreover, to solve the security issue they found, they observed that including the prover's identity in the signature would make the protocol no longer vulnerable to farther adversary attacks.

Basin et al. [BCSS09, SSBC09] introduced a simple yet powerful formal approach for distance-bounding verification. Their model captures dishonest prover behaviors and, by extension, distance-fraud and distance-hijacking attacks, of which the latter was referred to as *impersonation attacks*. Their implementation of the formalization is written in the higher-order logic theorem prover Isabelle/HOL [NPW02]. Similarly to Malladi et al. [MBK10], they prove that the signature-based Brands and Chaum's protocol can be fixed by explicitly adding the prover's identity to the responses in the fast phase.

In [CRSC12], Cremers et al. extended Basin et al's model to capture bit-level message manipulation on wireless networks, introduced as *overshadowing* in [PTDC11]. Supported by this, they proved that including the prover's identity (neither by XOR-ing it with the challenge responses nor by using secure channels) in Brands and Chaum's protocol does not solve its vulnerability to distance hijacking.

Several symbolic approaches have been created to remove the dependence of timing data from security analysis. Chothia et al. [CdRS18] classify attacks on distance bounding protocols based on configurations of locations of honest and dishonest provers. They build a tool which takes a distance bounding protocol and produces a machine-verifiable protocol specification in Proverif for each possible configuration of agent locations. Debant et al. [DD19, DDW19] construct a model in an extended process calculus, which is then embedded inside the Proverif tool directly.

Our work also uses a symbolic approach that abstracts away the use of time and location. Our focus is on making a model that allows protocols to be rapidly implemented and verified.

The Model of Basin et al.

We now describe the formalism of Basin et al. [BCSS09, SSBC09] which is the basis for our work. The formalism employs logic theories to handle inductively-defined sets of traces that represent the protocol's executions. It considers execution traces that consist of a sequence of timed-events, e.g. denoting the sending and reception of messages, where timestamps represent the point in time at which the events occurred.

Agents and Messages.
The set of agents is denoted by Agent, and $\{\mathsf{Honest}, \mathsf{Dishonest}\}$ is a partition of the set of agents into honest and dishonest agents.

During a protocol execution, agents exchange messages through the network. Basic messages are agent names (Agent), nonces (Nonce), and constants (Const). More complex messages can be defined by using atomic messages as the arguments of a function, by pairing them together into a single message or by denoting an encrypted message. Formally, the set of messages Msg is defined by the following grammar, where $atom \in \mathsf{Const} \cup \mathsf{Agent} \cup \mathsf{Nonce}$ and $f \in \mathcal{F}$ are terminal symbols and $\mathcal{F}$ is a countably infinite set of function symbols.

$$\mathsf{Msg} ::= atom \mid (\mathsf{Msg}, \mathsf{Msg}) \mid \{\mathsf{Msg}\}_{\mathsf{Msg}} \mid f\,(\mathsf{Msg})\,.$$

The term (m_1, m_2) denotes the pairing of messages m_1 and m_2. Further, $\{m_1\}_{m_2}$

stands for the encryption of m_1 with the key m_2. An agent's signature on a message is represented by the encryption of the message with the secret key of the agent. Finally, $f(m_1)$ indicates the output of the function f on the input m_1. Functions with multiple arguments can be represented through pairing of arguments.

Agents' cryptographic keys are denoted by the functions $pk\colon \mathsf{Agent} \to \mathsf{Msg}$, $sk\colon \mathsf{Agent} \to \mathsf{Msg}$ and $sh\colon \mathsf{Agent} \times \mathsf{Agent} \to \mathsf{Msg}$ that indicate the asymmetric public key of an agent, asymmetric secret key of an agent and the symmetric shared key of two agents, respectively. Lastly, the function $\_^{-1}\colon \mathsf{Msg} \to \mathsf{Msg}$ maps an encryption key onto the corresponding decryption key, and vice-versa.

The set $\mathcal{B} = \{sk, pk, sh, \_^{-1}\} \subseteq \mathcal{F}$ is the set of *basic functions* and its functions are assumed to satisfy that $sh(A,B) = sh(B,A)$, $pk(A)^{-1} = sk(A)$ and $sk(A)^{-1} = pk(A)$; for all $A, B \in \mathsf{Agent}$. In addition, we assume that $k \notin \{pk(A), sk(A)\}$ implies $k^{-1} = k$; for all $k \in \mathsf{Msg}$ and $A \in \mathsf{Agent}$. These assumptions represent the properties for symmetric and asymmetric encryption/decryption.

Events and Traces.
An event denotes an agent's action, such as sending or receiving a message, or an agent's security claim. We define the set of events Ev via the following grammar, for $A, B \in \mathsf{Agent}$.

$$\begin{aligned} \mathsf{Ev} ::= {} & \mathsf{send}_A\left(\mathsf{Msg}\right)\left[\mathsf{Msg}\right] \mid \mathsf{recv}_A\left(\mathsf{Msg}\right) \mid \\ & \mathsf{claim}_A\left(B, \mathsf{Ev}, \mathsf{Ev}\right). \end{aligned}$$

Given messages m_1 and m_2, and agents A and B, $\mathsf{send}_A\left(m_1\right)\left[m_2\right]$ indicates that A has sent the message m_1 and updated the agent's local state with the message m_2, and $\mathsf{recv}_A\left(m_1\right)$ means that A has received m_1. In the original model, claiming events have the form $\mathsf{claim}_A\left(B, d\right)$, where $d \in \mathbb{R}$ is a distance value. This allows an agent A to claim that another agent B is within a radius of length d, which is computed based on the round-trip time of a message exchange. We will make the message exchange explicit, and use $\mathsf{claim}_A\left(B, e_1, e_2\right)$ where e_1 and e_2 are the events used to compute the round-trip time and, by extension, the distance bound d.

We define the sets $\mathsf{Send}, \mathsf{Recv} \subseteq \mathsf{Ev}$ of all send and receive events, respectively. The function $actor\colon \mathsf{Ev} \to \mathsf{Agent}$ maps events onto their corresponding actor agent (i.e., the instance of A from the syntax). We extend this notation by using $actor\left(\tau\right)$, for a given trace τ, to refer to the set $\{actor\left(e\right) \mid \left(t, e\right) \in \tau\}$. We require for an event $\mathsf{claim}_A\left(B, e_1, e_2\right)$ that $actor\left(e_1\right) = actor\left(e_2\right) = A$.

A trace τ is a finite sequence of timed-events $\tau \in \left(\mathbb{R} \times \mathsf{Ev}\right)^*$, representing the execution of a protocol.

Agents' Knowledge.
As the trace evolves, agents may gain knowledge by receiving messages from other agents. At the beginning of a protocol execution, every agent is provided with an *initial knowledge* consisting of all agents' names and constants, his own nonces and secret keys, and all public keys. We use the function $init\colon \mathsf{Agent} \to \mathcal{P}\left(\mathsf{Msg}\right)$ to

$$\frac{m \in \mathit{init}\,(A)}{m \in \mathit{dm}_A\,(\tau)} \qquad \frac{(t, \mathsf{recv}_A\,(m)) \in \tau}{m \in \mathit{dm}_A\,(\tau)}$$

$$\frac{\begin{array}{c} m_1 \in \mathit{dm}_A\,(\tau) \\ m_2 \in \mathit{dm}_A\,(\tau) \end{array}}{(m_1, m_2) \in \mathit{dm}_A\,(\tau)} \qquad \frac{\begin{array}{c} m \in \mathit{dm}_A\,(\tau) \\ f \in \mathcal{F} \setminus \mathcal{B} \end{array}}{f(m) \in \mathit{dm}_A\,(\tau)}$$

$$\frac{\begin{array}{c} (m_1, m_2) \in \mathit{dm}_A\,(\tau) \\ i \in \{1, 2\} \end{array}}{m_i \in \mathit{dm}_A\,(\tau)} \qquad \frac{\begin{array}{c} m \in \mathit{dm}_A\,(\tau) \\ k \in \mathit{dm}_A\,(\tau) \end{array}}{\{m\}_k \in \mathit{dm}_A\,(\tau)}$$

$$\frac{\{m\}_k \in \mathit{dm}_A\,(\tau) \qquad k^{-1} \in \mathit{dm}_A\,(\tau)}{m \in \mathit{dm}_A\,(\tau)}$$

FIGURE 3.3: Rules for message deduction.

$$\frac{}{\epsilon \in \mathsf{Tr}\,(\mathcal{P})}\ \texttt{Start}$$

$$\frac{\begin{array}{c} \tau \in \mathsf{Tr}\,(\mathcal{P}) \qquad A \in \mathsf{Dishonest} \\ t \geq \mathit{maxt}(\tau) \qquad m \in \mathit{dm}_A\,(\tau) \end{array}}{\tau \cdot (t, \mathsf{send}_A\,(m)\,[]) \in \mathsf{Tr}\,(\mathcal{P})}\ \texttt{Int}$$

$$\frac{\begin{array}{c} \tau \in \mathsf{Tr}\,(\mathcal{P}) \qquad t \geq \mathit{maxt}(\tau) \\ (t', \mathsf{send}_A\,(m)\,[s]) \in \tau \\ t \geq t' + d\,(A, B)\,/\mathsf{c} \end{array}}{\tau \cdot (t, \mathsf{recv}_B\,(m)) \in \mathsf{Tr}\,(\mathcal{P})}\ \texttt{Net}$$

FIGURE 3.4: Start, Intruder and Network rules.

represent the initial knowledge of an agent:

$$\begin{aligned} \mathit{init}\,(A) = {} & \mathsf{Agent} \cup \mathsf{Const} \cup \mathsf{Nonce}_A \cup \{\mathit{sk}(A)\} \cup \\ & \{\mathit{pk}(B) \mid B \in \mathsf{Agent}\} \cup \{\mathit{sh}(A, B) \mid B \in \mathsf{Agent}\}, \end{aligned}$$

where Nonce_A denotes the set of nonces for a given agent $A \in \mathsf{Agent}$. We assume that $\{\mathsf{Nonce}_A | A \in \mathsf{Agent}\}$ forms a partition of the set Nonce.

Given an agent A and a trace τ, $\mathit{dm}_A(\tau)$ denotes the set of all *deducible messages* from a trace τ. This set is inductively defined by the rules in Figure 3.3.

Network and Intruder.
For a given protocol $\mathcal{P}$, the set of possible traces $\mathsf{Tr}\,(\mathcal{P})$ is inductively defined by the Start rule (`Start`), the Intruder rule (`Int`), the Network rule (`Net`) and the rules specifying the protocol. The Start, Intruder and Network rules are depicted in Figure 3.4.

The rules make use of the function $\mathit{maxt}\colon (\mathbb{R} \times \mathsf{Ev})^* \to \mathbb{R}$, defined as $\mathit{maxt}(\tau) = \max_{(t,e) \in \tau}\{t\}$, yields the latest time at which an event of τ occurred. The expression $d(A, B)$ gives the distance between two agents A and B based on an uninterpreted function $l\colon \mathsf{Agent} \to \mathbb{R}^3$, which associates each agent to a location in the real coordinate space $\mathbb{R}^3$. It is worth remarking that this interpretation of location assumes that agents are static, including dishonest agents.

The Start rule states that the empty trace ϵ is always part of the set of traces. The Intruder rule enables a dishonest agent, typically known as the *intruder* or the *adversary*, to inject (by sending) on the network any of his deducible messages. Finally, the Network rule establishes that a message m sent by and agent A can be received by an agent B without violating a time/location constraint that we describe in the next paragraph. This constraint is actually what makes this model particularly different from standard security models.

The Network rule also enforces that a message sent by an agent A and received by an agent B at times t' and t, respectively, must satisfy $d(A,B) \leq (t-t') \cdot \mathsf{c}$. In this way the physical law that messages cannot travel faster than the speed of light is made explicit. Observe that message loss is captured by *not* applying the network rule for a given sending event.

Protocol Specification.
A protocol is specified by a set of rules similar to the rules in Figure 3.4. Two syntactic restrictions (whose semantic interpretations will be given in Section 3.3) are applied:

- Neither the premises nor the conclusion of a protocol rule contain references to dishonest agents. This means that the behavior of dishonest agents is fully specified by the intruder rule.
- The premise of a protocol rule cannot contain events whose actors are not the same as the actor of the event in the premise of the rule. That is to say, agents are unaware of what other agents do. They can interact exclusively through the network rule.

Example 2.1 (Hancke and Kuhn's protocol)**.** *Figure 3.5 shows the formalization of Hancke and Kuhn's protocol [HK05] (see the representation in Figure 3.2). The first four rules in Figure 3.5 correspond to the four transmissions that take place in the protocol. The receiving events are derived from the network rule. The last rule from Figure 3.5 refers to the claim event for the property* secure distance-bounding *represented as "P is close" in Figure 3.2.*

The function used: $(\mathbb{R} \times \mathsf{Ev})^* \to \mathcal{P}\,(\mathsf{Msg})$ *defined as* $used(\tau) = \bigcup_{(t,e)\in\alpha} subt(cont\,(e))$, *is utilized to make sure that newly generated nonces are fresh, where subt*: $\mathsf{Msg} \to \mathcal{P}\,(\mathsf{Msg})$ *indicates the set of atomic messages that are sub-terms of a given message and cont*: $\mathsf{Ev} \to \mathsf{Msg}$ *gives us the content of a given event. The function subt is recursively defined as follows.*

$$subt(m) = \begin{cases} subt(m_1) \cup subt(m_2) & \text{if } m = (m_1, m_2) \\ subt(m_1) \cup subt(m_2) & \text{if } m = \{m_1\}_{m_2} \\ subt(m_1) & \text{if } m = f(m_1) \\ \{m\} & \text{otherwise .} \end{cases}$$

Example 2.1 also illustrates the purpose of the information in square brackets at the end of the send actions. In this case, it is implicitly used to define the notion of a session, by extending the send actions with the random nonces from that session.

$$\frac{\begin{array}{c}\tau \in \mathsf{Tr}\,(\mathcal{P}) \qquad V \in \mathsf{Honest} \qquad t \geq maxt(\tau) \\ N_V \in \mathsf{Nonce}_V \setminus used(\tau)\end{array}}{\tau \cdot (t, \mathsf{send}_V\,(N_V)\,[]) \in \mathsf{Tr}\,(\mathcal{P})}$$

$$\frac{\begin{array}{c}\tau \in \mathsf{Tr}\,(\mathcal{P}) \qquad P \in \mathsf{Honest} \qquad t \geq maxt(\tau) \\ (t', \mathsf{recv}_P\,(N_V)) \in \tau \qquad N_P \in \mathsf{Nonce}_P \setminus used(\tau)\end{array}}{\tau \cdot (t, \mathsf{send}_P\,(N_P)\,[N_V]) \in \mathsf{Tr}\,(\mathcal{P})}$$

$$\frac{\begin{array}{c}\tau \in \mathsf{Tr}\,(\mathcal{P}) \qquad V \in \mathsf{Honest} \qquad t \geq maxt(\tau) \\ (t', \mathsf{send}_V\,(N_V)\,[]) \in \tau \\ (t'', \mathsf{recv}_V\,(N_P)) \in \tau \qquad C \in \mathsf{Nonce}_V \setminus used(\tau)\end{array}}{\tau \cdot (t, \mathsf{send}_V\,(C)\,[N_V, N_P]) \in \mathsf{Tr}\,(\mathcal{P})}$$

$$\frac{\begin{array}{c}\tau \in \mathsf{Tr}\,(\mathcal{P}) \qquad P \in \mathsf{Honest} \qquad t \geq maxt(\tau) \\ (t', \mathsf{send}_P\,(N_P)\,[N_V]) \in \tau \qquad (t'', \mathsf{recv}_P\,(C)) \in \tau\end{array}}{\tau \cdot (t, \mathsf{send}_P\,(\mathsf{h}(sh(V,P), N_V, N_P, C))\,[]) \in \mathsf{Tr}\,(\mathcal{P})}$$

$$\frac{\begin{array}{c}\tau \in \mathsf{Tr}\,(\mathcal{P}) \qquad V \in \mathsf{Honest} \qquad tw \geq maxt(\tau) \\ u = \mathsf{send}_V\,(C)\,[N_V, N_P] \\ v = \mathsf{recv}_V\,(\mathsf{h}(sh(V,P), N_V, N_P, C)) \\ (tu, u) \in \tau \qquad (tv, v) \in \tau\end{array}}{\tau \cdot (tw, \mathsf{claim}_V\,(P, u, v)) \in \mathsf{Tr}\,(\mathcal{P})}$$

FIGURE 3.5: Formalization of Hancke and Kuhn's protocol.

Further, it is used to specify in which order the events of a session will have to be executed.

Security Properties.
The model uses claim events as placeholders to indicate where a security property needs to be satisfied. In this paper we focus on the property of *secure distance-bounding*, which is syntactically represented by claims of the form $\mathsf{claim}_V\,(P, u, v)$, where $V, P \in \mathsf{Agent}$ and $u, v \in \mathsf{Ev}$. A claim event $\mathsf{claim}_V\,(P, u, v)$ intuitively means that the agent V believes that the events u and v can be used to correctly compute an upper bound on his distance to P.

As the Intruder rule suggests, dishonest agents might disclose their secret key material by sending them out. This means that two dishonest provers might be indistinguishable to a legitimate verifier. In other words, a verifier V cannot securely decide whether a particular dishonest prover P is close, as another dishonest prover P' could have obtained all P's secrets and therefore P' can impersonate P. This leads to the following statement: V cannot claim that "P is close" but V can claim that "someone who knows P's secrets is close", at most. To capture this notion, we define the relation $\approx \subseteq \mathsf{Agent} \times \mathsf{Agent}$ as:

$$\approx = \{(A, A) \mid A \in \mathsf{Honest}\} \cup \mathsf{Dishonest} \times \mathsf{Dishonest}.$$

We use $A \not\approx B$ to indicate that $(A, B) \notin \approx$. By considering the relation $\approx$, we provide next a formal definition of *secure distance-bounding*.

Definition 2.2 (Secure distance-bounding)**.** *A protocol $\mathcal{P}$ satisfies* secure distance-bounding *if and only if:*

$$
\begin{aligned}
&\forall \tau \in \mathsf{Tr}\left(\mathcal{P}\right), V, P \in \mathsf{Agent}, u, v, w \in \mathsf{Ev}, tw \in \mathbb{R}:\\
&\quad (tw, w) \in \tau \wedge w = \mathsf{claim}_V\left(P, u, v\right) \implies\\
&\qquad \exists tu, tv \in \mathbb{R}, P' \in actor\left(\tau\right):\\
&\qquad\quad (tu, u) \in \tau \wedge (tv, v) \in \tau \wedge P \approx \P' \wedge\\
&\qquad\quad d(V, P') \leq \frac{\mathsf{c}}{2}\left(tv - tu\right).
\end{aligned}
\tag{3.1}
$$

A distance-bounding protocol is secure if the occurrence of a claim event $\mathsf{claim}_V\left(P, u, v\right)$ in a protocol execution implies that V has correctly computed an upper bound on his distance to either P (if P is honest) or some dishonest agent P' (if P is dishonest).

Our definition of secure distance-bounding slightly differs from the original one provided by Basin et al., but the difference is merely notational, allowing us to cleanly formulate our main result. Note that claim events are formulated in such a way that they relate to a single challenge/response pair. Thus, similar to Basin et al's approach, we will need to include several claim events if the fast phase cannot be abstracted to a single challenge/response pair.

3.3 Distance Bounding Security

In this section we use the specification model of Basin et al. as a starting point in order to construct a new framework. This framework has the advantage of considering only time only in a discrete way – comparing only the order of events and not their timings. We first define a set of well-formedness rules for specifying distance bounding protocols using this language. We produce a new security definition for distance bounding in this setting, and finally prove that under our assumption set this definition is in fact equivalent to the secure distance bounding security goal of Basin et al. .

An important characteristic of the approach presented previous section is that security protocols are specified using the same type of derivation rules as used for the definition of the general semantics of the system. Consequently, protocol specifications are much more liberal than in comparable formal approaches that define a domain specific language for the definition of protocols. Alternative approaches, like the one by Cremers and Mauw [CM12] provide a dedicated protocol specification language and impose syntactical or semantical constraints to prevent users from specifying meaningless or simply undesired protocols.

An example of a protocol rule that may be considered undesirable is the one in Figure 3.6. It specifies that after reception of the message Hello at time t, agent A sends a message Hi back at time $t-1$. This is clearly an infringement of a time consistency property, because it leads to the trace $(1, \mathsf{recv}_A\left(\mathsf{Hello}\right)) \cdot (0, \mathsf{send}_A\left(\mathsf{Hi}\right)[\,])$.

The solution proposed by Basin et al. is to consider only those traces that have non-decreasing timestamps for subsequent events. In our approach we will take this line

$$\frac{\begin{array}{c}\tau \in \mathsf{Tr}\left(\mathcal{P}\right) \qquad A \in \mathsf{Honest} \\ \mathsf{Hello}, \mathsf{Hi} \in \mathsf{Const} \qquad \left(t, \mathsf{recv}_A\left(\mathsf{Hello}\right)\right) \in \tau\end{array}}{\tau \cdot \left(t-1, \mathsf{send}_A\left(\mathsf{Hi}\right)[]\right) \in \mathsf{Tr}\left(\mathcal{P}\right)}$$

FIGURE 3.6: A protocol rule that leads to incorrect traces.

of reasoning one step further, in that we will define a number of assumptions that a proper semantics should satisfy and that are sufficient to derive our main result. We will argue that these properties are valid for the semantics from the previous section, under the assumption of a class of "reasonable" protocol specifications.

Basic Properties of the Semantics

In line with the previous example, the first property that we formulate is *time consistency*. It states that events of a trace are timestamped in non-decreasing order.

Property 3.1 (Time consistency). *A protocol P satisfies* time consistency *if for every trace* $\tau = (t_1, e_1) \cdots (t_n, e_n) \in \mathsf{Tr}\left(\mathcal{P}\right)$*, it holds that* $t_1 \leq \cdots \leq t_n$*.*

The second property that we consider is *speed-of-light consistency*. It states that all traces satisfy the restrictions of the speed of light. In particular, this means that the time between the sending of a message by agent A and the reception of this message by agent B must be equal to or larger than the distance between the two agents divided by the speed of light.

Because this definition requires the correspondence between a send event and its related receive event, we define the relation $\rightsquigarrow \subseteq \mathsf{Send} \times \mathsf{Recv}$ as follows:

$$\rightsquigarrow = \left\{(e, e') \in \mathsf{Send} \times \mathsf{Recv} \mid cont\left(e\right) = cont\left(e'\right)\right\}.$$

The relation $\rightsquigarrow$ defines whether an event e' is a receive event that could have occurred as consequence of the send event e. As followed from its formulation, $\rightsquigarrow$ is not a one-to-one relation. This lines up with the fact that it does not need to be the case that there is a unique send event that triggers a given receive event. In the semantics above, the relation $\rightsquigarrow$ can be easily derived from the application of the Network rule in Figure 3.4.

Property 3.2 (Speed-of-light consistency). *A protocol P satisfies* speed-of-light consistency *if for every trace* $\tau = (t_1, e_1) \cdots (t_n, e_n) \in \mathsf{Tr}\left(\mathcal{P}\right)$ *the following holds: for all* $j \in \{2, \ldots, n\}$*, if* $e_j \in \mathsf{Recv}$*, then there exists* $i \in \{1, \ldots, j-1\}$ *such that* $e_i \rightsquigarrow e_j$ *and* $t_j - t_i \geq d\left(e_i, e_j\right)/\mathsf{c}$*.*

Even though we define Properties 3.1 and 3.2 for protocols, we will also use them in relation to traces. Thus we will talk about time consistency and speed-of-light consistency of a given trace, with the obvious interpretation.

The formulation of the remaining properties requires the notion of *untimed* traces, or simply a sequence of (untimed) events. The projection $\pi(\alpha)$ of a trace $\tau =$

$(t_1, e_1) \cdots (t_n, e_n) \in (\mathbb{R} \times \mathsf{Ev})^*$ is the untimed trace $e_1 \cdots e_n \in \mathsf{Ev}^*$. Likewise, the projection of the set of traces is defined as $\pi(\mathsf{Tr}\,(\mathcal{P})) = \{\pi(\tau) \mid \tau \in \mathsf{Tr}\,(\mathcal{P})\}$. We say that two traces α and β are content-wise equal, denoted $\tau \sim \tau'$, if $\pi(\alpha) = \pi(\beta)$.

The third property states that traces are built inductively by appending events.

Property 3.3 (Prefix-closure). *A protocol P is* prefix-closed *if for every $\gamma = \sigma \cdot e \in \pi(\mathsf{Tr}\,(\mathcal{P}))$, it holds that $\sigma \in \pi(\mathsf{Tr}\,(\mathcal{P}))$.*

The fourth property expresses that the notion of time is only used for the verifier's decision-making process on whether the prover passed the protocol or not. Time will not be used to make any other decision during the execution of the protocol (e.g., to take a different branch depending on the time). This means that any trace can be *retimed*, as long as it still satisfies time consistency and speed-of-light consistency.

Property 3.4 (Time-unawareness). *A protocol P is* time-unaware *if for every trace $\tau \in \mathsf{Tr}\,(\mathcal{P})$ the following holds: for all time consistent and speed-of-light consistent traces $\beta \in (\mathbb{R} \times \mathsf{Ev})^*$, $\tau \sim \beta$ implies $\beta \in \mathsf{Tr}\,(\mathcal{P})$.*

Different agents only interact through the network via sending and receiving events. As a consequence, a non-receive action can only be triggered by the actor agent's own preceding actions and another agent's actions in between can be disregarded or delayed. This leads to the fifth property, *locally-enabled events*. We use untimed events in order to easily express that the resulting trace $\sigma \cdot e'$ might require a re-timing of event e'.

Property 3.5 (Locally-enabled events). *A protocol P satisfies* locally-enabled events *if for every $\gamma = \sigma \cdot e \cdot e' \in \pi(\mathsf{Tr}\,(\mathcal{P}))$ such that $e' \notin \mathsf{Recv}$ and $actor\,(e) \neq actor\,(e')$, it holds that $\sigma \cdot e' \in \pi(\mathsf{Tr}\,(\mathcal{P}))$.*

The *locally-enabled events* property allows non-receive events to move left in a trace under specific conditions. The next property expresses when a receive event can be appended to a trace.

Property 3.6 (Transmission-enabled events). *A protocol P satisfies* transmission-enabled events *if for every $\gamma = \sigma \cdot e \in \pi(\mathsf{Tr}\,(\mathcal{P}))$ and every $e' \in \mathsf{Recv}$ such that $e \rightsquigarrow e'$, it holds that $\gamma \cdot e' \in \pi(\mathsf{Tr}\,(\mathcal{P}))$.*

Agents in the model are universally quantified. Therefore, in a given trace we can replace an agent by another and still obtain a valid trace, as long as both agents are either honest or dishonest. An agent substitution is denoted by $A \mapsto B$ where A and B are agents. Given a message $m \in \mathsf{Msg}$, $m[A \mapsto B]$ represents the substitution of all occurrences in m of A by B. We extend substitutions onto events and traces in the obvious way.

Property 3.7 (Substitution-closure). *A protocol P is* substitution-closed *if for every $\sigma \in \pi(\mathsf{Tr}\,(\mathcal{P}))$ and every $A, B \in \mathsf{Agent}$ such that $\{A, B\} \subseteq \mathsf{Honest}$ or $\{A, B\} \subseteq \mathsf{Dishonest}$, it holds that $\sigma[A \mapsto B] \in \pi(\mathsf{Tr}\,(\mathcal{P}))$.*

Observe that $e \rightsquigarrow e'$ implies $e[A \mapsto B] \rightsquigarrow e'[A \mapsto B]$. We say that a protocol is *well-formed* if it satisfies the seven properties mentioned above.

$$\frac{\begin{array}{c}\tau \in \mathsf{Tr}\left(\mathcal{P}\right) \quad A \in \mathsf{Honest} \quad t \geq \mathit{maxt}(\tau) \\ \mathit{prem}_1 \quad \mathit{prem}_2 \quad \cdots \quad \mathit{prem}_p \\ (t_1, e_1) \in \tau \quad (t_2, e_2) \in \tau \quad \cdots \quad (t_q, e_q) \in \tau\end{array}}{\tau \cdot (t, e) \in \mathsf{Tr}\left(\mathcal{P}\right)}$$

FIGURE 3.7: Prototype of rules that lead to well-formed protocols.

Validity of the Properties

As stated in the beginning of the current section, the mechanism for specifying protocols is too liberal to ensure the well-formedness properties. Therefore, we use a restricted format for protocol rules inspired by the example specification of Hancke and Kuhn's protocol from Figure 3.5. The restricted format is specified by the rule prototype in Figure 3.7. We additionally require that $p + q > 0$, $A = \mathit{actor}\left(e\right) = \mathit{actor}\left(e_1\right) = \mathit{actor}\left(e_2\right) = \cdots = \mathit{actor}\left(e_q\right)$, $e \notin \mathsf{Recv}$ and none of the premises prem_i involve any of the timestamps t_j or t. Even though the protocol format is restricted with respect to the liberal format specified by Basin et al., we conjecture that it is sufficiently expressive to specify all relevant protocols from literature. We validate this by specifying a number of protocols in this format and analysing them with our implementation (see Section 3.3).

Together with the Start, Intruder and Network rules from Figure 3.4, the restricted format implies well-formedness of the specified protocol. We will briefly argue the validity of the properties under this restricted format. Time consistency follows from the precondition $t \geq \mathit{maxt}(\tau)$ in the Intruder and Network rules and in the restricted protocol rule. Speed-of-light consistency follows from the precondition $t \geq t' + d\left(A, B\right) / \mathsf{c}$ in the Network rule and the requirement that $e \notin \mathsf{Recv}$ in the restricted protocol rule. *Prefix-closure* follows from the precondition $\tau \in \mathsf{Tr}\left(\mathcal{P}\right)$ in all rules, together with the fact that the conclusion extends this trace with a single event. *Time-unawareness* follows from the fact that in the construction of the traces any time $t \geq \mathit{maxt}(\tau)$ is allowed for the next event, as long as *speed-of-light consistency* is satisfied. The property *locally-enabled events* follows from the requirement that a rule only concerns a single actor. The *transmission-enabled events* property follows directly from the Network rule. *Substitution-closure* expresses the (implicit) universal quantification over agents' names in all rules.

Causality-Based Verification

Given the definitions and properties from the previous sections, we can now formulate the notion of *causality-based secure distance-bounding* and prove that it is equivalent to the original definition of *secure distance-bounding* from Definition 2.2. The main feature of this new formulation is that it is causality-based, i.e., it only takes into account the relative occurrence of events, while ignoring the actual timestamps of the events and agents' locations.

This new formulation strongly relates to authentication properties, such as *aliveness* (see [CM12]). It states that for every claim that prover P is in the vicinity of verifier

V, due to a challenge event u and the reception of its corresponding response event v in the fast phase, agent P (or a conspiring agent, if P is dishonest) must have been active in between these two events. The main difference with Definition 2.2 is that we require the prover to be active, instead of measuring the time between u and v.

Definition 3.8 (Causality-based secure distance-bounding)**.** *A well-formed protocol P satisfies* causality-based secure distance-bounding *if and only if:*

$$\begin{aligned}&\forall\sigma \in \pi(\mathsf{Tr}\,(\mathcal{P})), V, P \in \mathsf{Agent}, u, v \in \mathsf{Ev}\colon\\ &\mathsf{claim}_V\,(P,u,v) \in \sigma \implies \exists i,j,k \in \{1,\ldots,|\sigma|\}\colon\\ &i<j<k \wedge u=\sigma_i \wedge v=\sigma_k \wedge P \approx actor\left(\sigma_j\right).\end{aligned} \tag{3.2}$$

In Definition 3.8 we formalize our causality-based notion of secure distance-bounding. This formulation impacts only the security analysis in the design stage. It does not affect the runtime behavior of the agents executing the protocol. In particular, the verifying agent still has to measure the round-trip time of the message exchanges in the fast phase.

In the remainder of this section, we develop the proof that the causality-based definition is equivalent to the secure distance-bounding property from Definition 2.2. To do so, we first present a few lemmas that follow from the basic properties of the semantic domain described in Section 3.3. They will prove useful when deriving our main result.

Given two events $e, e' \in \mathsf{Ev}$, we use $d\,(e,e')\,/\mathsf{c}$ as a shorthand notation for $d\,(actor\,(e)\,, actor\,(e'))\,/\mathsf{c}$. Also, we say that two timed-events $(t,e), (t',e') \in \mathbb{R}\times\mathsf{Ev}$ satisfy the time/location constraint if $|t'-t| \geq d\,(e,e')\,/\mathsf{c}$. For example, all pairs of events used in the network rule satisfy this constraint. In addition, we define the predicate $\psi\,(\tau)$, where τ is a trace, that holds if all pairs of consecutive timed-events on τ satisfy the time/location constraint. Likewise, we say that timed-trace β is a subsequence of a timed-trace $\tau = (t_1,e_1)\cdots(t_n,e_n)$, denoted by $\beta \sqsubseteq \tau$, if there exist $m \in \{0,\ldots,n\}$ and $\{w_1,\ldots,w_m\} \subseteq \{1,\ldots,n\}$ such that $w_1 < \cdots < w_m$ and $\beta = (t_{w_1}, e_{w_1})\cdots(t_{w_m}, e_{w_m})$.

In Lemma 3.9 below, we demonstrate that for any well-formed protocol P, any valid timed-trace $\tau\cdot(t,e) \in \mathsf{Tr}\,(\mathcal{P})$ must contain a subsequence β that is also a valid trace in P, and contains (t,e) and $\psi\,(\beta)$. We use $|.|$ to denote the length of a (timed or not) trace, in terms of the number of events.

Lemma 3.9. *Then the following holds:*

$$\begin{aligned}&\forall\tau \in \mathsf{Tr}\,(\mathcal{P})\,, (t,e) \in \mathbb{R}\times\mathsf{Ev}\colon \tau\cdot(t,e) \in \mathsf{Tr}\,(\mathcal{P}) \implies\\ &\qquad\exists\beta \in \mathsf{Tr}\,(\mathcal{P})\,:\,(t,e) \in \beta \wedge \beta \sqsubseteq \tau\cdot(t,e) \wedge \psi\,(\beta)\,.\end{aligned}$$

Proof. We will proceed by induction over $|\tau|$. The base case $|\tau| = 0$ trivially holds by setting $\beta = (t,e)$. So, let $n \in \mathbb{N}\setminus\{0\}$ and assume by the induction hypothesis that the lemma holds for all $\tau \in \mathsf{Tr}\,(\mathcal{P})$ with $|\tau| < n$. Now, let $\tau = (t_1,e_1)\cdots(t_n,e_n) \in \mathsf{Tr}\,(\mathcal{P})$ and $(t,e) \in \mathbb{R}\times\mathsf{Ev}$ such that $\gamma = \tau\cdot(t,e) \in \mathsf{Tr}\,(\mathcal{P})$. Let us analyse the two cases:

Case 1 ($e \in \mathsf{Recv}$**)** From Property 3.2 we have that there exists $i \in \{1,\ldots,n\}$ such that $e_i \rightsquigarrow e$ and $t - t_i \geq d\,(e_i,e)\,/\mathsf{c}$. Consider $\tau' = (t_1,e_1)\cdots(t_{i-1},e_{i-1})$. Then, from the induction hypothesis (given that $|\tau'| = i - 1 < n$ and $\tau' \in \mathsf{Tr}(\mathcal{P})$ due to Properties 3.3 and 3.4) it follows that there exists $\beta' \in \mathsf{Tr}(\mathcal{P})$ with $(t_i, e_i) \in \beta'$ such that $\beta' \sqsubseteq \tau'$ and $\psi(\beta')$. Thus, $\psi(\beta')$ along with $t - t_i \geq d(e_i, e)/\mathsf{c}$ give us that $\psi(\beta' \cdot (t, e))$ and $\beta' \cdot (t, e)$ is time and speed-of-light consistent.

Now, from Property 3.6 we derive $\pi(\beta') \cdot e \in \pi(\mathsf{Tr}(\mathcal{P}))$. On the other hand, $\beta' \cdot (t, e) \sim \beta''$ for some $\beta'' \in \mathsf{Tr}(\mathcal{P})$ such that $\pi(\beta') \cdot e = \pi(\beta'')$. Finally, Property 3.4 gives us $\beta' \cdot (t, e) \in \mathsf{Tr}(\mathcal{P})$.

Case 2 ($e \notin \mathsf{Recv}$**)** Let i be the largest number in $\{1, \ldots, n\}$ such that $\mathit{actor}(e_i) = \mathit{actor}(e)$. If i does not exist, then from Property 3.5 we obtain that $e \in \pi(\mathsf{Tr}(\mathcal{P}))$ and therefore $(t', e) \in \mathsf{Tr}(\mathcal{P})$ for some $t' \in \mathbb{R}$. Hence, as (t, e) is time and speed-of-light consistent, Property 3.4 gives us $(t, e) \in \mathsf{Tr}(\mathcal{P})$ as $(t, e) \sim (t', e)$. Further, $\psi((t, e))$ trivially holds, which leaves us with the remaining case in which i exists.

Let $\tau' = (t_1, e_1) \cdots (t_i, e_i)$. Then, from the induction hypothesis (given that $|\tau'| = i - 1 < n$ and $\tau' \in \mathsf{Tr}(\mathcal{P})$ due to Properties 3.3 and 3.4) it follows that there exists $\beta' \in \mathsf{Tr}(\mathcal{P})$ with $(t_i, e_i) \in \beta'$ such that $\beta' \sqsubseteq \tau'$ and $\psi(\beta')$. Thus, $\psi(\beta')$ along with $t - t_i \geq d(e_i, e)/\mathsf{c} = 0$ give us that $\psi(\beta' \cdot (t, e))$ and $\beta' \cdot (t, e)$ is time and speed-of-light consistent.

Now, from Property 3.6 we derive $\pi(\beta') \cdot e \in \pi(\mathsf{Tr}(\mathcal{P}))$. On the other hand, $\beta' \cdot (t, e) \sim \beta''$ for some $\beta'' \in \mathsf{Tr}(\mathcal{P})$ such that $\pi(\beta'') = \pi(\beta') \cdot e$. Finally, Property 3.4 gives us $\beta' \cdot (t, e) \in \mathsf{Tr}(\mathcal{P})$. □

Lemma 3.10 below is an extension of Lemma 3.9. It states that if a valid trace τ satisfies $\psi(\tau)$, then not only any pair of consecutive events in τ satisfy the time/location constraint but also any pair of events in τ. The proof follows from the application of the triangle inequality $d(e, e')/\mathsf{c} + d(e', e'')/\mathsf{c} \geq d(e, e'')/\mathsf{c}$, for all $e, e', e'' \in \mathsf{Ev}$, given that d models physical distances.

Lemma 3.10. *Suppose we have $\tau \in \mathsf{Tr}(\mathcal{P})$ such that $\psi(\tau)$. Then for all $(t, e), (t', e') \in \tau$ it holds that $|t - t'| \geq d(e, e')/\mathsf{c}$.*

Proof. Let $\tau = (t_1, e_1) \cdots (t_n, e_n)$ and $i, j \in \{1, \ldots, n\}$. Assume without loss of generality that $i < j$. Given that $\psi(\tau)$ we have that $t_x - t_{x-1} \geq d(e_{x-1}, e_x)/\mathsf{c}$ for all $x \in \{i + 1, \ldots, j\}$. Hence,

$$\begin{aligned} t_j - t_i &= (t_j - t_{j-1}) + (t_{j-1} - t_{j-2}) + \cdots + (t_{i+1} - t_i) \\ &\geq d(e_i, e_{i+1})/\mathsf{c} + d(e_{i+1}, e_{i+2})/\mathsf{c} + \cdots + \\ &\quad d(e_{j-1}, e_j)/\mathsf{c}. \end{aligned} \tag{3.3}$$

Thus, by applying the triangle inequality in Equation 3.3 above, we obtain $t_j - t_i \geq d(e_i, e_j)/\mathsf{c}$. □

The last lemma of this section concerns agent substitutions. We extend Property 3.7 from the set of untimed-traces $\pi(\mathsf{Tr}(\mathcal{P}))$ of a given protocol P to the set of timed-traces $\mathsf{Tr}(\mathcal{P})$. The lemma proves that, given a protocol's valid trace $\tau = (t_1, e_1) \cdots (t_n, e_n)$, it is possible to replace an agent A by another agent B (under certain conditions described in the lemma) to obtain another valid trace $\tau' = (t'_1, e'_1) \cdots (t'_n, e'_n)$ such that the difference between t'_i and t_i only depends on the number of events before the i-th event on τ that were executed by A. Consequently, the time-difference between two events of τ where A does not act is equal to the time-difference between the corresponding events of τ'. This is actually a strong result because it implicitly shows that event-intervals where the prover does not act cannot be used to securely upper-bound the prover-to-verifier distance.

Lemma 3.11. *and* $\tau = (t_1, e_1) \cdots (t_n, e_n) \in \mathsf{Tr}(\mathcal{P})$. *Let* $A \in \mathit{actor}(\tau)$, $B \in \mathsf{Agent} \setminus \mathit{actor}(\tau)$ *such that either* $\{A, B\} \subseteq \mathsf{Honest}$ *or* $\{A, B\} \subseteq \mathsf{Dishonest}$. *Then there exists* $\mu \in \mathbb{R}_{\geq 0}$ *such that* $\tau' = (t'_1, e'_1) \cdots (t'_n, e'_n) \in \mathsf{Tr}(\mathcal{P})$ *where for all* $i \in \{1, \ldots, n\}$ *it holds that:*

$$\begin{aligned} & e'_i = e_i[A \mapsto B] \text{ and } t'_i = t_i + \mu \cdot q_i, \text{ where} \\ & q_i = \left|\{j \in \{1, \ldots, i-1\} \mid \mathit{actor}(e_j) = A\}\right| + s_i, \text{ and} \\ & s_i = 1 \text{ if } (A = \mathit{actor}(e_i) \wedge e_i \in \mathsf{Recv}), \text{ or otherw. } s_i = 0. \end{aligned}$$

Proof. Consider the set $R = \{B\} \cup \mathit{actor}(\tau)$ and $\mu = \max_{X \in R}\{d(A, X) / \mathsf{c}\}$. We will proceed to prove that $\tau' \in \mathsf{Tr}(\mathcal{P})$. To do so we will first prove time and speed-of-light consistency for τ'.

Time consistency For all $i \in \{1, \ldots, n-1\}$, we have that $q_{i+1} \geq q_i$ and therefore $t'_{i+1} - t'_i = t_{i+1} - t_i + \mu \cdot (q_{i+1} - q_i) \geq t_{i+1} - t_i \geq 0$.

Speed-of-light consistency Let $j \in \{1, \ldots, n\}$ such that $e_j \in \mathsf{Recv}$. Also, as τ is speed-of-light consistent, we derive that there exists $i < j$ such that $e_i \rightsquigarrow e_j$ and $t_j - t_i \geq d(e_i, e_j) / \mathsf{c}$. Hence, given that $e'_i \rightsquigarrow e'_j$, it becomes sufficient to prove that $t'_j - t'_i \geq d\left(e'_i, e'_j\right) / \mathsf{c}$. Let us consider the three cases:

1. $A = \mathit{actor}(e_i)$. In this case $q_j \geq q_i + 1$ because $e_i \notin \mathsf{Recv}$. Therefore $t'_j - t'_i \geq t_j - t_i + \mu \geq d\left(e'_i, e'_j\right) / \mathsf{c}$ as $\mu \geq d\left(e'_i, e'_j\right) / \mathsf{c}$.
2. $A \neq \mathit{actor}(e_i)$ and $A = \mathit{actor}(e_j)$. In this case we have again $q_j \geq q_i + 1$ as $e_j \in \mathsf{Recv}$, and it follows analogously to the previous case.
3. $A \notin \{\mathit{actor}(e_i), \mathit{actor}(e_j)\}$. This case gives us $\mathit{actor}(e_i) = \mathit{actor}(e'_i)$ and $\mathit{actor}(e_j) = \mathit{actor}(e_j)$. Thus, $d(e_i, e_j) / \mathsf{c} = d\left(e'_i, e'_j\right) / \mathsf{c}$ and therefore $t'_j - t'_i = t_j - t_i + \mu \cdot (q_j - q_i) \geq t_j - t_i \geq d(e_i, e_j) / \mathsf{c} = d\left(e'_i, e'_j\right) / \mathsf{c}$.

Thus, τ' is time consistent and speed-of-light consistent. Consider now $\sigma = \pi(\tau)$. From Property 3.7 we have that $\sigma[A \mapsto B] \in \pi(\mathsf{Tr}(\mathcal{P}))$. Therefore, there exists $\gamma \in \mathsf{Tr}(\mathcal{P})$ such that $\pi(\gamma) = \sigma[A \mapsto B]$. Finally, given that $\gamma \sim \tau'$, from Property 3.4 we obtain $\tau' \in \mathsf{Tr}(\mathcal{P})$. □

Theorem 3.12. *A well-formed protocol P satisfies* secure distance-bounding *(Definition 2.2) if and only if P satisfies* causality-based secure distance-bounding *(Definition 3.8).*

Proof. We will proceed by proving *Sufficiency* (i.e., Equation 3.1 ⇒ Equation 3.2) and *Necessity* (i.e., Equation 3.2 ⇒ Equation 3.1):

Sufficiency Assume Equation 3.1 holds and Equation 3.2 does not. Our goal is to reach a contradiction. The statement that Equation 3.2 does not hold is equivalent to stating that there exist $\sigma = \sigma_1 \cdots \sigma_n \in \pi(\mathsf{Tr}(\mathcal{P}))$, $V, P \in \mathsf{Agent}$, $u, v \in \mathsf{Ev}$ and $l \in \{1, \ldots, n\}$ such that $\sigma_l = \mathsf{claim}_V(P, u, v)$ and:

$$\begin{aligned} &\forall i, j, k \in \{1, \ldots, n\}: \\ &\quad u = \sigma_i \wedge v = \sigma_k \wedge i < j < k \implies P \not\approx \mathit{actor}(\sigma_j). \end{aligned} \tag{3.4}$$

Consider now the following sets:

$$\begin{aligned} IK &= \{(i,k) \in \mathbb{N} \times \mathbb{N} \mid \sigma_i = u \wedge \sigma_k = v\}, \\ J &= \{j \in \mathbb{N} \mid \exists (i,k) \in IK\colon i < j < k\}, \\ \{G_1, \ldots, G_g\} &= \{G \in \mathit{actor}(\sigma) \mid P \approx G\}. \end{aligned}$$

If P is honest, then the set $\{G_1, \ldots, G_g\}$ consists of the singleton $\{P\}$, otherwise it contains all dishonest agents acting in σ.

Let $\mathit{Eve}, \mathit{Charlie} \in \mathsf{Agent} \setminus \mathit{actor}(\sigma)$ be two different agents such that $\{P, \mathit{Eve}, \mathit{Charlie}\} \subseteq \mathsf{Honest}$ or $\{P, \mathit{Eve}, \mathit{Charlie}\} \subseteq \mathsf{Dishonest}$.

Consider the sequence of traces $\sigma^1, \ldots, \sigma^{g+1} \in \pi(\mathsf{Tr}(\mathcal{P}))$ such that $\sigma^1 = \sigma$ and for all $i \in \{1, \ldots, g\}$, $\sigma^{i+1} = \sigma^i[G_i \mapsto \mathit{Eve}]$. The fact that $\sigma^1, \ldots, \sigma^{g+1} \in \pi(\mathsf{Tr}(\mathcal{P}))$ follows from the *substitution-closedness* property. Hence, let $e_1 \cdots e_n = \sigma^{g+1}$, i.e., the trace resulting from σ after the successive substitutions of all agents $G_1, \ldots, G_g$ by E. Therefore $N \subseteq \mathsf{Agent}$ exists such that:

$$\begin{aligned} \mathit{actor}(e_1 \cdots e_n) = \{V, E\} \cup N \text{ and} \\ \forall E \in N\colon \mathit{Eve} \not\approx E. \end{aligned} \tag{3.5}$$

Let $t_1, \ldots, t_n \in \mathbb{R}$ such that $(t_1, e_1) \cdots (t_n, e_n) \in \mathsf{Tr}(\mathcal{P})$. Observe that the t_i's exist because $e_1 \cdots e_n \in \pi(\mathsf{Tr}(\mathcal{P}))$. Hence, from Equations 3.1 and 3.5 and given that $e_l = \mathsf{claim}_V(\mathit{Eve}, e_i, e_k)$ for some $(i,k) \in IK$, we derive that $\delta \in \mathbb{R}_{\geq 0}$ exists such that:

$$d(V, E) + \delta = \frac{\mathsf{c}}{2} \max_{(i,k) \in IK} \{t_k - t_i\}. \tag{3.6}$$

From Lemma 3.11 we have that there exist $\mu \in \mathbb{R}_{\geq 0}$, $(t'_1, e'_1) \cdots (t'_n, e'_n) \in \mathsf{Tr}(\mathcal{P})$ and $q_1, \ldots, q_n \in \mathbb{N}$ such that for all $i \in \{1, \ldots, n\}$, $e'_i = e_i[\mathit{Eve} \mapsto \mathit{Charlie}]$ and $t'_i = t_i + \mu \cdot q_i$ (see the construction of the q_i's in Lemma 3.11). On the other hand, from Equation 3.4 we have that $\forall j \in J\colon E \neq \mathit{actor}(e_j)$. Therefore

$$\forall (i,k) \in IK\colon t'_k - t'_i = t_k - t_i. \tag{3.7}$$

Furthermore, given that $\{Eve, Charlie\} \subseteq \mathsf{Honest}$ or $\{Eve, Charlie\} \subseteq \mathsf{Dishonest}$, it holds that:

$$\begin{aligned} actor\left(e'_1 \cdots e'_n\right) &= \{V, Charlie\} \cup N \text{ and} \\ \forall C &\in N\colon Charlie \not\approx C. \end{aligned} \tag{3.8}$$

Again, $e'_l = \mathsf{claim}_V\left(Charlie, e'_i, e'_k\right)$ for some $(i,k) \in IK$, so from Equations 3.1 and 3.8 we derive:

$$d(V, Charlie) \leq \frac{\mathsf{c}}{2} \max_{(i,k)\in IK} \{t'_k - t'_i\}. \tag{3.9}$$

Finally, from Equations 3.6, 3.7 and 3.9 we derive that $d(V, Charlie) \leq d(V, E) + \delta$. This is a contradiction, as δ does not depend on *Charlie* who is an *arbitrary* agent from the same set as P in Honest or $\mathsf{Dishonest}$. Therefore we can always find *Charlie* such that his distance to V is larger than $d(V, E) + \delta$.

Necessity Assume Equation 3.2 holds. We will prove that Equation 3.1 holds as well. Let $\sigma \in \pi(\mathsf{Tr}\left(\mathcal{P}\right))$ and $\tau \in \mathsf{Tr}\left(\mathcal{P}\right)$ such that $\sigma = \pi(\tau)$. Let $V, P \in \mathsf{Agent}$, $u, v, w \in \mathsf{Ev}$ and $tw \in \mathbb{R}$ such that $(tw, w) \in \tau$ and $w = \mathsf{claim}_V\left(P, u, v\right)$. Also, let $\beta \in \mathsf{Tr}\left(\mathcal{P}\right)$ such that $\beta \sqsubseteq \tau$, $(tw, w) \in \beta$ and $\psi\left(\beta\right)$. Observe that β exists because of Lemma 3.9.

From Equation 3.2 and given that $\pi(\beta) \in \pi(\mathsf{Tr}\left(\mathcal{P}\right))$, we have that there exist $tu', tv' \in \mathbb{R}$, $P' \in \mathsf{Agent}$ and $(t, e) \in \beta$ such that $P' = actor\left(e\right)$, $tu' \leq t \leq tv'$, $(tu', u) \in \beta$, $(tv', v) \in \beta$ and $P \approx P'$. Hence, Lemma 3.10 gives us:

$$\begin{aligned} tv' - tu' = (tv' - t) + (t - tu') &\geq \frac{d(e, v) + d(u, e)}{\mathsf{c}} \\ &= 2d\left(V, P'\right)/\mathsf{c}, \end{aligned}$$

which proves Equation 3.1 as $(tu', u) \in \beta \sqsubseteq \tau$, $(tv', v) \in \beta \sqsubseteq \tau$ and $(tw, w) \in \beta \sqsubseteq \tau$. □

The result obtained from Theorem 3.12 means that, within the semantic domain described in Section 3.3, the secure distance-bounding property can be verified by simply analysing the ordering of events in the traces. Therefore, the notions of time and location are indeed unnecessary for the symbolic verification of distance-bounding protocols.

3.4 Verifying Distance Bounding Protocols

We implemented the causality-based definition of secure distance-bounding in the Tamarin [MSCB13] tool. This allowed us to automatically verify the (in)security of multiple distance-bounding protocols and their variations. In this section we discuss some of the considerations made when performing this translation. Later, In Section 4.6, the full results of the Tamarin analysis of a range of protocols from the literature is presented.

When considering distance hijacking, our security analysis is consistent with the analysis performed by Cremers et al. in [CRSC12]. That is to say, in general protocols based on the Brand and Chaum's design are vulnerable to this type of attacks, whereas those based on Hancke and Kuhn's are not. In addition, we observed that protocols following Hancke and Kuhn's approach seem to be resistant not only to distance hijacking but also to mafia and distance frauds.

Tamarin Implementations

Our method compares well with the Isabelle/HOL implementation of Basin et al. While our approach is fully automatic, proving a protocol insecure with Isabelle/HOL requires user-assistance to prove the existence of an attack trace. In addition, the code complexity of a protocol when implemented in Isabelle/HOL tends to be much larger. For example, the implementation of Brands and Chaum's protocol consists of 185 lines of Tamarin code, whilst the Isabelle implementation (including attack trace) takes 653. Tamarin is not guaranteed to terminate for all claims, but in our case study no such problems arose, with all protocols taking less than 20 seconds (< 6 on average) to either successfully verify or provide an attack trace.

The definition of the *causality-based secure distance-bounding* property is not immediately compatible with the specification model that Tamarin uses. We note the following factors:

1. The definition uses a claim event that refers specifically to other events (which mark the start and end of the fast phase). This requires that we can successfully select terms which will bind these events to the specific session being analysed.
2. The specification language partitions agents into the sets Honest and Dishonest of honest agents (who attempt to perfectly follow the protocol's intended execution) and dishonest agents (who are willing to make use of other rules in order to violate security properties). Tamarin carries no understanding of the intended execution of a protocol. Further, Tamarin does not inherently carry the notion of agents, although they are trivially modelled by public variables. The sending of messages by the adversary is modelled using built-in rewriting rules that are often not straightforward to write claims around.
3. The security property is dependent on the identity of the *actor* of an event: i.e. the agent who performed the action. Tamarin does not explicitly attach an identity to a rewriting rule's application, as a consequence of agents not being an inherent feature of Tamarin.

These issues were addressed as follows:

1. In order to model claim events, state facts containing session data were used. In particular, the VerifierComplete($params$) state fact was added to all rewrite rules designed to symbolise that the verifier role believes they successfully completed the protocol with session data *params*. The term *params* is built from all session

data added to the protocol in the order that it is added. This necessarily includes two public variables to model the identities of the prover and verifier, as well as at least one fresh variable used in the protocol execution. However, different protocols make use of different numbers of fresh variables, so the number of them inside this state fact varies slightly. The state facts StartFastPhase($data$) and EndFastPhase($data$) model the start and end of the fast phase as defined by the protocol specification. Note that in protocols involving pre-commitments, the verifier is not fully aware of the value of all of the session data at the start or end of the fast phase, and so it is not necessarily the case that *params* and *data* will be equal. However, the subterms of *data* are a strict subset of the subterms of *params*.

Assuming that the session data of a protocol is different between different runs of the protocol, the subterms of the VerifierComplete fact refer unambiguously to the corresponding state facts for the denoting fast phase. If a protocol does not have different session data between executions, it is trivially vulnerable to replay attacks.

2. Agents are modelled in protocol specifications as public terms. Rewrite rules are included to model an agent receiving any secret keys or other information they have at the start of a protocol's execution. In this case, we see facts of the form Ltk(A, k), denoting that the agent A has key k. Additional rewrite rules are added for the corruption of agents (in which a fact containing a secret key is sent on to the network, revealing the identity of the agent), and also to model a corrupt agent sending a message on the network. This is important for adding state facts to symbolise the adversary acting during the fast phase.

3. For the secure distance-bounding claim to make sense the identity of the prover must be used in the protocol in some way: either their identity is used in a message, or the prover possesses a long-term secret key used in a calculation. This could be for symmetric or asymmetric encryption, or in some cases for signed hashes. Any multiset rule that uses an agent's identity (or carries session data from an earlier rule in the protocol which does) is marked with the state fact Action($agent$).

With these in mind, the Tamarin lemma `dbsec` is defined in Figure 3.8. This lemma can be understood as meaning that whenever a verifier reaches the end of their protocol execution, one of three following events is possible:

1. The verifier is corrupt: they have revealed their long term secret key to the adversary, making their claim invalid.

2. Between the start and end of the fast phase, the agent P that the verifier believes is close performed some action.

3. The agent P that the verifier believes is close has revealed their long term secret key to the adversary. Between the start and end of the fast phase, some corrupt agent (who may be P or another agent who has revealed their long-term key) performed an action.

Dbsec lemma

```
lemma dbsec:
"
All P V m n #t. (
VerifierComplete(P, V, m, n)@t ) ==>
(
  Ex #tc.
    Corrupt(V)@tc
)|(
  Ex #t1 #t2 #t3.
    StartFastPhase(V, m)@t1 &
    Action(P)@t2 &
    EndFastPhase(V, m)@t3 &
    (#t1 < #t2) &
    (#t2 < #t3) &
    ( (#t3 < #t ) | (#t3 = #t) )
)|(
  Ex CAgent #t4 #t5 #t6 #t7.
    StartFastPhase(V, m)@t5 &
    EndFastPhase(V, m)@t7 &
    Corrupted(P, V)@t4 &
    CAction(CAgent)@t6 &
    (#t5 < #t6)&
    (#t6 < #t7)&
    ( (#t7 < #t) | (#t7 = #t) )
)
"
```

FIGURE 3.8: The Tamarin lemma dbsec.

Note that $\#t$, the time of the claim for dbsec security, marks the end of the protocol execution. It may be the case that $\#t = \#t3$, the time when the Verifier receives the response in the fast phase.

Our implementations of the protocols also involve a number of *reachability* lemmas. These lemmas are not related to the main dbsec lemma, but instead prove that the protocol has been implemented in such a way that the various stages of the protocol can be reached as per their intended execution. If the end of the protocol is not reachable, then the dbsec property is trivially true.

Finally, the protocol implementations include some *trace restrictions*. These are claims that are assumed to be true when Tamarin constructs proofs for the lemmas. The main restrictions used are at_most_once (Figure 3.9) and equality (Figure 3.10).

The restriction at_most_once is used to ensure that a single agent (or pair of agents) may only be given a single long term key (or shared key, respectively), and the restriction equality serves to verify that an equation holds: typically used in the case of verifying that a signature lines up with the message it is intended to be signing.

Uniqueness restriction

```
restriction at_most_once:
"
All A #t1 #t2.
Once(A)@t1 & Once(A)@t2 ==>
(
   #t1 = #t2
)
"
```

FIGURE 3.9: The Tamarin restriction `at_most_once`.

Equality restriction

```
restriction equality:
"
All a b #t1. Eq(a, b)@t1 ==> a = b
"
```

FIGURE 3.10: The Tamarin restriction `equality`.

3.5 Case Study: TREAD Protocol

The TREAD protocol was claimed to satisfy various security properties, making use of the computational model DFKO introduced in [DFKO11]. Relaying on this model, a proof is given to show probabilistic resistance[1] against mafia-fraud, distance-fraud, terrorist-fraud, and distance-hijacking attacks. However, by using our framework, we have identified mafia-fraud and distance-hijacking attacks on this protocol.

TREAD consists of three phases (see Figure 3.11). First, the prover P generates two nonces α and β, and creates the message $\sigma = \alpha|\beta|\mathsf{idpriv}(P)$, where $\mathsf{idpriv}(P)$ is an anonymous group identity. This message is signed by P and sent encrypted to the verifier V, together with P's identity $\mathsf{idpub}(P)$. Upon reception, V decrypts the message and verifies the signature. If correct, V finishes the first phase by sending a random nonce m of size n to P. The second phase is a standard n-round fast phase wherein V sends a random bit c_i with $i \in \{0, \ldots, n-1\}$ and P replies back with α_i if $c_i = 0$, with $\beta_i \oplus m_i$ otherwise. The protocol finishes successfully if all responses during the fast phase are correct and the round-trip times are below a predefined threshold (third phase).

To symbolically verify TREAD, we transform the fast phase into a single challenge-response message exchange (see Figure 3.12). We also ignore details that are irrelevant to our security analysis, such as the anonymous identity of the prover, and upgrade bitwise operations to stronger cryptographic primitives, such as a hash function. Overall, our goal is to obtain an abstraction of the original protocol such that every attack found in the abstraction can be mapped back onto the original protocol.

[1] No attack succeeds with non-negligible probability.

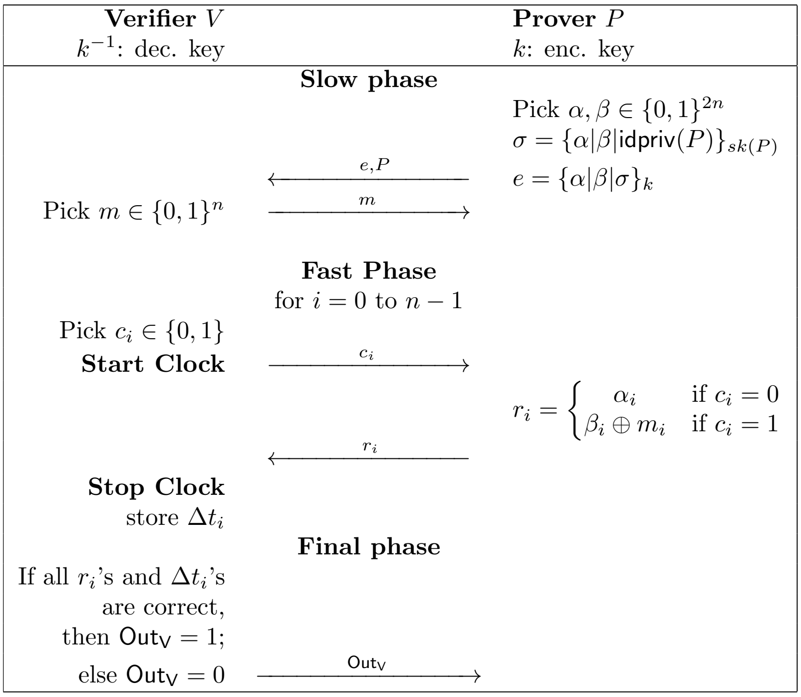

Verifier V
k^{-1}: dec. key
Prover P
k: enc. key
Slow phase
Pick $\alpha, \beta \in \{0,1\}^{2n}$
$\sigma = \{\alpha|\beta|\mathsf{idpriv}(P)\}_{sk(P)}$
e,P
$e = \{\alpha|\beta|\sigma\}_k$
Pick $m \in \{0,1\}^n$
m
Fast Phase
for $i = 0$ to $n-1$
Pick $c_i \in \{0,1\}$
Start Clock
c_i
$r_i = \begin{cases} \alpha_i & \text{if } c_i = 0 \\ \beta_i \oplus m_i & \text{if } c_i = 1 \end{cases}$
r_i
Stop Clock
store Δt_i
Final phase
If all r_i's and Δt_i's
are correct,
then $\mathsf{Out_V} = 1$;
else $\mathsf{Out_V} = 0$
$\mathsf{Out_V}$

FIGURE 3.11: The TREAD protocol.

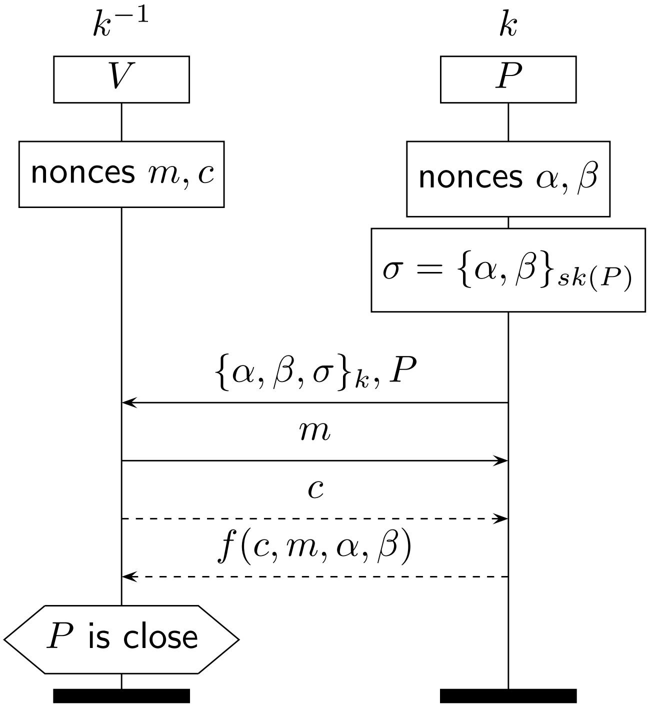


FIGURE 3.12: A representation of the TREAD protocol.

TREAD can be instantiated with either a symmetric or an asymmetric encryption scheme. We thus specified in Tamarin two variants of the TREAD protocol: one where k is a symmetric key and another one where k is an asymmetric key. In the second variant, Tamarin finds a simple man-in-the-middle attack that violates the secure distance-bounding property. The attack is depicted in Figure 3.13 and works as follows. An intruder I initiates a session with the prover P by requesting P to prove proximity. P then sends the message $(\{\alpha, \beta, \{\alpha, \beta\}_{sk(P)}\}_{pk(I)}, P)$ to I. Now the intruder decrypts the received message, learns the nonces α and β, and re-encrypts the message with the public key of the verifier. Next, the intruder starts a session with a legitimate verifier V with goal of impersonating P. To do so, I sends $(\{\alpha, \beta, \{\alpha, \beta\}_{sk(P)}\}_{pk(V)}, P)$ to V. Then V checks that the signed message $\{\alpha, \beta\}_{sk(P)}$ indeed corresponds to P, and sends back two nonces m and c. The attack ends with the intruder correctly replying to the challenges with $f(c, m, \alpha, \beta)$.

Observe that the attack described above and depicted in Figure 3.13 not only breaks standard authentication properties such as agreement and synchronization [Low97, CM12], but also the secure distance-bounding property as follows. Assume P is far from V and the intruder wants to convince V that P is close. To do so, the intruder just needs to be close to V and executes the attack above. Note that the fast phase corresponds to the events containing the messages c and $f(c, m, \alpha, \beta)$, which the intruder can successfully produce without relaying.

Interesting enough, if k is a symmetric key the described mafia-fraud attack does not work. The reason is that the intruder does not know the secret key shared between P and V. Thus the intruder is prevented from re-encrypting the message received from P with the correct key. Nevertheless, a distance-hijacking type of attack exists irrespective of the encryption scheme. The attack is represented in Figure 3.14. Assume an honest prover P is close to the verifier V, while the intruder I is far from V. As before, P executes the protocol to prove its proximity to I. This allows I to learn α and β. Thus I starts a session with V by using the nonces α and β from P. At this point, V believes I is a legitimate prover and accept its signature. During the fast phase, P, which is close to V, receives the challenge (supposedly from I) sent by V and replies correctly. Then V receives the response $f(c, m, \alpha, \beta)$ (supposedly from I) from P who is close to V, and finishes the protocol with I correctly.

Neither of the two described attacks are possible when considering the adversary model used by the authors of the TREAD protocol, because their model does not allow for "malicious" verifiers. In their model an honest prover will fail to initiate a communication with an untrusted verifier as the first message in each attack will not be sent. This adversary model is weaker than other models that are more common in the distance-bounding literature.

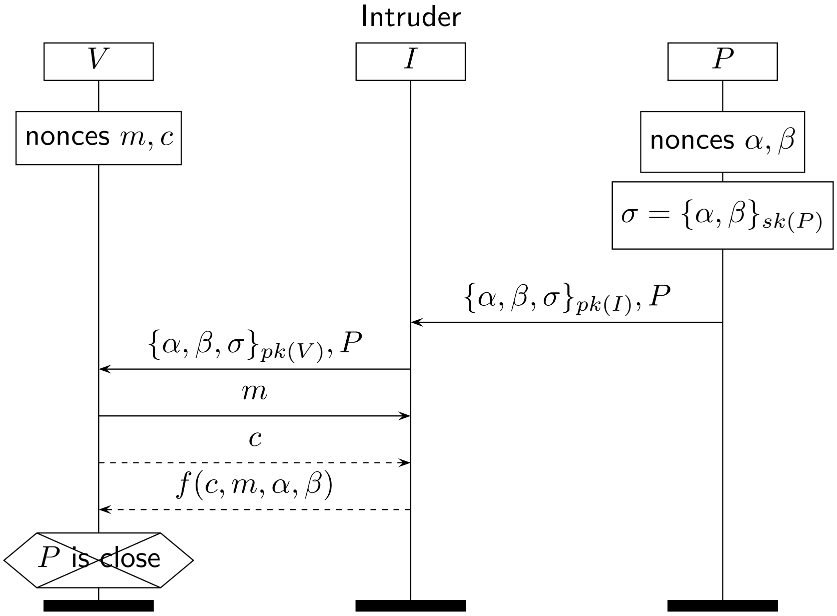

Intruder
V
I
P
nonces m, c
nonces α, β
$\sigma = \{\alpha, \beta\}_{sk(P)}$
$\{\alpha, \beta, \sigma\}_{pk(I)}, P$
$\{\alpha, \beta, \sigma\}_{pk(V)}, P$
m
c
$f(c, m, \alpha, \beta)$
P is close

FIGURE 3.13: A mafia fraud on TREAD with asymmetric encryption.

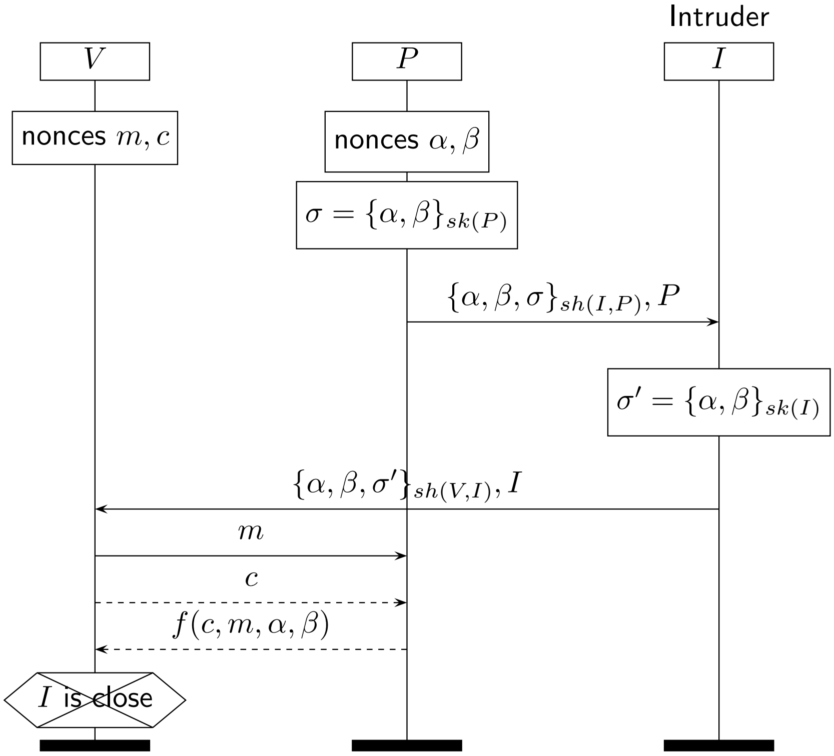

Intruder
V
P
I
nonces m, c
nonces α, β
$\sigma = \{\alpha, \beta\}_{sk(P)}$
$\{\alpha, \beta, \sigma\}_{sh(I,P)}, P$
$\sigma' = \{\alpha, \beta\}_{sk(I)}$
$\{\alpha, \beta, \sigma'\}_{sh(V,I)}, I$
m
c
$f(c, m, \alpha, \beta)$
I is close

FIGURE 3.14: A distance hijacking on TREAD with symmetric encryption.

Chapter 4

Relations on Security Properties

Lowe demonstrated a hierarchy of security properties: the idea that certain goals strictly dominate others. For example, any protocol that satisfies recent aliveness necessarily satisfies weak aliveness. In this chapter we generalise this approach in two main dimensions.

First, we consider relations between security goals on the protocol level, rather than being universal over all protocols. This is important for being able to prove results about families of protocols which share common properties or requirements. Second, we consider a novel relation in the form of irreversibility. Intutively, we capture the idea that violation of one security goal inevitably leads to violation of another.

We find a natural application of this theory by extending the analysis of Distance Bounding protocols from Chapter 3. A well-documented class of attacks on such protocols is known as Terrorist Fraud. These attacks result from a prover colluding with the adversary - temporarily deviating from their protocol specification in order to fool the verifier. This gives a clear contrast to the traditional corruption model given by the Dolev-Yao adversary, which is binary in nature.

We define an authentication property modelling post-collusion security, which is a natural application of irreversibility on protocols extended by collusion rules. This then allows for a definition for Terrorist Fraud resistance, which is used for a thorough survey of protocols in the literature.

This chapter extends the work in the ACM CCS 2019 Paper "Post-collusion security and distance bounding" [MSTPTR19]. In particular, some of the underlying theory developed while working on this paper has been added. An additional section is added to highlight the concept of relations between security goals, highlighting irreversibility (and thus collusion resistance) as a special case. Further, some additional discussion is given on the feasibility of brute force approaches to solving the general case of the security of protocol mutations.

4.1 Introduction to Relations

There are many well known descriptions of *relations* between security properties. The most distinct of these is a hierarchical one: recent aliveness is a “stronger" property than weak aliveness, in that any attack on weak aliveness is necessarily also an attack on recent aliveness.

Relations such as these are defined on a global level – that is, the relation exists regardless of the protocol being examined. Further, for a majority of protocols certain other relations hold: for all but the most trivial protocols, an attack on key secrecy will entail an attack on some agreement property.

In this chapter, we will define a relation between security goals whose edges change on a per-protocol basis. In addition, we will look at how the truth value of a security goal can change over slight variations of a protocol, by extending the Dolev-Yao adversary.

In some cases, the Dolev-Yao model has been shown to be too coarse-grained. This is because this model assumes that agents can be categorised as being either honest: those who precisely follow their protocol specification; or compromised: those who deviate from their protocol specification as desired by the adversary. We will now investigate the idea of agents who cannot be classified in either group.

For example, *covert adversaries* [AL10, FY92, CO99] are agents who are willing to cheat by deviating from the protocol specification, as long as the cheating would not be detected. One might think of an online gaming platform, in which some players secretly cooperate to cheat against other players, whilst avoiding being caught, or else face consequences such as being thrown out of the platform.

A common concern in the study of multiparty computation protocols is that of an *honest but curious* party, who might break some protocol rules in order to gain more information, but still protect the overall integrity of the protocol (e.g. ensuring that the result of the computation is still correct). In some cases, this might involve collaboration between parties, which we classify as collusion in our model.

Variations of the Dolev-Yao threat model capturing more refined *dishonest* behaviour have been studied [AL10, FY92, BRS16, BC14, BC10, CCG16], which have led to rethinking the security models to properly account for such fine-grained adversaries. Such models attribute dishonest behaviour to the adversary’s compromise capabilities, but in some scenarios such behaviour might not be ruled by the adversary, but rather by the protocol’s participants themselves. For example, a given agent might choose to deviate from the protocol specification, but only if certain guarantees are met in later executions of the protocol. Would a university student willingly, due to certain benefit, lend their campus access card to a university-external friend? The student’s decision might be conditional on their assertion that their friend will not be able to later access the campus, after the card has been returned to its owner. Would a user of a video streaming platform utilize a VPN extension to fool geo-location restrictions? The user’s decision might be based on whether they are certain that the VPN extension is not malicious and will not cause irreversible harm.

In this paper we refine the traditional Dolev-Yao adversary model in order to capture *collusion*. Collusion refers to any *deviation* of the protocol specification by agents who are not under control of the adversary. Furthermore, we introduce the notion of *post-collusion security*, which refers to security guarantees about claims made in execution sessions initiated after the collusion. Informally, one can interpret the relation of these two notions as follows: post-collusion security allows the potential colluding agents to decide whether colluding is worth it. After all, what the agents gain out of colluding must outweigh the collateral effect that such collusion might have on themselves. On the other hand, a protocol designer might aim to increase the cost of collusion.

A related notion was introduced by Cohn-Gordon et al. [CCG16], called *post-compromise security*, that looks at the timeline of the compromise actions and their impact in the security of future protocol sessions. As motivated earlier, collusion differs from compromise in that compromise is an action performed by the adversary in order to exert control over the protocol, whilst collusion represents a deliberate choice of the agent involved.

In post-compromise security, the compromise is ruled by the adversary, regardless of the (future) consequences on the compromised agent. Post-collusion security, instead, allows the agents to base their choice of collusion on post-collusion guarantees. One can think of "not getting caught", in the online gaming example given earlier, as the post-collusion guarantee. In Section 4.2 we give further technical differences between post-compromise and post-collusion security.

Our notion of post-collusion security finds a straightforward application in *distance-bounding protocols* [BD90, BC93], which are security protocols that aim to guarantee physical proximity. These protocols are used in RFID and NFC technologies, with numerous applications in secure systems such as contactless payment and access control. Post-collusion security allows us to formally analyse a non-trivial type of attack on distance-bounding protocols known as *terrorist fraud* [DGB87b]. In this attack, agents collude to falsely prove proximity for one run of the protocol, whereas no further false proximity proofs can be issued without further collusion.

Contributions.
The contributions of this chapter are:

- We provide a formal symbolic model based on multiset rewriting systems that captures collusion in security protocols, which represents non-compromised agents deviating from their given protocol specification.
- We introduce the notion of post-collusion security, which refers to the validity of security claims made in protocol sessions initiated after the collusion. We provide a concrete formulation of this notion that can be easily implemented in protocol verification tools such as TAMARIN [MSCB13].
- Our definitions are used to provide a formal description of the sophisticated terrorist fraud on distance-bounding protocols. Further, we develop a TAMARIN-based framework for verification of such type of protocols that exhaustively

accounts for all classes of attack from literature.

- We conduct a security survey of over 25 protocols, which include industrial protocols based on the ISO/IEC 14443 standard. We propose computer-verified fixes for the vulnerabilities encountered in these protocols.

Organisation.

This chapter is organised as follows. In Section 4.2 we discuss the idea of *collusion* rules: creating variants of a protocol to model dishonest agents. Section 4.3 uses these extensions to define Post-Collusion security – which describes the irreversible consequences of certain protocol deviation rules. Next, in Section 4.4 we use this theory to return to the domain of Distance Bounding protocols from the previous chapter, using it to model the class of attacks known as Terrorist Fraud. Section 4.5 contains a case study of several industrial protocols used for card payments. Finally, Section 4.6 contains results from this chapter and the previous one – a large survey of distance bounding protocols from the literature.

4.2 Collusion and Irreversibility

In this section we introduce the idea of protocol deviations. We start by introducing related work from across the literature, and then introduce our own definition of well-formed collusion rules.

Related Work

In this section we describe some works in which the authors analyse alternative adversary models that modify the Dolev-Yao capabilities. We pay special attention to existing symbolic verification frameworks for distance-bounding protocols, which is the main application field of our findings.

Alternative Adversary Models.
In 2010, Basin and Cremers [BC10] proposed a model in which they formally defined several extensions to the Dolev-Yao adversary. These extensions were used to analyse a variety of protocols against adversaries of varying strength [BC14]. As a result, they identified new attack vectors in key-exchange protocols such as KEA+ [LM06], Naxos [LLM07] and the MQV protocol family [Kra05].

In [BRS16] the authors provide a formalism to model and reason about human misbehaviour. A set of rules describe an untrained human, who is willing to perform arbitrary actions but follows a set of guidelines, such as "private keys must be kept secret". The TAMARIN tool is used to automatically analyse security protocols involving human errors.

Cohn-Gordon et al. introduced post-compromise security in [CCG16], defined as an agent's security guarantees about a communication with their peer, even if their peer has been already compromised. They analysed two types of compromise: weak and total. Weak compromise corresponds to temporary adversarial control of an agent's long-term keys in form of a cryptographic oracle, which outputs the result of a crypto-operation , without revealing the long-term keys. Post-compromise security has been recently used in [CCG+18] to analyse group messaging protocols.

The adversary model for post-compromise security is similar to that of post-collusion security in that they both allow for dishonest behavior not conceived by the Dolev-Yao adversary. Yet, they differ in that weak compromise is controlled by the adversary regardless of the compromised agents' will, whilst collusion is the agents' deliberate choice. This choice can be based on whether or not certain post-collusion guarantees are met. Furthermore, Cohn-Gordon et al. 's post-compromise security focuses on *stateful* protocols, such as authenticated key-exchange (AKE) and messaging protocols. Our post-collusion security notion can be applied, but is not limited to this type of protocol. In addition, our approach is oriented to symbolic security analysis, whereas theirs uses a computational approach. As a result, our methods can be more smoothly implemented in state-of-the-art verification tools for analysing complex protocols.

Collusion can also be considered in the context of adveraries other than Dolev-Yao, or even with no adversary at all. Tompa and Woll [TW86] present an attack on Shamir's secret sharing [Sha79], based on the principle of colluding agents. In the domain of multiparty protocols, Hirt and Maurer [HM97] give a classification of how different agents may deviate from their specification (e.g. 'honest-but-curious' participants, or may collude between each other. Syverson et al. [SMC00] build upon an adversary model (named "Machiavelli") which does not directly corrupt agents, but instead manipulates them through an extensive collection of collusion rules. We build upon these papers, by looking at the impact on security *after* collusion occurs, and to make progress towards identifying the key deviations from the protocol specification that will result in "successful" collusion within certain domains.

Collusion

This section is dedicated to providing a formal description of the notions of *collusion*, which is an extension to the adversary model, and *post-collusion security*, which is a security model under the extended adversary.

More precisely, in Section 4.2 we extend the adversary model with collusion rules, which express ways in which non-compromised agents can deviate from the protocol specification.

Collusion Rules

In the traditional Dolev-Yao compromise model, agents are assumed to be either *compromised* (a.k.a. corrupt, dishonest) or *non-compromised* (a.k.a. honest). Non-compromised agents follow precisely the protocol specification, whilst compromised agents deviate from it as pleased by the adversary.

We refine the traditional Dolev-Yao compromise model so that agents can *collude* in order to provide false proof to their communication partners of a certain claim's validity. Collusion refers to non-compromised agents' deviation from their protocol specification. The basic deviation consists of leakage of session data, cryptographic oracles, reuse of nonces, or state reveals.

For example, assume *Alice* is running an authentication protocol (supposedly) with *Bob*. Consider also a third party *Charlie* who, in cooperation with *Bob*, impersonates *Bob* when communicating with *Alice*. *Bob* could trivially achieve this by giving all his secret keys to *Charlie*. But, does *Bob* really have to do so in order to deceive *Alice*? Not necessarily. Indeed, *Bob* can provide *Charlie* (possibly in advance) with all the messages that *Charlie* needs to successfully complete a protocol session with *Alice*, posing as *Bob*. Such aid by *Bob* is what we call collusion, and we call *Bob* a colluding agent.

One example of a deviation is an encryption oracle, which can be modelled as follows:

$$\texttt{EncOracle} := \begin{bmatrix} \mathsf{In}(m), \\ \mathsf{Shk}(I,R,k) \end{bmatrix} \xrightarrow{\mathsf{Collusion}(\,)} \begin{bmatrix} \mathsf{Out}(\mathrm{senc}\, mk) \end{bmatrix}.$$

The rules that extend the protocol specification to model collusion are called *collusion rules*. By convention, and also to syntactically distinguish legitimate protocol rules from collusion rules, we will assume that all collusion rules have an action fact of the form $\mathsf{Collusion}()$. We denote by $\mathcal{C} \subseteq \mathcal{R}$ the universe of all collusion rules. We restrict the set of collusion rules by requiring them to not prevent agents from completing legitimate protocol runs.

Definition 2.1 (Valid Extension). *Let $P \subseteq \mathcal{R} \setminus \mathcal{C}$ be a protocol and $C \subseteq \mathcal{C}$ be a set of collusion rules, we say that $P' = P \cup C$ is a* valid extension *of P if:*

$$\begin{aligned} &\forall \alpha \in \mathit{Traces}(P'), i, x. \\ &\quad (\mathsf{Start}(x) \in \alpha_i \wedge \nexists j.\ \mathsf{End}(x) \in \alpha_j) \implies \\ &\qquad \exists \beta.\ \alpha \cdot \beta \in \mathit{Traces}(P') \wedge \mathsf{End}(x) \in \beta_{|\beta|}. \end{aligned}$$

Definition 2.1 states that collusion rules do not create points of no-return during execution. That is to say, agents must always be able to complete their runs even if they have colluded.

Other than this requirement of not preventing termination, we place no other restrictions on collusion rules. Besides leakage rules or function oracles (as demonstrated), we also allow for more esoteric deviations, such as re-use of fresh values, or passing of messages between multiple colluding protocol participants (and not an adversary or fully compromised agent).

This means that protocol extensions may be defined which, for example, have a colluding agent behave identically to a corrupt agent – by leaking their long-term encryption keys. However, we will see in the following section that these cases do not play a significant role in our definitions. Importantly, our results following from post-collusion security will involve quantifying over **all** extensions that satisfy a given

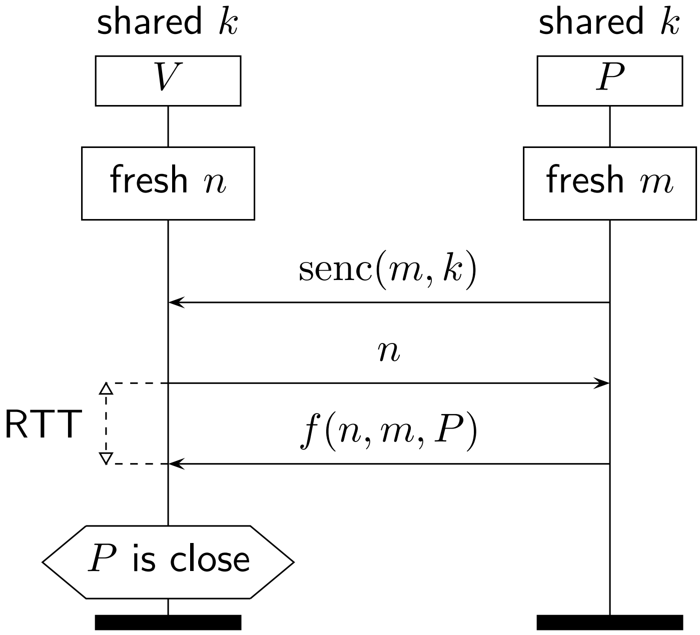


FIGURE 4.1: The *DBToy* protocol.

property (violation of a security goal). We will build a definition of Terrorist Fraud resistance which intuitively says that any collusion rule which leads to violation of security goals is indistinguishable from full corruption.

4.3 Post-Collusion Security

In this section we introduce the notion of post-collusion security. We informally define it as follows.

Definition 3.1 (Informal). *Post-collusion security is the guarantee of security claims made in sessions initiated* after *collusion occurs.*

The remainder of this section is intended to formalise the above informal definition of post-collusion security. To illustrate our definitions and intuitions, we will use the *Toy* protocol, as follows:

Example 3.2 (The *DBToy* Protocol). *Figure 4.1 depicts the DBToy protocol, which works as follows. The prover P encrypts a fresh name m with the shared key between P and the verifier V. Then P sends the encrypted message to V. Hence, the fast phase starts with V sending the fresh name n as the challenge, to which P must reply with $f(n, m, P)$. If P replies correctly and on time, then V declares P as being close. The specification rules of DBToy are shown in Figure 4.2.*

To identify claims made in sessions initiated after the collusion, which we call *post-collusion claims*, we must make sure that all sessions before or while the (last) collusion occurred are complete. The reason for this is that an agent who makes a security claim cannot always decide whether their communication partner is still acting on a run initiated before or during the collusion. That is, a claim by *Alice* about her

$$
\begin{aligned}
&\mathtt{KeyGen} := [\ \mathsf{Fr}(k)\ ]\!\rightarrow\![\ \mathsf{Shk}(V,P,k)\ ]\\
&\mathtt{KeyRevV} := [\ \mathsf{Shk}(V,P,k)\ ]\xrightarrow{\mathsf{Compromise}(V)}\begin{bmatrix}\mathsf{Out}(k),\\ \mathsf{Compromise}(V)\end{bmatrix}\\
&\mathtt{KeyRevP} := [\ \mathsf{Shk}(V,P,k)\ ]\xrightarrow{\mathsf{Compromise}(P)}\begin{bmatrix}\mathsf{Out}(k),\\ \mathsf{Compromise}(P)\end{bmatrix}\\
&\mathtt{DBInject} := [\ \mathsf{In}(m),\mathsf{Compromise}(X)\ ]\!\rightarrow\![\ \mathsf{Send}(X,m)\ ]\\
&\mathtt{DBSend} := [\ \mathsf{Send}(X,m)\ ]\xrightarrow{\mathsf{Send}(X,m),\mathsf{Action}(X)}[\ \mathsf{Net}(m),\mathsf{Out}(m)\ ]\\
&\mathtt{DBRecv} := [\ \mathsf{Net}(m)\ ]\xrightarrow{\mathsf{Action}(Y),\mathsf{Recv}(Y,m)}[\ \mathsf{Recv}(Y,m)\ ]\\
&\mathtt{P1} := [\ \mathsf{Fr}(m),\ \mathsf{Shk}(V,P,k)\ ]\xrightarrow{\mathsf{Start}(m)}\begin{bmatrix}\mathsf{Send}(P,\mathsf{senc}\,mk),\\ \mathsf{ProvSt1}(P,m)\end{bmatrix}\\
&\mathtt{V1} := \begin{bmatrix}\mathsf{Fr}(n),\ \mathsf{Shk}(V,P,k),\\ \mathsf{In}(\mathsf{senc}\,mk)\end{bmatrix}\xrightarrow{\mathsf{Start}(n),\mathsf{Send}(V,n)}\begin{bmatrix}\mathsf{Out}(n),\\ \mathsf{VerifSt1}(V,P,n,m)\end{bmatrix}\\
&\mathtt{P2} := [\ \mathsf{ProvSt1}(P,m),\ \mathsf{In}(n)\ ]\xrightarrow{\mathsf{End}(m)}[\ \mathsf{Send}(P,f(n,m,P))\ ]\\
&\mathtt{V2} := \begin{bmatrix}\mathsf{VerifSt1}(V,P,n,m),\\ \mathsf{Recv}(V,f(n,m,P))\end{bmatrix}\xrightarrow{\mathsf{DBSec}(V,P,n,f(n,m,P)),\ \mathsf{End}(n)}[\ \ ]
\end{aligned}
$$

FIGURE 4.2: Specification rules of the *DBToy* protocol.

communication with *Bob* is a post-collusion claim if both *Alice* and *Bob* have completed their runs that started before or while *Bob* performed the collusion action(s). That way, we make sure that *Alice* makes her claim in a session initiated after *Bob*'s collusion action.

Consider a trace $t = t_1 \cdots t_e \cdots t_i \cdots t_n$, and an index e such that all collusion actions (if any) occurred before e. If all runs initiated before e were completed before e too, then we call the security claims made after e post-collusion claims. See Figure 4.3 for a graphical representation. Note that every claim that occurs after a post-collusion claim is also a post-collusion claim.

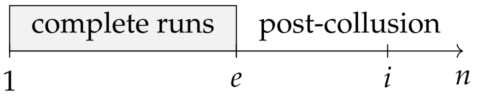


FIGURE 4.3: A trace $t = t_1 \cdots t_e \cdots t_i \cdots t_n$ can be broken down into a pre-collusion trace consisting of completed runs (e.g. before e), and a second subtrace containing post-collusion claims (e.g. a claim made in t_i).

Below, in Definition 3.3 we formulate post-collusion security, in which we use the following helper predicates on sequences of sets of ground facts:

$$
\begin{aligned}
\mathit{complete}(l) &\iff \forall i,x.\ (\mathsf{Start}(x)\in l_i \implies \exists j.\ \mathsf{End}(x)\in l_j),\\
\mathit{nocollusion}(l) &\iff \nexists j.\ \mathsf{Collusion}()\in l_j.
\end{aligned}
$$

In words, $\mathit{complete}(l)$ holds if all runs initiated in l are also completed in l; $\mathit{nocollusion}(l)$ means that no collusion actions occurred in l. We note that the $\mathit{complete}()$ predicate gives a strong divide between the complete runs and the post-collusion runs. However, we assert that for interleaved traces (i.e. those in which there is always an active session), there is an equivalent trace which satisfies the predicate. Intuitively, sessions between unrelated agents have no causal dependence and so can be reordered. This leaves only series of sessions by the same agents. In the case that an agent may be participating in multiple sessions simultaneously, we must require that all of them are finished before we can make post-collusion claims. This is because we cannot guarantee that a collusion action taken in one session will necessarily only lead to attacks in that session - for example, an agent may collude by acting as a function oracle that can be used in any one of their active sessions.

We now define post-collusion security:

Definition 3.3 (Post-collusion Security)**.** *Given a protocol P, a valid extension P' of P, and a security property φ, we say that P'* is post-collusion secure with respect to *φ, denoted $P' \models^{\star} \varphi$, if:*

$$\begin{aligned} &\forall t \in \mathit{Traces}(P'), e \in \{1, \ldots, |t|\}. \\ &\quad (\mathit{complete}(t_1 \cdots t_e) \wedge \mathit{nocollusion}(t_{e+1} \cdots t_{|t|})) \\ &\qquad \implies \forall i > e.\ \varphi(t, i). \end{aligned} \tag{4.1}$$

We write $P' \not\models^{\star} \varphi$ to indicate that $P' \models^{\star} \varphi$ does not hold.

Post-collusion security is really a statement about *reversibility*. Informally, we are told that the damage caused by a protocol deviation (for example, that it breaks φ) is ultimately temporary. Eventually, the protocol will be able to return to its original security guarantees, as long as no further collusion occurs.

As Figure 4.4 shows, $\mathit{Toy} \cup \{\texttt{Leak\_ni}\}$ is *not* post-collusion secure with respect to non-injective agreement, i.e.

$$\mathit{Toy} \cup \{\texttt{Leak\_ni}\} \not\models^{\star} \mathit{ni\_agreement}. \tag{4.2}$$

The attack works with two consecutive sessions, in which a compromised agent *Eve* can re-use the messages senc *nik* and *ni* from the first session to impersonate *I* in the second session. Observe that the second claim is a post-collusion claim, as the first session is complete and no collusion occurred in the second session.

The impact of post-collusion security can depend on the circumstances in which a given protocol is deployed. We see from the *Toy* protocol that the effects of collusion can cause an irreversible change to the truth value of future authentication claims. Thus, a legitimate agent playing the initiator role would not want to collude with a "friend" by giving them their nonce *ni*, as this would lead to impersonation. On the contrary, suppose a given protocol is post-collusion secure with respect to a desirable authentication property. Then, an agent can issue their one-time keys to their friends if desired, confident that these friends will not be able to re-use this information for later authentication.

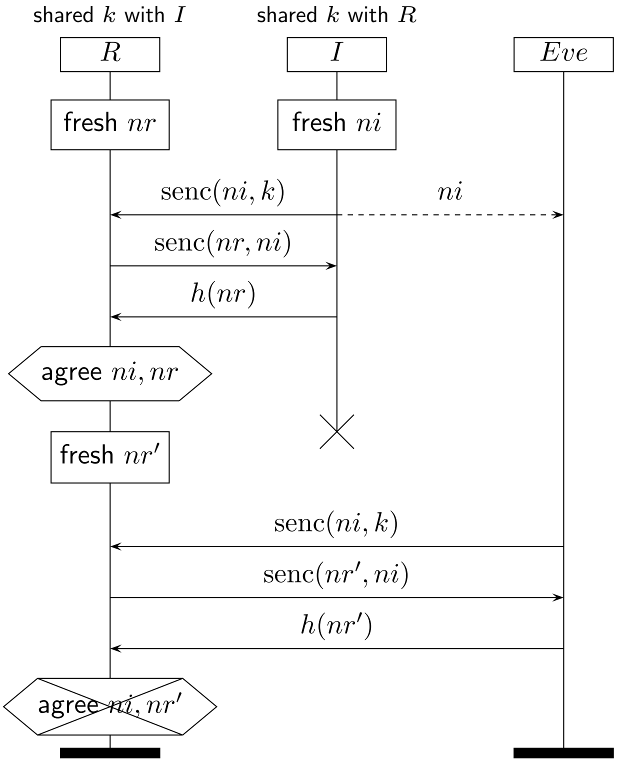


FIGURE 4.4: An MSC showing that the *Toy* protocol with collusion, represented by the dashed arrow, is not post-collusion secure with respect to non-injective agreement.

4.4 Terrorist Fraud on Distance Bounding Protocols

In this section we use post-collusion security to develop a symbolic formulation of terrorist fraud in distance-bounding protocols.

Formalising (Resistance To) Terrorist Fraud

We informally define terrorist fraud as follows.

Definition 4.1 (Informal). *Terrorist fraud (TF) is an attack in which a remote and non-compromised prover P colludes with a close and compromised prover A to make the verifier believe that P is close. Conditionally, A (or any other compromised prover) must not be able to attack the protocol again without further collusion.*

The *dbsec* property allows us to detect attacks in which the proving party is compromised, such as distance fraud [Des88] and distance hijacking [CRSC12]. However, *dbsec* is too fine-grained for modelling terrorist fraud, as we require the distant and colluding prover to be non-compromised (in the case of a compromised prover, collusion actions do little to aid the adversary). In line with this reasoning, we define below a property weaker than *dbsec*, that is conditional on non-compromise of both prover and verifier:

$$\begin{aligned}
&\mathit{dbsec\_hnst}(t,l) \iff \\
&\quad \forall V,P,C,R.\ \mathsf{DBSec}(V,P,C,R) \in t_l \implies \\
&\qquad (\exists i,j,k.\ i<j<k \wedge \mathsf{Send}(V,C) \in t_i \wedge \\
&\qquad\quad \mathsf{Action}(P) \in t_j \wedge \mathsf{Recv}(V,R) \in t_k) \vee \\
&\qquad (\exists i.\ \mathsf{Compromise}(V) \in t_i \vee \mathsf{Compromise}(P) \in t_i).
\end{aligned}$$

Intuitively, a trace satisfies *dbsec_hnst* if, whenever a verifier V believes a prover P is close, P took some action between the verifier sending the challenge ch and receiving reponse rp.

We formally define next *resistance to terrorist fraud*, a property formulated by means of post-collusion security with respect to *dbsec_hnst*.

Definition 4.2 (Resistance to Terrorist Fraud)**.** *A protocol $P \subseteq \mathcal{R} \setminus \mathcal{C}$ is* resistant to terrorist fraud *if every valid extension P' of P that breaks dbsec_hnst is* not *post-collusion secure with respect to dbsec_hnst, i.e.*

$$P' \not\models \mathit{dbsec\_hnst} \implies P' \not\models^{\star} \mathit{dbsec\_hnst}.$$

Observe that resistance to terrorist fraud is a property on protocols rather than on traces. Further, terrorist fraud uses the negation of post-collusion security. This is because in a terrorist fraud attack, the colluding prover wishes to allow their partner to complete the protocol only whilst they are cooperating.

Note that Terrorist Fraud resistance is defined only for a single security property, *dbsec_hnst*. In fact, it is a special case of the following:

Definition 4.3 (Irreversibility Relation)**.** *Given a protocol $P \subseteq \mathcal{R} \setminus \mathcal{C}$, and two security goals φ_1 and φ_2, we say that φ_2 is* irreversible after violating φ_1 in P *if every valid extension P' of P that breaks φ_1 is* not *post-collusion secure with respect to φ_2, i.e.*

$$P' \not\models \varphi_1 \implies P' \not\models^{\star} \varphi_2.$$

Informally, this irreversibility relation tells us that any protocol extension that breaks φ_1 inevitably leads to a system in which φ_2 does not hold.

Heuristics for Brute-force Verification of Irreversibility

Definition 4.2 is quantified over all (valid) extensions of a collection of protocol rules. As such, it can present obstacles in providing proofs of security, as the number of extensions is at least exponential in the complexity of the protocol. Indeed, attempting to fully automate this process is an open problem which is also considered by other approaches [BC14, BRS16].

In this subsection we present a heuristic approach for automating the analysis of collusion-resistance, with an implementation for considering the special case of distance-bounding protocols.

Intuition.

Our approach is as follows:

1. We define classes of possible collusion rules that a partially-honest agent may take. For example:
   - Subterm Leakage: Disclosing a subterm of a message that would previously be inaccessible to the adversary
   - Function Oracles: Performing functions such as signing or encrypting on behalf of the adversary without revealing the associated keys
2. For each possible collusion within each collusion class, we define a rule that specifies this deviation from the protocol specification (marking it with the appropriate Collusion facts)
3. For each combination of collusion rules we build a version of the protocol containing these collusion actions.
4. We define a "lower-bound" security goal: violation of this goal should be equivalent to full agent compromise. For example, an agent should never perform collusions that lead to violations of key secrecy.
5. (Automated) analysis is run against each variant of the protocol. An attack is found if there is a variant which satisfies the lower-bound security goal, but not the desired security goal (here, post-collusion security with respect to *dbsec_hnst*).

This approach is not exhaustive, as we are limited by the creativity of the modeller with respect to designing classes of collusion rules. As such, although it can identify attacks, it cannot verify security. However, it can provide increasing levels of confidence as our analysis becomes more fine-grained.

Handling Complexity.

The complexity of this analysis is a major concern. Although distance-bounding protocols are relatively simple compared to many web-based protocols (for example), the cost of running analysis against a number of variants exponential in size to the protocol's complexity is daunting. As we analyse more variants, we are also at risk of accidentally constructing versions of the protocol that are subject to non-termination or similar problems in our automated tools. To help with this, some heuristics can be applied:

- Flagging rules that do not need to be considered for analysis. In the case of distance bounding, our key security claims are all made by agents in the Verifier role. As such we are uninterested in the actions of colluding Verifiers: only fully honest or corrupt ones.

- Pruning the search space based on prior results. We can be sure that the addition of collusion rules will only make the protocol weaker. As such, if a collusion rule is found that leads to a violation of our lower-bound security property, we need not analyse any variant of the protocol that includes this rule.

- Choosing a smart ordering system for examining variants. On a simple level, analysing at most one variant at a time is most likely to weed out individual collusion rules that lead to violations of our lower bound. On the other hand, maximising the set of collusion rules active as much as possible means we are more likely to find an attack (even if only one of the collusion actions is in fact relevant to the attack).

- Using cluster computing to parallelize analysis. Although analysis tools often support running across numerous threads, using a cluster computer to delegate analysis of specific combinations of collusion rules might present an efficiency advantage. As well as improving the rate at which the full search space is covered, this also potentially presents advantages in identifying collusion classes which lead to non-termination of the tool without halting the overall verification process.

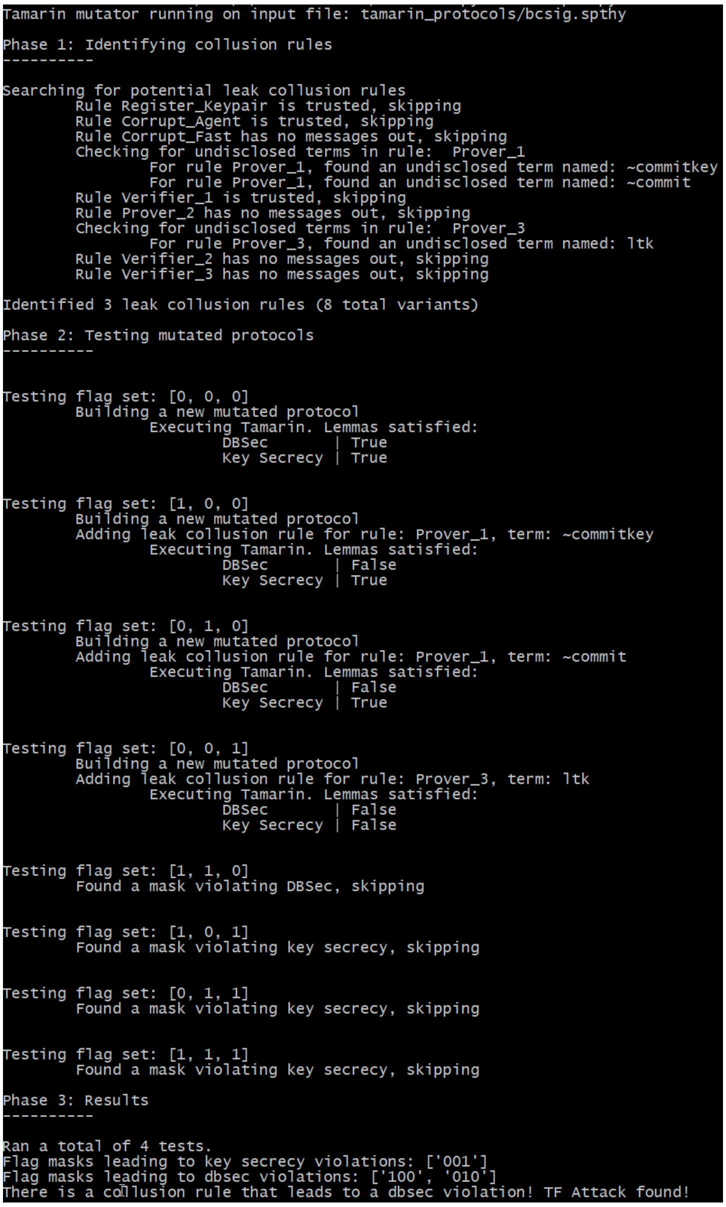
Implementation.

To judge the effectiveness of this approach, we built a command-line tool using the scripting language Python. This tool parses a Tamarin protocol specification file, identifies potential locations for "leak" rules, and builds a variant for each. It then invokes Tamarin against these variants in order to identify all possible combinations of collusion actions that lead to Terrorist Fraud attacks on a given protocol. We implemented all but the last of the above techniques to improve execution time.

An example output of the tool is given in Figure 4.5. Figures 4.6 and 4.7 show some code snippets from the tool.

The tool proves effective against simpler distance-bounding protocols, where the execution time of the original protocol is only a few seconds. In these situations, even a tenfold increase in analysis time is quite tolerable for analysis that considers only leakage rules.

Notably, we discover that for the distance-bounding protocols covered by our analysis, it is sufficient to consider only collusion rules involving subterm leakage in order to identify any attacks. However, as the complexity of a protocol increases, it seems feasible that potential attacks will involve more advanced collusion rules.



```
Tamarin mutator running on input file: tamarin_protocols/bcsig.spthy

Phase 1: Identifying collusion rules
----------

Searching for potential leak collusion rules
        Rule Register_Keypair is trusted, skipping
        Rule Corrupt_Agent is trusted, skipping
        Rule Corrupt_Fast has no messages out, skipping
        Checking for undisclosed terms in rule:  Prover_1
                For rule Prover_1, found an undisclosed term named: ~commitkey
                For rule Prover_1, found an undisclosed term named: ~commit
        Rule Verifier_1 is trusted, skipping
        Rule Prover_2 has no messages out, skipping
        Checking for undisclosed terms in rule:  Prover_3
                For rule Prover_3, found an undisclosed term named: ltk
        Rule Verifier_2 has no messages out, skipping
        Rule Verifier_3 has no messages out, skipping

Identified 3 leak collusion rules (8 total variants)

Phase 2: Testing mutated protocols
----------


Testing flag set: [0, 0, 0]
        Building a new mutated protocol
                Executing Tamarin. Lemmas satisfied:
                        DBSec        | True
                        Key Secrecy | True


Testing flag set: [1, 0, 0]
        Building a new mutated protocol
        Adding leak collusion rule for rule: Prover_1, term: ~commitkey
                Executing Tamarin. Lemmas satisfied:
                        DBSec        | False
                        Key Secrecy | True


Testing flag set: [0, 1, 0]
        Building a new mutated protocol
        Adding leak collusion rule for rule: Prover_1, term: ~commit
                Executing Tamarin. Lemmas satisfied:
                        DBSec        | False
                        Key Secrecy | True


Testing flag set: [0, 0, 1]
        Building a new mutated protocol
        Adding leak collusion rule for rule: Prover_3, term: ltk
                Executing Tamarin. Lemmas satisfied:
                        DBSec        | False
                        Key Secrecy | False


Testing flag set: [1, 1, 0]
        Found a mask violating DBSec, skipping


Testing flag set: [1, 0, 1]
        Found a mask violating key secrecy, skipping


Testing flag set: [0, 1, 1]
        Found a mask violating key secrecy, skipping


Testing flag set: [1, 1, 1]
        Found a mask violating key secrecy, skipping

Phase 3: Results
----------

Ran a total of 4 tests.
Flag masks leading to key secrecy violations: ['001']
Flag masks leading to dbsec violations: ['100', '010']
There is a collusion rule that leads to a dbsec violation! TF Attack found!
```

FIGURE 4.5: Output of collusion rule generating tool, run against a simplified version of the Brands-Chaum protocol.

```
def identify_subterm_leak_mutations(self):
  """
  Finds all possible subterm leak rules for a protocol
  """
  mutations_out = []
  for rule in self.rules:
    outFound = False
    for fact in rule.output:
      if fact.name == "Net":
        outFound = True

      if not outFound:
        continue

      if rule.is_trusted():
        continue

      undisclosed_vars = rule.find_undisclosed_subterms()
      for var in undisclosed_vars:
        mutations_out.append(("leak", rule, var))

  return mutations_out

def find_undisclosed_subterms(self):
  """
  Finds undisclosed subterms in Net messages for a rule
  """
  output_terms = set()

  for fact in self.output:
    if fact.name == "Net":
      allV = find_all_subterms(fact.terms)
      discV = find_disclosed_subterms(fact.terms)
      undisclosed_variables = allV.difference(discV)
      output_terms |= undisclosed_variables

  return output_terms
```

FIGURE 4.6: Identifying Leak collusion rules in a protocol

```python
def build_mutated_protocol(protocol, mutations):
  """
  Applies a list of mutations to a protocol
  """
  output_protocol = copy.deepcopy(protocol)

  for mutation in mutations:
    if mutation[0] == "leak":
      output_protocol.apply_leak_mut(mutation[1], mutation[2])
    ...

  return output_protocol

def apply_leak_mut(self, rule, undisclosed_var):
  """
  Create a subterm-leakage collusion rule for a protocol
  """
  new_mutRule_needed = True
  for mutRule in self.mutatedRules:
    if mutRule.name == rule.name+"_leak_mutation":
      new_mutRule_needed = False
      undisclosed = mutRule.find_undisclosed_variables()
      if undisclosed_var in undisclosed:
        mutRule.output.append(Fact("Net", undisclosed_var))

  if new_mutRule_needed:
    mutRule = copy.deepcopy(rule)
    mutRule.name += "_leak_mutation"
    mutRule.output.append(Fact("Out", undisclosed_var))
    self.mutatedRules.append(mutRule)
```

FIGURE 4.7: Applying Collusion Rules

Ideal Implementations for TF-Resistance

To deal with this completeness issue for the problem of proving terrorist fraud resistance, we introduce the notion of a *least-disclosing* message. Such message is a knowledge-minimal message that the adversary needs, in order to produce the fast phase response upon reception of the challenge. For instance, if C is the verifier's fast phase challenge, and the prover's fast phase response can be written as $f(C, z_1, \ldots, z_n)$ for some $z_1, \ldots, z_n \in \mathit{term}$ such that $\lambda C.f$ is either injective or constant, then a least-disclosing message is $\langle z_1, \ldots, z_n \rangle$. Such message can lead, in some cases, to the disclosure (directly or not) of the long-term keys. To better illustrate the least-disclosing notion, le us consider again the *DBToy* protocol.

Theorem 4.4. *DBToy is resistant to terrorist fraud.*

Proof. Let *DBToy′* be a valid extension of *DBToy* such that $\mathit{DBToy}' \not\models \mathit{dbsec\_hnst}$. Thus, there exist $t_1 \cdots t_l \in \mathit{Traces}(\mathit{DBToy}')$, and $n, m, V, P \in \mathcal{T}_\Sigma$, and $i, k \in \{1, \ldots, l\}$ with $i < k$, such that:

$$\begin{aligned}
&\mathsf{Send}(V, n) \in t_i \wedge \mathsf{Recv}(V, f(n, m, P)) \in t_k \wedge \\
&\mathsf{DBSec}(V, P, n, f(n, m, P)) \in t_l \wedge \\
&(\nexists j \in \{i+1, \ldots, k-1\}.\ \mathsf{Action}(P) \in t_j) \wedge \\
&(\nexists j \in \{1, \ldots, l\}.\ \mathsf{Compromise}(V) \in t_j) \wedge \\
&(\nexists j \in \{1, \ldots, l\}.\ \mathsf{Compromise}(P) \in t_j),
\end{aligned} \tag{4.3}$$

Hence, because of Equation 4.3 above and given the fact that $\mathsf{Recv}(V, f(n, m, P))$ can only occur due to the rule `DBNet` (see Figure 4.2), we derive that:

$$\begin{aligned}
&\exists c, j \in \{1, \ldots, k-1\}, C. \\
&\quad (\mathsf{Send}(C, f(n, m, P)) \in t_j \wedge \mathsf{Compromise}(C) \in t_c).
\end{aligned} \tag{4.4}$$

Equation 4.4 implies that $\exists w < k.\ \mathsf{K}(m) \in t_w$. This means that *DBToy′* has a collusion rule in which m is given away.Notice that m (or equivalently (m, P)) is indeed a *least-disclosing* message because of the following two reasons: m is needed by the adversary to break *dbsec_hnst*, and m is atomic (i.e. it cannot be learned by pieces). But, if the adversary knows m, then they can use a compromised prover to run again the protocol with V on behalf of P, by using the messages $\mathrm{senc}\, mk$ and $f(n_2, m, P)$ in that order, where n_2 is V's (new) challenge. This reasoning can be formalized as follows.

Given that *DBToy′* is valid (see Definition 2.1) we have that $e \geq l$, and $t_{l+1}, \ldots, t_e$ exist such that:

$$t_1 \cdots t_l \cdots t_e \in \mathit{Traces}(\mathit{DBToy}') \wedge \mathit{complete}(t_1 \cdots t_l \cdots t_e). \tag{4.5}$$

Now, $l_2 \geq e$, and $t_{e+1}, \ldots, t_{l_2}$, and n_2, and $i_2, k_2 \in \{e+1, \ldots, l_2 - 1\}$ exist such that:

$$\begin{aligned}
&t_1 \cdots t_l \cdots t_e \cdots t_{l_2} \in \mathit{Traces}(\mathit{DBToy}') \wedge \\
&\mathsf{Send}(V, n_2) \in t_{i_2} \wedge \mathsf{Recv}(V, f(n_2, m, P)) \in t_{k_2} \wedge \\
&\mathsf{DBSec}(V, P, n_2, f(n_2, m, P)) \in t_{l_2} \wedge \\
&(\nexists j \in \{i_2 + 1, \ldots, k_2 - 1\}.\ \mathsf{Action}(P) \in t_j) \wedge \\
&(\nexists j \in \{1, \ldots, l_2\}.\ \mathsf{Compromise}(V) \in t_j) \wedge \\
&(\nexists j \in \{1, \ldots, l_2\}.\ \mathsf{Compromise}(P) \in t_j).
\end{aligned} \tag{4.6}$$

Therefore, from Equations 4.5 and 4.6 we deduce that $\mathit{DBToy}' \not\models^{\star} \mathit{dbsec\_hnst}$, which completes the proof[1]. □

The reasoning about the least-disclosing messages is supported by the observation that any follow-up, collusion-free trace which the adversary can lead to with less knowledge, they can also lead to with further knowledge.

4.5 Case Study: ISO/IEC 14443 Protocols

The ISO/IEC 14443 standard is used in more than 80 contactless smart cards. Within our case studies, we analysed 3 protocols based on this standard. Those protocols are:

- NXP's MIFARE Plus[2] (versions X and EV1) with proximity check (patent [TDJM+11]) with worldwide applications in public transport, access management, school and campus cards, citizen cards, employee cards, and car parking.
- PaySafe [CGdR+15], which is a distance-bounding-enabled version of Visa's contactless payment protocol payWave (in qVSDC mode) [EMV18b].
- PayPass [EMV18a], which is Mastercard's contactless payment protocol with relay resistance.

To demonstrate our analysis, we examine the PayPass protocol, represented in Figure 4.8. The analyses of the other two protocols are analogous. In the context of these protocols, the verifier R is the reader terminal and the prover C is the card.

PayPass is a relay-resistance-enabled version of the EMV[3] payment protocol implemented in Mastercard's contactless cards. EMV (which stands for Europay, Mastercard and Visa) has become the international standard for smart cards/chips payment protocols.

In a regular EMV session, a transaction is initiated by the exchange of SELECT and SELECTED commands along with the selected EMV applet that will be used for the transaction (PayPass in this case). Then, the terminal issues the GPO command to

[1] A TAMARIN proof for a given DBToy' is also available in our repository.

[2] https://www.mifare.net/en/products/chip-card-ics/mifare-plus

[3] https://www.emvco.com

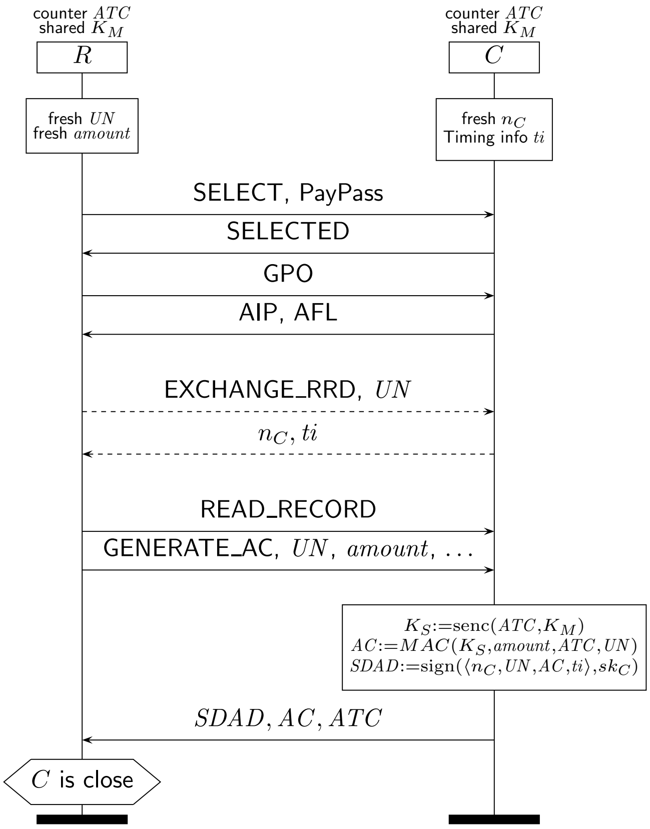


FIGURE 4.8: Mastercard's PayPass protocol.

inform the card on the terminal's capabilities. The card then responds to this command with the Application Interchange Profile (AIP) and Application File Locator (AFL) which indicate the card's capabilities and the location of data files, respectively. Then, the terminal issues the GENERATE_AC command, which includes an Unpredictable Number *UN*, the amount of the transaction, the currency code, and other data. The cards responds with the Application Cryptogram ($\sim$), the Signed Dynamic Application Data (*SDAD*) and the Application Transaction Counter (*ATC*). The $\sim$ is a the result of keyed-MAC on the transaction information whose key is an encryption of the Application Transaction Counter (*ATC*, equal to the number of transactions previously made by the card) with a long-term symmetric key shared between the terminal and the card. The $\sim$ is the proof of the transaction, which can be verified by the card issuer. The *SDAD* is the card's signature of the transaction information.

To ensure the EMV protocol satisfies relay resistance, after the AIP and AFL commands, the terminal issues the new Exchange Relay Resistance Data EXCHANGE_RRD command, along with the Terminal Relay Resistance Entropy number (which equals *UN*). This message initiates the fast phase of the protocol. The card must respond on time with their nonce n_C (Device Relay Resistance Entropy) and three timing estimates: minimum time for processing, maximum time for processing and estimated transmission time.

When modelling the PayPass protocol in TAMARIN, and also the other ISO/IEC 14443 protocols, we made the following abstractions: (1) the timing information is considered a nonce; and (2) we did not model any exchanged messages that are fully composed of constant terms, e.g. the first message $(\mathsf{SELECT}, \mathsf{PayPass})$.

As Table 4.1 shows, PayPass satisfies *dbsec_hnst*, which means that it does resist mafia fraud and in particular, relay attacks. Indeed, defending against relay is a fundamental security goal of this protocol. However, PayPass fails to defend against distance fraud [Des88] and distance hijacking [CRSC12]. Those attacks refer to a remote and compromised card which successfully tricks the reader into believing they are co-located, and thus the reader accepts the transaction.

One might argue that those attacks are irrelevant for payment systems. After all, it is the compromised card's owner's bank account which ends up being charged. However, suppose an attacker has acquired the payment card of a victim and wishes to cause them harm. After compromising the card, they might place a concealed device near the checkout area of a store that performs a distance hijacking attack using the compromised card. Shoppers at the store would then perform transactions, believing that they were paying for products, whilst in fact all payments came from the one corrupted card. The attacker could even mix in several transactions of their own, which would be indistinguishable from the honest shoppers. As a result of this "Robin Hood" style attack, the victim will be charged for these illegitimate transactions with no clear perpetrator.

Fixing the ISO/IEC 14443 Protocols.

As before, we will focus on the PayPass protocol. A distance fraud attack is possible on this protocol, as there is no causal relation between the fast phase challenge and response. That is, the fast phase response can be produced prior to reception of the challenge. One possible way to resolve this is to include the reader's nonce *UN* within the card's response.

Although this approach does prevent distance fraud, it does not prevent distance hijacking. To prevent the latter, we must bind the fast phase messages to the card's identity. We do so by adding to the card's fast phase response, besides *UN*, the card's signature on the nonce n_C. Thus, the card's fast phase response becomes:

$$(n_C, ti, \mathsf{sign}\, n_C sk_C, UN)\,.$$

This modification results in a protocol PayPass_Fix that satisfies *dbsec*. Observe that the signature $\mathsf{sign}\, n_C sk_C$ can be computed prior to the fast phase, so it does not delay the card's response.

The very same solution of adding $(\mathsf{sign}\, n_C sk_C, UN)$ into the card's fast phase response works for both the PaySafe and MIFARE Plus protocols as well. Though, to keep consistency with the usage of cryptographic operations in the case of the latter protocol, we propose a keyed-MAC message $MAC(K_M, n_C, \text{'1'}, \text{'2'})$ instead of the signature $\mathsf{sign}\, n_C sk_C$. As before, the keyed-MAC message can be computed prior to the fast phase.

The modified protocol PayPass_Fix does not resist terrorist fraud, because the card's leakage of $(n_C, ti, \mathsf{sign}\, n_C sk_C)$ prior to the fast phase leads to a valid attack. To prevent terrorist fraud, we propose to further modify the PayPass protocol by changing the card's fast phase response and *SDAD* messages so that they become:

$$(n_C, ti, f(UN, n_C \oplus K_M)) \quad \text{and} \quad \text{sign}\,(UN, \sim)sk_C,$$

respectively; where f is an irreversible function. The referred modification on PayPass results in a protocol PayPass_FixTF that satisfies *dbsec* and resists terrorist fraud.

Similar constructions can be performed on PaySafe and MIFARE Plus in order to repair them. The TAMARIN models and security proofs of the two versions of each protocol are available in our repository. We give two different repaired versions of each protocol in order to leave the choice up to the requirements of the application system. For example, if terrorist fraud is not a critical issue, then the first modification (i.e. Protocol_Fix) is suggested over the second one (i.e. Protocol_FixTF) as the latter modifies the standard more "aggressively". We do always suggest the first modified version over the original protocol, regardless of the application.

Other modifications for the ISO/IEC 14443 protocols that make them resistant to terrorist fraud possibly exist, and likely all of them (like ours) would require major changes to the standard. For example, the composition of the *SDAD* message would likely have to be modified due to the occurrence of the card's nonces within it. Furthermore, we conjecture that if the card's nonces (e.g. n_C) can be inferred from passive observation of the execution, then versions of the protocols in question that resist terrorist fraud would be vulnerable to relay attacks, thus violating a primary security goal of these protocols.

4.6 Results and Conclusions

A Survey of Distance Bounding Protocols

We conducted verification in TAMARIN of a number of distance-bounding protocols from the literature. For each of them, we verify whether it satisfies *dbsec_hnst* (without collusion), whether it satisfies *dbsec* (also without collusion) and whether it resists terrorist fraud (Definition 4.2). The results are shown in Table 4.1.

We remark that the Tree-based, Poulidor, Hancke and Kuhn's and Uniform protocols have equivalent Tamarin implementation as their symbolic formalization is the same. Similarly, the Brands and Chaum's (BC) protocol versions with Fiat-Shamir and Schnorr identification schemes have also the same representation. When verifying these two versions of the protocol, we found a distance-fraud attack against them. However, as the authors have pointed out, such an attack is no longer possible if a challenge/response causal relation is used during the fast phase, such as the XOR operation employed in the signature-based version of the protocol.

To identify the type of attack against a given protocol, we make two observations: (1) if the protocol does not satisfy *dbsec_hnst*, then a mafia fraud exists; and (2) if the protocol satisfies *dbsec_hnst* but it does not satisfy *dbsec*, then a distance fraud and/or a distance hijacking exist. In this second case, it is highly recommended to run TAMARIN in interactive mode and inspect the trace invalidates the property

Protocol	**Satisfies**	**Satisfies**	**Resists**
	dbsec_hnst	*dbsec*	**TF**
Brands-Chaum			
- Signature id.	✓	×	× (!)
- Fiat-Shamir id.	✓	×	× (!)
CRCS			
- Non-revealing sign.	✓	✓	×
- Revealing sign.	✓	×	×
Meadows et al.			
- $f := (N_V, P \oplus N_P)$	✓	×	×
- $f := N_V \oplus h(P, N_P)$	✓ (!)	✓ (!)	× (!)
- $f := (N_V, P, N_P)$	✓ (!)	✓ (!)	× (!)
Lookup-based	✓	✓	×
- Tree			
- Poulidor			
- Hancke-Kuhn			
- Uniform			
Munilla-Peinado	✓	✓	× (!)
Kim-Avoine	✓	✓	× (!)
Reid et al.	✓	✓	✓ (!)
MAD (one way)	✓	×	×
DBP	✓ (!)	✓ (!)	✓ (!)
Swiss Knife	✓	✓	✓ (!)
UWB			
- PKI	× (!)	× (!)	✓ (!)
- keyed-MAC	× (!)	× (!)	✓ (!)
WSBC+DB	✓ (!)	× (!)	× (!)
Hitomi	✓ (!)	✓ (!)	× (!)
TREAD			
- Asymmetric	× (!)	× (!)	✓ (!)
- Symmetric	✓ (!)	× (!)	✓ (!)
ISO/IEC 14443			
- PaySafe	✓ (!)	× (!)	× (!)
- MIFARE Plus	✓ (!)	× (!)	× (!)
- PayPass	✓ (!)	× (!)	× (!)

TABLE 4.1: Results of Tamarin Automated Analysis of Distance-Bounding Protocols. (!) indicates a result is novel, or differs from previous analysis.

dbsec in order to visually assert the existence of the attack. Further details on this can be found in our repository. Out of the analysed protocols, only three protocols are distance-bounding secure and resist terrorist fraud. These protocols are Reid et al. 's [RNTS07], DBPK [BB05], and Swiss Knife [KAK+08]. A total of nineteen protocols were found vulnerable to terrorist fraud.

The authors of UWB impulse radio based protocol [KLT10] do not give precise specifications of their *secure* channel. Hence we employed two schemes: asymmetric encryption/decryption and a message authentication code (MAC). We found a mafia fraud against each variation. Such attack is not reported in [KLT10], as the authors only consider verbatim relay.

For each one of the protocols reported as *not* resistant to terrorist fraud, the valid extension used to invalidate Equation 4.3 is the prover's leakage of the least-disclosing message, whose notion was discussed in 4.4. For each protocol P reported as resistant to terrorist fraud, one of the following three cases occurred:

1. $P \not\models \mathit{dbsec\_hnst}$ and $P \not\models^\star \mathit{dbsec\_hnst}$, thus $P' \not\models^\star \mathit{dbsec\_hnst}$ for any valid extension P' of P, because traces$(\subseteq)\mathit{Traces}(P')$. The protocols of this type are TREAD [ABG+17] with asymmetric encryption, and both versions of UWB [KLT10].

2. Every valid extension P' of P such that $P' \not\models \mathit{dbsec\_hnst}$ leads to replay of an attack on *dbsec_hnst*, therefore $P' \not\models^\star \mathit{dbsec\_hnst}$. The protocol of this type is TREAD [ABG+17] with symmetric encryption.

3. Every valid extension P' of P such that $P' \not\models \mathit{dbsec\_hnst}$ leads to disclosure of the symmetric key shared by the prover and verifier, therefore $P' \not\models^\star \mathit{dbsec\_hnst}$. The protocols of this type are Reid et al. [RNTS07], DBPK [BB05], and Swiss-Knife [KAK+08].

The proofs of terrorist fraud resistance of the four protocols from the last two cases were constructed by following the *semi-automatic* approach based on least-disclosing messages, applied to the *DBToy* protocol proof of Section 4.4. It is a semi-automatic approach, because TAMARIN alone cannot faithfully verify that Definition 4.2 holds for any protocol. This is because of the complexity of handling the universal quantifier over all valid collusion extensions. However, a simple manual proof analogous to that of the *DBToy* protocol, in combination with the tool successfully led to security proofs of the referred protocols.

On average, a TAMARIN model of a protocol consists of about 260 lines of code, out of which 170 are of generic code, approximately. On a modern laptop, the verification of all lemmas for a given protocol takes about half of a minute on average and a few seconds in most cases – some specific protocols with complex equational theories such as the exclusive-or operator take the prover additional time to analyse, increasing the average. All (in)security proofs were constructed without any proof oracles for speeding up the verification.

Conclusions

We have addressed symbolic analysis of security protocols in the presence of colluding agents. Colluding agents are agents who are not under full control of the adversary, yet they are willing to deviate from the intended protocol execution with the goal to invalidate a given property. By looking at different use-cases, we observe that post-collusion security may or may not be a desirable goal. This is because the risk of irreparable damage to the security of a protocol may motivate agents to avoid collusion.

We proposed a concrete symbolic formulation of post-collusion security that can be implemented in state-of-the-art protocol verification tools such as TAMARIN. We used our definition to illustrate that leakage of session data can lead to impersonation of agents. This is particularly interesting in the context of authentication properties in which agents, by leaking only session-fresh data, enable the adversary to successfully break the authentication property in every session thereafter. By means of post-collusion security, we provided the first formal symbolic definition of (resistance to) the sophisticated terrorist fraud attack against distance-bounding protocols. By using our theoretical model and the TAMARIN tool, we provided computer-verifiable proofs of the (in)security of over 25 distance-bounding protocols that account for all classes of attacks, as given by the literature on distance bounding. To the best of our knowledge, this is the most extensive and sound set of security/vulnerability proofs within this research subject.

Our verification reports that for the vast majority of the analysed protocols at least one attack exists. The vulnerable protocols include protocols based on the ISO/IEC 14443 standard such as Mastercard's PayPass [EMV18a], Visa's payWave with distance-bounding [CGdR+15], and NXP's MIFARE Plus with proximity check [TDJM+11]. Finally, we proposed fixes for these protocols and provide computer-verifiable security proofs of the repaired protocols. The proposed fixes form demonstrative examples that could be used to improve proximity-based secure systems that follow the standard, or may even form guidance for a future version of the standard itself.

Chapter 5

Desynchronisation Resistance

Key-updating protocols form a class of communication protocols in which participants change their encryption keys between executions. There are several formally demonstrated security properties which demonstrate the benefits of such protocols. For example, forward privacy prevents an attacker from learning about past sessions, even after compromising a participant. However, such protocols are presented with a new problem: requiring participants to synchronise their key updates and maintain consistent states.

As a result, a new family of attacks has arisen. In some cases, the adversary may cause agents to update their keys in an improper manner, preventing them from correctly interpreting communcations from their partner. This kind of Denial-of-Service attack is called a desynchronisation attack. Such attacks allow an adversary to prevent future runs of a communication protocol, stopping the protocol from achieving its intended purpose.

Key updating protocols are frequently used for RFID devices. One such domain is in grouping protocols, in which multiple provers attempt to simultaneously prove their presence to a verifier. These are often deployed in the supply chain, where key updates allow for untraceability - preventing an attacker from identifying shipments from the radio communication performed during the protocol.

In this chapter we introduce a framework for describing key-updating protocols. Within this model we give a definition for desynchronisation resistance. Finally, we prove a set of under- and over- approximations of desynchronisation resistance, given in terms of verifiable security properties.

The work in this chapter is based on the ESORICS 2018 Paper "Automated Identification of Desynchronisation Attacks on Shared Secrets" [MSTPTR18]. Some content that was cut from the paper for conciseness has been re-added.

5.1 Introduction to Key Updating Protocols

Modelling key-updating protocols presents an additional challenge because they are intrinsically stateful. This can cause issues in analysis because of explosions in the state space. Further, desynchronisation resistance - ensuring that the protocol can always meaningfully continue - is a liveness property, which are known to have a significant computational complexity.

A significant obstacle is present in analysing key-updating protocols in that they are inherently *stateful*. That is, information is carried between sessions, and our security goals must respect this. This can cause problems in analysis due to the explosion of the state space. Indeed, reachability queries are in general an undecidable problem [DLM04, Bla11].

Existing formalisms of desynchronisation resistance.

Desynchronisation represents a class of attacks that are not covered by traditional definitions. A protocol that is impervious to such attacks is said to be *desynchronisation resistant*, and while there is a strong intuitive understanding of what this property means, there are few attempts at formal definitions in the literature.

A recurring theme in desynchronisation resistance is the idea of safety *in absence of the adversary*. This is comparable to the work on post-compromise security by Cohn-Gordon et al. [CCG16] discussed in the previous chapter. Indeed, post-compromise security is usually a (sometimes unintended) side effect of key updating protocols, particularly those that involve key updates using session data. This is often not the case for RFID protocols, which tend to use deterministic updates such as hashing the previously used key.

There exist a variety of works that either claim a form of desynchronisation resistance [SDZ13, LXC14, JJ13, SAKM15] or provide a desynchronisation attack on published protocols [KP09, LW07, SZ16]. Both types of papers only provide an informal treatment of the topic, without automated tool support. Only few papers provide a formal definition of a desynchronisation attack or desynchronisation resistance. We will briefly discuss two of these approaches, namely the work of Van Deursen et al. [vDMRV09] and the work of Radomirović and Dashti [RD15].

Van Deursen et al. [vDMRV09] introduce desynchronisation in the context of RFID protocols. They say an RFID reader *owns* a tag if it knows a secret key allowing it to authenticate the tag in absence of the adversary. A protocol is then said to be desynchronisation resistant if being owned is an *invariant* property. For example, if there is a time t such that a tag T is owned by a reader R, then at time $t+1$ there must exist some reader R' (who may be the same or different to R) which 'owns' T. The authors demonstrate how existing RFID protocols violate their definition. They do not provide, however, any means for formally verifying that it holds for an arbitrary protocol.

A second existing approach that relates to desynchronisation resistance is the work on *derailing attacks* by Radomirović and Dashti [RD15]. In a *derailing attack*, a protocol is led away from its intended execution by an adversary. Reachable states in the protocol are labelled as *safe*, *unsafe*, or *transitional*, describing whether a desirable 'success' condition is reachable from the current point. A protocol is said to be *susceptible to derailing attacks* if there exists a reachable state S such that in absence of the adversary, there are no safe states that are reachable from S.

Contributions.

In this chapter, a formal definition of desynchronisation resistance is given in terms of the traces of a security protocol. The definition we provide can be seen as an extension of the two theories above. Like Radomirović and Dashti, our definition concerns the reachability of certain states, and an examination of the transitions between them. Like Deursen et al., the knowledge of secret keys is an important factor in our definitions. However, we go further by providing a set of conditions for key-updating protocols that allows for automated verification (or falsification) of desynchronisation.

Although traditional security protocol verification tools allow for *reachability* queries, they lack inherent support for the *liveness* properties that we are verifying. As such, we provide under- and over- approximations in the form of verifiable security properties.

Organisation.

This chapter is organised as follows. In Section 5.2 we introduce a set of definitions for reachability and desynchronisation-resistance, and build a framework for key-updating protocols by breaking them down into a setup phase and subsequent rounds of execution. In Section 5.3, we demonstrate upper- and lower- bounds for desynchronisation resistance within our framework, allowing us to identify attacks (or verify) protocols. Sections 5.4 and 5.5 contain two case studies of these bounds, identifying attacks on key-updating protocols from the literature.

5.2 A Framework for Key Updating Protocols

In this Section we enhance our framework for modelling protocols from Section 2.1 in order to build a framework for describing key-updating protocols. In particular, we will introduce *reachability*. Reachability is a property describing the ability of the protocol to transition from a given state to some desirable situation. Indeed, all authentication security goals we consider are in some sense reachability goals.

In particular, we will refine an intuitive definition of reachability into progressively stronger versions, before introducing our definition of *desynchronisation resistance*. The intuition behind desynchronisation is that the protocol reaches a state from which it can no longer proceed in a meaningful way. From this it follows that a desynchronisation resistant protocol is one in which the adversary cannot prevent the protocol from completing, but rather only delay it.

We begin as follows:

Definition 2.1 (State Reachability)**.** *Given a protocol $P = (R, \mathsf{F}, \Sigma, E)$, a set of rules $W \subseteq R$ and two states S, S', we say that S' is* reachable from S avoiding W, *denoted by $S \rightsquigarrow_{\neg W} S'$, if:*

$$\begin{aligned}&\forall \tau \in \mathit{Traces}(P).\, \mathrm{lastState}(\tau) = S \implies \\ &\quad \exists \tau' \in \mathit{Traces}(P). \tau \sqsubseteq \tau' \wedge \mathrm{lastState}(\tau') = S' \wedge \mathrm{rules}(\tau' \setminus \tau) \cap W = \emptyset.\end{aligned}$$

Note that we pay particular attention to the idea of reachability avoiding certain rules. We wish to show that no matter which actions an adversary takes, it is possible for the execution of a protocol to continue once the adversary becomes inactive. As such, we use $\rightsquigarrow_{\neg Adv}$ to denote reachability in absence of the adversary (i.e. without any adversary rules), and $\rightsquigarrow$ for the particular case when no rules are forbidden.

Given a protocol P and a state $S \in \mathbb{U}(\Sigma)$ we define the set of states reachable from S as $\mathrm{reachable}(S) = \{S' \in \mathbb{U}(\Sigma) \mid S \rightsquigarrow S'\}$. Overloading notation, we define the set of states reachable by P as $\mathrm{reachable}(P) = \bigcup_{\tau_0} \mathrm{reachable}(S^0)$.

Next, the notion of reachability is extended from the context of states to the context of event facts.

Definition 2.2 (Event Reachability)**.** *Let P be a protocol, $S \in \mathbb{U}(\Sigma)$ a state, W a set of rules and* E *an event fact. We say that* E *is* reachable from S avoiding W, *denoted by $S \rightsquigarrow_{\neg W}$* E, *if:*

$$\exists S' \in \mathbb{U}(\Sigma).\ (S \rightsquigarrow_{\neg W} S') \wedge (|\mathrm{E}|_S < |\mathrm{E}|_{S'}).$$

Intuitively, given a trace τ that contains S, it is possible to extend τ in such a way that the event fact E is reached. Like before, we will write $S \rightsquigarrow$ E to indicate $S \rightsquigarrow_{\neg\emptyset}$ E.

Reachability captures the idea that a desired state or event can be achieved once. However, we desire that our protocol not only be able to successfully complete once, but arbitrarily many times. To do this, we need a definition stronger than standard reachability. Similar to the $\mathsf{End}(x)$ event fact from the previous chapter, define the Complete event fact as follows:

- $\mathrm{Complete}(\mathit{pub}, \mathit{pub})$ indicates that the first agent believes they have successfully completed a run of the protocol with the second.

Desynchronisation occurs when two agents who were originally able to finish a protocol execution lose this ability.

Definition 2.3 (Desynchronisation Resistance)**.** *A protocol P is* desynchronisation resistant *if:*

$$\begin{aligned}&\forall A, B \colon \mathit{pub},\ P \rightsquigarrow_{\neg Adv} \mathrm{Complete}(A, B) \implies \\ &\quad \big(\forall \tau \in \mathit{Traces}(P).\ \mathrm{firstState}(\tau) = S^0 \implies \\ &\qquad \mathrm{lastState}(\tau) \rightsquigarrow_{\neg Adv} \mathrm{Complete}(A, B) \vee \\ &\qquad\qquad \mathrm{Corrupt}(A) \in \tau \vee \\ &\qquad\qquad \mathrm{Corrupt}(B) \in \tau\big).\end{aligned}$$

Intuitively, if A and B are able to complete the protocol once without any actions being performed by the adversary, then they will always be able to do this, except in the case that one of the participants been corrupted, giving secret data to the adversary.

A Sequential Key Updating Environment

In order to simplify our search space, we now introduce the notion of *starting states*. Intuitively, we will split our protocol rules into a "preparation" phase: where keys between agents are established and corruption is determined, and an "execution" phase: in which the protocol actually takes place.

By their nature, key-updating protocols are such that each execution of the protocol will leave us in a state that is reducible to a starting state. We will assume that protocols have a mechanism in place for handling execution being aborted part-way through – for example, because the communication partner has gone out of range or offline.

Recall that a protocol specification is defined by a tuple $P = (R, \mathsf{F}, \Sigma, E)$. We provide next a framework composed of Σ, E, and F. Depending on the protocol, it may be necessary to extend the equational theory. The set of rules R is a consequence of the protocol being examined.

We use the standard function symbols and equational theory from Section 2.1.

$$\begin{aligned}
\Sigma = \{&senc\colon msg \times msg \to msg,\ sdec\colon msg \times msg \to msg,\\
&aenc\colon msg \times msg \to msg,\ adec\colon msg \times msg \to msg,\\
&pk\colon msg \to msg, h\colon msg \to msg\}.\\
E = \{&sdec(senc(msg, key), key) = msg,\\
&adec(aenc(msg, pk(ltk)), ltk) = msg\}.
\end{aligned}$$

We require the following set of linear facts:

$$\mathrm{ShKey}(pub, pub, \langle msg, \ldots \rangle),\ \mathrm{Session}(pub, pub, \langle msg, \ldots \rangle),$$

, and event facts:

$$\begin{aligned}
&\mathrm{AddKey}(pub, pub, msg),\ \mathrm{DropKey}(pub, pub, msg),\\
&\mathrm{Complete}(pub, pub)
\end{aligned}$$

The facts ShKey and Session provide information about the knowledge of an agent.

Session facts are used to store session data for a single execution of the protocol. ShKey facts represent their *long term* knowledge, in the form of communication keys for use with a named partner. Note that **unlike** our models in the rest of the document, we do not assume that the ShKey fact is persistent: we will allow it to change as agents update their shared keys. This is represented by rules which have an instance of the ShKey fact in both the pre- and post-conditions, with the inner values (i.e. the subterm of shape $\langle msg, \ldots \rangle$) being updated.

The AddKey and DropKey event facts are used to mark changes to an agent's keystore. Intuitively, they are used to mark an agent's key update actions on the trace. We will formalise this later.

We now introduce the notion of a starting states as the subset of all reachable states, conditional on satisfying a series of conditions. We are indifferent as to the series of rules that lead to these states – intuitively, a starting state indicates a situation where the protocol *could* immediately begin an honest execution, and there are no active sessions.

Definition 2.4 (Starting States)**.** *The set of starting states S^{start} is the set composed of all $S^0 \in \mathbb{U}(\Sigma)$ that satisfy the following conditions:*

$$
\begin{aligned}
&(i)\ \nexists x\colon msg.\, \mathrm{Net}(x) \in S^0,\\
&(ii)\ \nexists A, B\colon pub, y\colon msg.\ \mathrm{Session}(A, B, y) \in S^0,\\
&(iii)\ \forall A, B\colon pub, k_1, \ldots, k_n\colon msg.\ \mathrm{ShKey}(A, B, \langle k_1, \ldots, k_n \rangle) \in S^0 \implies\\
&\qquad \nexists l_1, \ldots, l_m\colon msg, \langle k_1, \ldots, k_n \rangle \neq \langle l_1, \ldots, l_m \rangle.\ \mathrm{ShKey}(A, B, \langle l_1, \ldots, l_m \rangle) \in S^0,\\
&(iv)\ \forall A, B\colon pub, k_i\colon msg.\\
&\qquad \mathrm{ShKey}(A, B, \langle \ldots k_i \ldots \rangle) \in S^0 \implies \mathrm{AddKey}(A, B, k_i) \in S^0,\\
&(v)\ \forall A, B\colon pub, k\colon msg.\ \mathrm{AddKey}(A, B, k) \in S^0 \iff\\
&\qquad \exists k_1 \ldots k_n\colon msg.\ \mathrm{ShKey}(A, B, \langle \ldots k \ldots \rangle) \in S^0 \lor \mathrm{DropKey}(A, B, k) \in S^0\\
&(vi)\ \forall A, B\colon pub, k\colon msg.\ \big(\mathrm{ShKey}(A, B, \langle \ldots k \ldots \rangle) \in S^0 \land ((S^0, K) \vdash k)\big) \implies\\
&\qquad \mathrm{Corrupt}(A) \in S^0 \lor \mathrm{Corrupt}(B) \in S^0.
\end{aligned}
$$

We note the following intuitions behind the above requirements:

1. A starting state may not contain messages.

2. A starting state may not contain session data.

3. An agent stores only one set of keys for use with each potential communication partner.

4. If a starting state contains an agent A who stores a secret key k_i for communicating with an agent B, then there is a corresponding AddKey fact showing that A has added this key.

5. If a starting state contains an AddKey fact, then either the corresponding agent has that key in their knowledge, or there is also a corresponding DropKey fact.
6. If a starting state contains an agent A who stores a secret key k_i for communicating with an agent B, and the adversary knows the value k_i, then either A or B is corrupt.

We point out that a starting state does allow for instances of the Complete event fact. This does not interfere with any reachability claims, as these describe the ability to add *new* instances of these event facts to the trace.

We grant the adversary the additional power to "cancel" the session of an agent, causing them to lose any stored session data. For example, this models the ability of an adversary to block messages sent on the network until an agent assumes their partner has halted communication. This is modelled by the rule `Sess_Cancel`, defined below.

$$\frac{\text{Session}(A, B, y)}{}\ \texttt{Sess\_Cancel}$$

5.3 Defining and Proving Desynchronisation Resistance

In this section we look at the consequences of building models in the framework described in the previous section, and show how we can use key updates to describe an invariant property. We use this to provide 'lower' and 'upper' bounds to desynchronisation resistance, proving that violating this combination of properties results in an attack. Note that other choices of environment could be made depending on the target domain, with comparable results.

We model a synchronous key updating environment, in which a pair of agents each store a number of secret communication keys to be used with their intended partner. In an ideal execution, the keys stored by one agent will always correspond to those stored by their partner.

Given a protocol constructed in the model above, we provide a set of conditions that, combined, are sufficient to satisfy desynchronisation resistance.

Properties of Key Updating Protocols

We start with a predicate stating whether two agents share a *common key* in a given state. Let P be a protocol and $S \in \text{reachable}(P)$. We say that two agents A and B have a common key in S, denoted $\text{CommonKey}_{A,B}(S)$, if and only if:

$$\begin{aligned}&\exists k_1, \ldots, k_n, l_1, \ldots, l_m \colon \mathit{msg}.\ \big(\{k_1, \ldots, k_n\} \cap \{l_1, \ldots, l_m\} \neq \emptyset \wedge \\ &\qquad \text{ShKey}(A, B, \langle k_1, \ldots, k_n \rangle) \in S \wedge \ \text{ShKey}(B, A, \langle l_1, \ldots, l_m \rangle) \in S\big)\ .\end{aligned}$$

Now we define *reachability conditional on a common key* as the property of a protocol that two agents are able to complete the protocol with each other in absence of the adversary if and only if they have a common key.

Property 3.1 (Reachable Conditional on Common Key). *We say that P satisfies* completion conditional on a common key *if:*

$$\forall S^0 \in S^{start}, A, B\colon agent,$$
$$S^0 \rightsquigarrow_{\neg Adv} \mathrm{Complete}(A,B) \iff \mathrm{CommonKey}_{A,B}(S^0).$$

We assert that any reasonable protocol should be able to resume from any state in which a pair of agents still share at least one key between them. As such, our aim is to show that a sufficient condition for desynchronisation resistance for is that two agents sharing a common key is an invariant property – no sequence of actions (short of corruption) can cause an agent to "forget" the last key shared by their partner, and so a pair of communication partners always have at least one key to resume correspondence with after adversarial interference ends.

With this in mind, we now define several other properties describing the nature in which the shared keys used by agents in a protocol are updated. Property 3.2 and Property 3.3 give syntactic requirements on protocols. In particular, we require that a protocol's specification is consistent in the way that ShKey linear facts are modified with respect to the addition of the AddKey and DropKey event facts. We also make the assumption that an agent always stores the same number of encryption keys for communicating with their partner.

Property 3.2 (Well-Formed Key Updates). *A protocol P satisfies* Well-Formed Key Updates *if the following two conditions hold for all rules $r \in R$:*

$$\begin{aligned}
&\mathrm{AddKey}(A,B,k) \in rhs(r) \iff \\
&\quad \big(\exists\, k_1 \ldots k_n, l_1 \ldots l_m.\ \mathrm{ShKey}(A,B,\langle k_1 \ldots k \ldots k_n\rangle) \in rhs(r) \wedge \\
&\quad\quad \mathrm{ShKey}(A,B,\langle l_1 \ldots l_m\rangle) \in lhs(r) \ \wedge\ \forall\, i.\ l_i \neq k\big)\ , \\
&\mathrm{DropKey}(A,B,k) \in rhs(r) \iff \\
&\quad \big(\exists\, k_1 \ldots k_n, l_1 \ldots l_m.\ \mathrm{ShKey}(A,B,\langle k_1 \ldots k \ldots k_n\rangle) \in lhs(r) \wedge \\
&\quad\quad \mathrm{ShKey}(A,B,\langle l_1 \ldots l_m\rangle) \in rhs(r) \ \wedge\ \forall\, i.\ l_i \neq k\big)\ .
\end{aligned}$$

Note that this definition, along with the assumption that the number of encryption keys stored by a party is constant, implies that there cannot be any rules which contain only ShKey() terms on one side. Otherwise, the consumption (or addition) of this fact would change an agent's number of stored keys.

Next we define the *Key Conservation* property. It states that every agent must keep the same number of keys during the execution of the protocol. We also require each rule to consider at most a single shared key fact.

Property 3.3 (Key Conservation). *A protocol $P = (\Sigma, E, R, S^{start})$ satisfies* Key Conservation *if for every rule $r \in R$, and every $A, B\colon agent$, $k_1, \ldots, k_n\colon msg$, there exists an* *instance of* ShKey$(A, B, \langle k_1, \ldots, k_n \rangle)$ *on the left-hand side of r if and only if there is some* $l_1, \ldots, l_n$: *msg such that the right-hand side of r contains* ShKey$(A, B, \langle l_1, \ldots, l_n \rangle)$.

Next we define *Key Uniqueness* as the notion that a given encryption key will only be generated at most once. Once discarded by an agent they will never re-use it, nor can a different pair of agents ever (intentionally or otherwise) generate the same encryption key.

Definition 3.4 (Key Uniqueness). *A protocol P satisfies* Key Uniqueness *if for every* $\tau \in \mathit{Traces}(P)$ *and every* A, B, A', B' : *agent and every* k: *msg with* $\{A, B\} \neq \{A', B'\}$ *it holds that:*

$$\begin{aligned}&\mathrm{AddKey}(A, B, k) \in \tau \implies \\ &\qquad |\mathrm{AddKey}(A, B, k)|_\tau = 1 \wedge |\mathrm{AddKey}(A', B', k)|_\tau = 0.\end{aligned}$$

We next describe the properties of *Key Preparedness* and *Key Resilience*. Together with Key Uniqueness, these are the main security requirements that are to be verified. Intuitively, they provide a semi-strict ordering on the key updates of paired agents.

Definition 3.5 (Key Preparedness for agents A and B). *A protocol P satisfies* Key Preparedness for agents A and B *if*

$$\begin{aligned}&\forall \tau \in \mathit{Traces}(P), \forall k\colon \mathit{msg}, \\ &\qquad \mathrm{AddKey}(A, B, k) \in \tau \implies \mathrm{AddKey}(B, A, k) \leq_\tau \mathrm{AddKey}(A, B, k).\end{aligned}$$

Definition 3.6 (Key Resilience for agents A and B). *A protocol P satisfies* Key Resilience for agents A and B *if*

$$\begin{aligned}&\forall \tau \in \mathit{Traces}(P), \forall k\colon \mathit{msg}, \\ &\qquad \mathrm{DropKey}(A, B, k) \in \tau \implies \\ &\qquad\qquad \mathrm{DropKey}(B, A, k) \leq_\tau \mathrm{DropKey}(A, B, k).\end{aligned}$$

The second case in the Key Resilience claim accounts for the trivial case of a starting state containing DropKey facts for which we cannot be sure of the source.

Desynchronisation Resistance

We note that the above properties are verifiable, either by examination of the protocol specification (3.1, 3.2, 3.3), or through verification of traces in an automated prover tool (3.4, 3.5, 3.6) . We denote the properties as WF, KC, KU, KP and KR respectively for Well Formedness, Key Conservation, Key Uniqueness, Key Preparedness and Key Resilience.

Theorem 3.7 (Sufficiency). *Let* $P = (\Sigma, E, R, S^{start})$ *be a protocol that satisfies Properties 3.1, 3.2, 3.3 and Definition 3.4. P satisfies desynchronisation resistance if for all* $S^0 \in S^{start}$ *and all agents* A, B *such that*
CommonKey$_{A,B}(S^0)$, *one of the following conditions holds:*

- *Key Preparedness (Definition 3.5) for agents A and B holds, and Key Resilience (Definition 3.6) for agents B and A holds, or*
- *Key Preparedness (Definition 3.5) for agents B and A holds, and Key Resilience (Definition 3.6) for agents A and B holds.*

Before we begin the proof of Theorem 3.7, we provide some helper lemmas. We define the *strip*() function, which allows us to transform a state into a starting state.

Definition 3.8 (Strip Function)**.** *We define the function strip(), which maps from states to states. We define strip*(S) *to be the multiset that is equal to* S*, but with all instances of* Session, K *and* Net *removed.*

Lemma 3.9. *Let P be a protocol which satisfies Key Conservation (Property 3.3) and Well-Formed Key Updates (Property 3.2). Suppose* $S \in \text{reachable}(P)$*. Then strip*$(S)$ *is a starting state of this protocol, as per the requirements of starting states in Definition 2.4.*

Proof. Points (i), (ii) and (vi) are immediate from the absence of corresponding facts. (iii) is a consequence of Key Conservation, (iv) and (v) from Well-Formed Key Updates. □

Lemma 3.10. *Let P be a protocol which satisfies Key Conservation (Property 3.3) and Well-Formed Key Updates (Property 3.2), and* τ *a trace of P with final state S. Suppose* γ *is a trace of P with starting state strip*(S) *that contains no adversary rules. Then* $\gamma \cdot \tau \in Traces(P)$ *is a trace extension of* τ*.*

Proof. Suppose $\gamma = (strip(S), r_1\sigma_1 \dots r_n\sigma_n)$. We claim that the series of rule applications $r_1\sigma_1 \dots r_n\sigma_n$ are valid from the state S. Indeed, the rule application $r_1\sigma_1$ can be dependent only on ShKey facts, as these are the only linear facts which can be in a starting state. These facts exist in both S and $strip(S)$. By the same logic, the rest of the series of applications are also valid. □

Theorem 3.7 . Assume that the agents A and B are not corrupt. Without loss of generality, we assume the first case holds - that we have Key Preparedness for A and B, and Key Resilience for B and A.

Our proof proceeds in two steps. First, we show that the common key predicate is sufficient to ensure completion from *any* state, not just the starting states:

$$\begin{aligned} &\forall S \in \text{reachable}(P), \\ &\qquad \text{CommonKey}_{A,B}(S) \implies S \rightsquigarrow_{\neg Adv} \text{Complete}(A, B). \end{aligned}$$

Secondly, we show that the common key property is *invariant*:

$$\begin{aligned} &\forall S \in \text{reachable}(P), r \in R, \\ &\qquad (\text{CommonKey}_{A,B}(S) \wedge S \xrightarrow{r\sigma} S') \implies \text{CommonKey}_{A,B}(S'). \end{aligned}$$

From these two claims, the result will immediately follow. To show the first point, we use the *strip*() function from Definition 3.8. Note that if A and B have a common key in S, then they have a common key in $strip(S)$. Then, by Lemma 3.10, the claim follows.

For the second point, we must show that for any rule application $r\sigma$ in which a DropKey event fact is added, the common key predicate is preserved. Indeed, the well-formedness properties of Property 3.2 ensure that these are the only possible rule applications which can affect the predicate.

Suppose we have $S \in \text{reachable}(P)$ such that $\text{CommonKey}_{A,B}(S)$, and a rule application $r_n\sigma_n$. We split into the cases when $\text{DropKey}(A,B,k)$ is added, or when $\text{DropKey}(B,A,k)$ is added. Suppose now $r_n\sigma_n$ adds $\text{DropKey}(A,B,k)$, then:

$$\begin{aligned}
&\overset{KC}{\Longrightarrow} \exists k' : msg\ . && r_n\sigma_n \text{ adds } \text{AddKey}(A,B,k')\\
&\overset{KP}{\Longrightarrow} \exists i < n\ . && r_i\sigma_i \text{ adds } \text{AddKey}(B,A,k')\\
&\overset{KU}{\Longrightarrow} \nexists j\ . && r_j\sigma_j \text{ adds } \text{DropKey}(A,B,k')\\
&\overset{KDR}{\Longrightarrow} \nexists m\ . && r_m\sigma_m \text{ adds } \text{DropKey}(B,A,k')\\
&\Longrightarrow && \text{ShKey}(B,A,\langle \ldots, k', \ldots \rangle) \in S
\end{aligned}$$

and so now k' is a common key after the rule application. Therefore the Common Key predicate is preserved.

Suppose instead $r_n\sigma_n$ adds $\text{DropKey}(B,A,k)$, then:

$$\begin{aligned}
&\overset{KDR}{\Longrightarrow} \exists i < n\ . && r_i\sigma_i \text{ adds } \text{DropKey}(A,B,k)\\
&\overset{WF}{\Longrightarrow} \exists j < i\ . && r_j\sigma_j \text{ adds } \text{AddKey}(A,B,k)\\
&\overset{KU}{\Longrightarrow} \nexists l \neq i\ . && r_l\sigma_i \text{ adds } \text{AddKey}(A,B,k)\\
&\Longrightarrow && \text{ShKey}(A,B,\langle \ldots, k, \ldots \rangle) \notin S
\end{aligned}$$

and so k was not a common key before the rule application. Therefore since S contained some key k' that was a common key, so does the state after the rule application, and so the common key predicate is preserved. □

Theorem 3.7 provides a set of sufficient conditions to ensure that a protocol in our model satisfies desynchronisation resistance. We provide one example of a *necessary* condition to satisfy desynchronisation resistance: any protocol that fails to meet this condition also fails to provide resistance against desynchronisation attacks.

Theorem 3.11 (Necessity)**.** *Let $P = (\Sigma, E, R, S^{start})$ be a protocol that satisfies Properties 3.1, 3.2, and 3.3. Let $S^0 \in S^{start}$ and* $\text{ShKey}(A,B,k) \in S^0$ *(i.e. A stores exactly one key for B) and assume P does not satisfy Key Preparedness (Definition 3.5) for A and B. Then P either contains no reachable key update rule applications for A, or it does not satisfy desynchronisation resistance.*

Proof. Suppose P contains at least one key update rule for A. We will construct a trace from which the $\text{Complete}(A,B)$ is no longer reachable without adversary interference.

Let $\tau = (S^0, r_1\sigma_1, \ldots, r_n\sigma_n)$ be a trace such that $r_n\sigma_n$ is a key update rule application for A that violates the Key Preparedness property. Consider the state $\mathit{strip}(\text{lastState}(\tau))$. Note this state is reachable from $\text{lastState}(\tau)$ through the rules SH_CANCEL and BLOCK.

By Reachability Conditional on a Common Key (Property 3.1), there exist no traces starting from $\mathit{strip}(\text{lastState}(\tau))$ that lead to the $\text{Complete}(A, B)$ event fact without adversary interference. Thus desynchronisation resistance is violated. □

5.4 Case Study: Grouping protocol of Sundaresan et al.

We end this chapter with two case studies that demonstrate the capability of Theorem 3.11 to identify attacks in an algorithmic manner. The first considers the grouping protocol of Sundaresan, Doss, and Zhou [SDZ15], but a similar attack also exists for another RFID grouping protocol, by Sundaresan, Doss, Piramuthu and Zhou [SDPZ14].

Grouping protocols are a simple multiparty authentication protocol in which several agents (usually RFID tags) aim to simultaneously prove that they are close to a verifier (e.g. an RFID reader). These protocols usually do not use strict timing rounds as in the distance bounding protocols from the previous chapters – instead, the focus is on proving liveness of multiple parties at the same time. This is important for use-cases such as shipping cargo, to show that goods are all indeed transported to the destination together.

In addition, many such protocols also aim to achieve additional security goals such as having only partial trust in the verifier device (who saves a transcript to be later checked by a central server). Importantly, many such protocols aim to achieve a form of unlinkability by using a key-updating step as part of the protocol, in order to ensure that a tag will not respond the same way to the same challenge twice.

Figure 5.1 shows the intended execution of the protocol. This protocol forms a proof of (near-)simultaneous liveness by having each tag perform its cryptographic operations on the previous tag's output. Each tag stores only one cryptographic key at a time, whilst the reader device stores two. In the intended execution, the reader sends two bundles of data – one mixed with each key – and the tag identifies whether or not it needs to update its own key based on which packet it is able to correctly deconstruct.

Ultimately, it is this mechanism which leads to an attack. The adversary constructs a modified replay message, taking advantage of the algebraic properties of the exclusive-OR function. After a tag runs the protocol once, the adversary blocks their reply to the reader and transmits the modified replay message. This replay causes the tag to incorrectly authenticate the adversary as a valid reader, and update their key past a safe threshold. The intended execution of the protocol, and a trace which leads to the attack, can be found in Figure 5.2.

Reader		**Tag**
ID_R, k_R^t, k_R^{t+1}		ID_R, k_{tag}
Fetch TS, $Data$ from server Generate n_R $D \leftarrow Data \oplus n_R$ $I \leftarrow ID_R \oplus h(TS \oplus n_R)$ $\delta_1 \leftarrow h(ID_R \oplus k_R^t) \oplus n_R$ $\delta_2 \leftarrow h(ID_R \oplus k_R^{t+1}) \oplus n_R$		
	$\xrightarrow{TS,D,I,\delta_1,\delta_2}$	

		$n' \leftarrow \delta_1 \oplus h(ID_R \oplus k_{tag})$
		if $ID_R = I \oplus h(TS \oplus n')$:
		    if $n' = (n')^{t-1}$: **abort**
		    else: $k_{tag} \leftarrow h(k_{tag})$
		else:
		    $n' \leftarrow \delta_2 \oplus h(ID_R \oplus k_{tag})$
		    if $ID_R \neq I \oplus h(TS \oplus n')$ **or**
		        $n' = (n')^{t-1}$: **abort**
		$D \leftarrow f(D \oplus n')$

	$\xleftarrow{D \oplus n'}$	

FIGURE 5.1: The grouping protocol of Sundaresan et al.

5.5 Case Study: A Two Round Grouping Protocol

Abughazalah, Markantonakis and Mayes provide a two-round RFID grouping proof protocol [AMM16], which uses updating keys. An RFID tag stores two updating keys, for authenticating itself as well as identifying the group that it is a part of.

A system is in place to allow a tag to re-synchronise its group key if it is absent for a run of the protocol, and does not receive the needed message to cause it to update its key naturally. However, this system allows for replay attacks to cause a tag to desynchronise its personal key with that stored by the verifier.

The analysis of the protocol in Tamarin revealed that it fails to satisfy the conditions of Theorem 3.11, resulting in an attack.

Protocol Description.

The protocol is described in detail in the original paper. Here, we provide a simplified description of the protocol for the sake of conciseness. For example, the attack involves communication only between the reader and a single tag, so we focus on only looking at one tag. We also adopt the slightly adapted notation from Table 5.1. A diagram of the intended execution of the protocol is provided in Figure 5.3.

Attack Trace Description.

The grouping protocol has the advantage of requiring only two exchanged messages

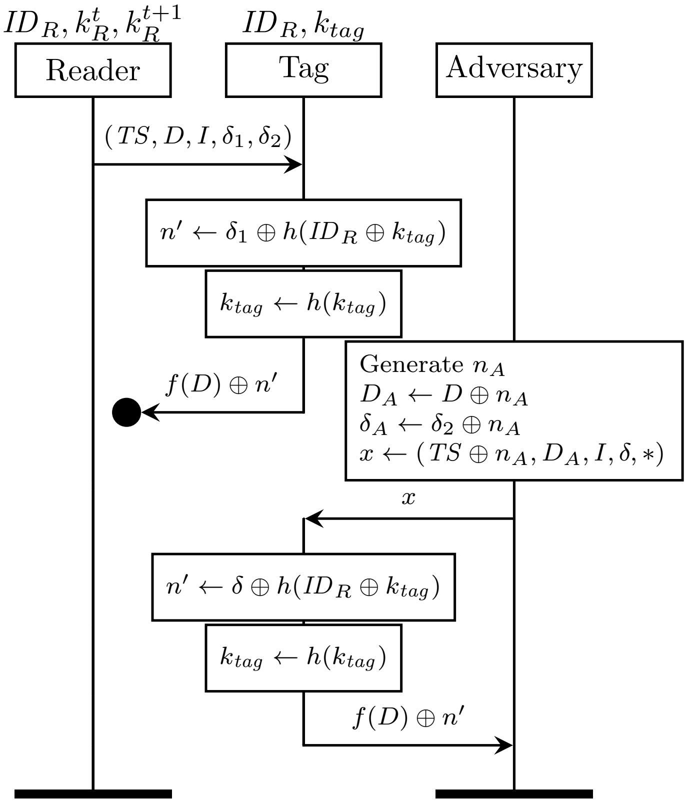

ID_R, k_R^t, k_R^{t+1}
ID_R, k_{tag}
Reader
Tag
Adversary
$(TS, D, I, \delta_1, \delta_2)$
$n' \leftarrow \delta_1 \oplus h(ID_R \oplus k_{tag})$
$k_{tag} \leftarrow h(k_{tag})$
$f(D) \oplus n'$
Generate n_A
$D_A \leftarrow D \oplus n_A$
$\delta_A \leftarrow \delta_2 \oplus n_A$
$x \leftarrow (TS \oplus n_A, D_A, I, \delta, *)$
x
$n' \leftarrow \delta \oplus h(ID_R \oplus k_{tag})$
$k_{tag} \leftarrow h(k_{tag})$
$f(D) \oplus n'$

FIGURE 5.2: Attack on EPC grouping protocol

TABLE 5.1: Notation used in the protocol of Abughazalah et al.

ID_G	The identity of the reader, a secret value
ID_T	The identity of the tag, a secret value
k_G	A secret key for the group being tested
k_T	A secret key for the specific tag being tested
TS^t	An encrypted timestamp, used in construction of the proof
n_R	A fresh (random) nonce generated by the reader
n_T	A fresh (random) nonce generated by the tag
$h(\cdot)$	A cryptographic hash function

during its main execution between the reader and tag. However, this results in a vulnerability which leads to a desynchronisation attacks. The protocol was analysed in Tamarin, with the server role merged into the reader role. This is because the reader and server are assumed to have a secure communications channel.

Note that blocking the tag's message to the reader during a run of the protocol leads to a situation where the tag updates its secret, but the reader will not. The next time the protocol runs, the tag will receive the first message from the reader. Regardless of whether the reader updated the group key k_G (which it may have, because of the presence of other tags in the group completing the protocol), the tag will authenticate to this message and update its key a further time. This attack is indicated in Figure 5.4.

The protocol in the previous section shows one method of attempting to overcome this style of attack, by serving a payload encrypted with several keys and conditionally updating depending on the key used – attempting to ensure that only outdated keys are updated.

The authors seem aware of this problem, and suggest that it is possible for the server to calculate future values of the tag's key in order to prevent desynchronisation. However, there exists the capability to perform replay former messages, causing the tag to update its personal key arbitrarily many times.

As mentioned in the protocol paper, each tag stores previous nonces that they successfully authenticated to. However, an RFID tag has limited memory capacity - a typical EPC Generation 2 tag (such as those mentioned in the paper) has around 512 bits of storage space, meaning that there is very little space to store previously received nonces.

As such, if an adversary is able to eavesdrop at least two runs of the protocol, a tag will readily accept a replay of a message from a previous execution. At this point, the tag will update its key. The adversary can then replay a different message, and repeat this cycle as long as desired.

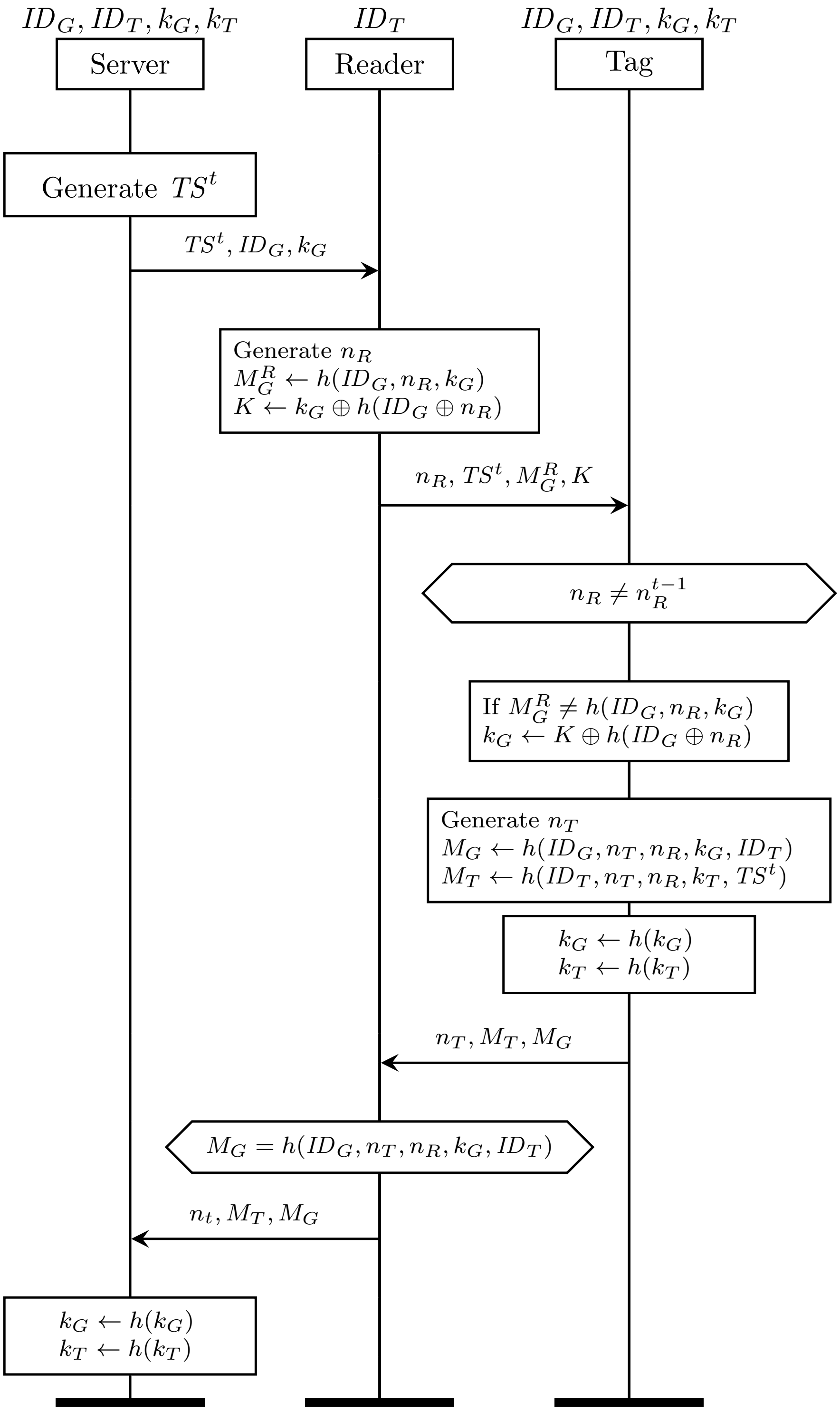


FIGURE 5.3: Two-rounds grouping proof protocol

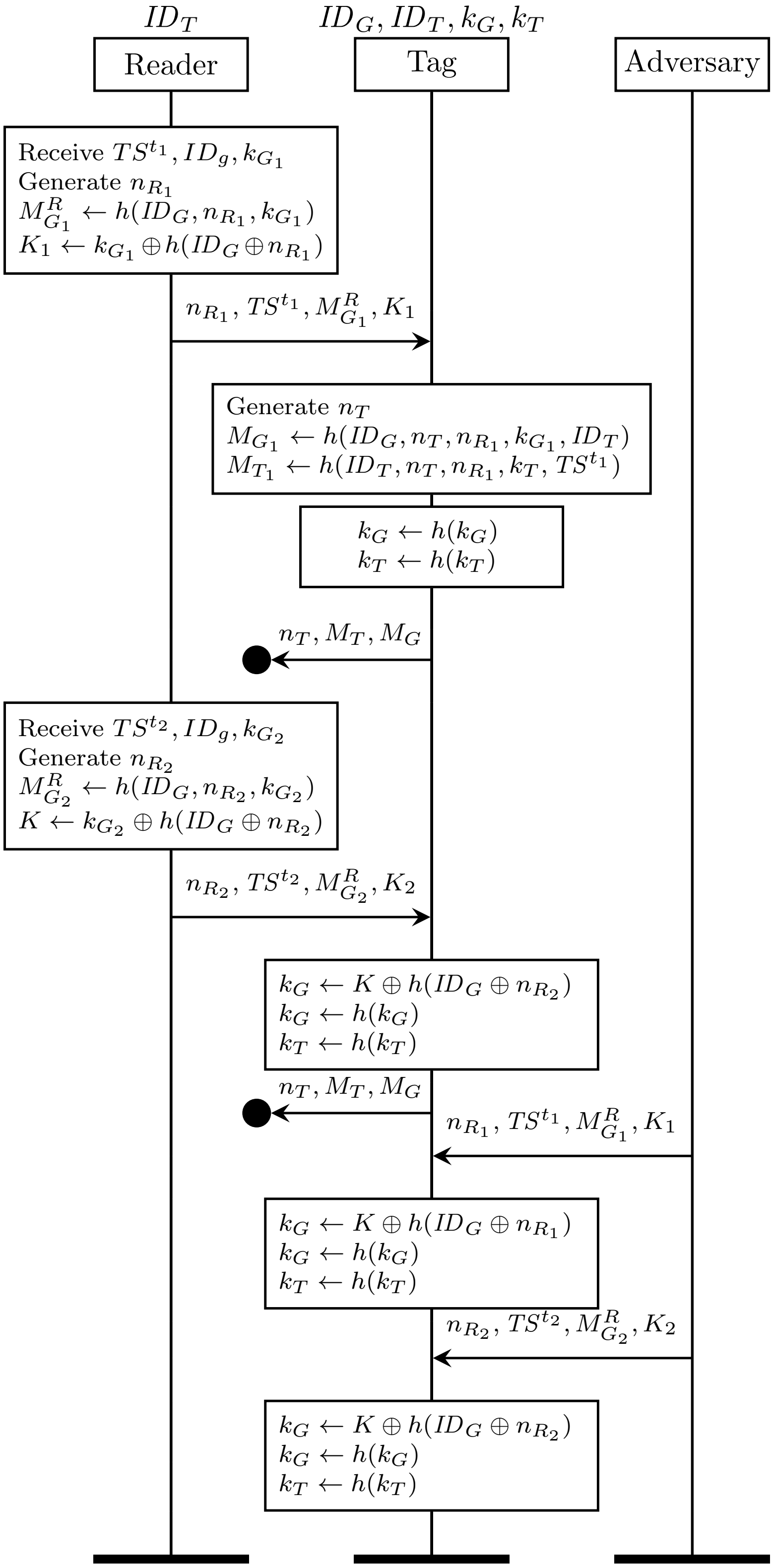

ID_T
ID_G, ID_T, k_G, k_T
Reader
Tag
Adversary
Receive TS^{t_1}, ID_g, k_{G_1}
Generate n_{R_1}
$M^R_{G_1} \leftarrow h(ID_G, n_{R_1}, k_{G_1})$
$K_1 \leftarrow k_{G_1} \oplus h(ID_G \oplus n_{R_1})$
$n_{R_1}, TS^{t_1}, M^R_{G_1}, K_1$
Generate n_T
$M_{G_1} \leftarrow h(ID_G, n_T, n_{R_1}, k_{G_1}, ID_T)$
$M_{T_1} \leftarrow h(ID_T, n_T, n_{R_1}, k_T, TS^{t_1})$
$k_G \leftarrow h(k_G)$
$k_T \leftarrow h(k_T)$
n_T, M_T, M_G
Receive TS^{t_2}, ID_g, k_{G_2}
Generate n_{R_2}
$M^R_{G_2} \leftarrow h(ID_G, n_{R_2}, k_{G_2})$
$K \leftarrow k_{G_2} \oplus h(ID_G \oplus n_{R_2})$
$n_{R_2}, TS^{t_2}, M^R_{G_2}, K_2$
$k_G \leftarrow K \oplus h(ID_G \oplus n_{R_2})$
$k_G \leftarrow h(k_G)$
$k_T \leftarrow h(k_T)$
n_T, M_T, M_G
$n_{R_1}, TS^{t_1}, M^R_{G_1}, K_1$
$k_G \leftarrow K \oplus h(ID_G \oplus n_{R_1})$
$k_G \leftarrow h(k_G)$
$k_T \leftarrow h(k_T)$
$n_{R_2}, TS^{t_2}, M^R_{G_2}, K_2$
$k_G \leftarrow K \oplus h(ID_G \oplus n_{R_2})$
$k_G \leftarrow h(k_G)$
$k_T \leftarrow h(k_T)$

FIGURE 5.4: Attack on two-rounds grouping protocol

Bibliography

[ABG$^+$17] Gildas Avoine, Xavier Bultel, Sébastien Gambs, David Gérault, Pascal Lafourcade, Cristina Onete, and Jean-Marc Robert. A terrorist-fraud resistant and extractor-free anonymous distance-bounding protocol. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017*, pages 800–814, 2017.

[ABK$^+$11] Gildas Avoine, Muhammed Ali Bingöl, Süleyman Kardas, Cédric Lauradoux, and Benjamin Martin. A framework for analyzing RFID distance bounding protocols. *Journal of Computer Security*, 19(2):289–317, 2011.

[AL10] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *J. Cryptology*, 23(2):281–343, 2010.

[AMM16] Sarah Abughazalah, Konstantinos Markantonakis, and Keith Mayes. Two rounds RFID grouping-proof protocol. In *2016 IEEE International Conference on RFID, RFID 2016, Orlando, FL, USA, May 3-5, 2016*, pages 161–174, 2016.

[AT09] Gildas Avoine and Aslan Tchamkerten. An efficient distance bounding RFID authentication protocol: Balancing false-acceptance rate and memory requirement. In *ISC'09*, pages 250–261, 2009.

[BB05] Laurent Bussard and Walid Bagga. Distance-bounding proof of knowledge to avoid real-time attacks. In *Security and Privacy in the Age of Ubiquitous Computing, IFIP TC11 20th International Conference on Information Security (SEC 2005), May 30 - June 1, 2005, Chiba, Japan*, pages 223–238, 2005.

[BBD$^+$91] Samy Bengio, Gilles Brassard, Yvo Desmedt, Claude Goutier, and Jean-Jacques Quisquater. Secure implementations of identification systems. *J. Cryptology*, 4(3):175–183, 1991.

[BC93] Stefan Brands and David Chaum. Distance-bounding protocols (extended abstract). In *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, pages 344–359, 1993.

[BC10] David A. Basin and Cas J. F. Cremers. Modeling and analyzing security in the presence of compromising adversaries. In *Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22, 2010. Proceedings*, pages 340–356, 2010.

[BC14] David A. Basin and Cas Cremers. Know your enemy: Compromising adversaries in protocol analysis. *ACM Trans. Inf. Syst. Secur.*, 17(2):7:1–7:31, 2014.

[BCSS09] David A. Basin, Srdjan Capkun, Patrick Schaller, and Benedikt Schmidt. Let's get physical: Models and methods for real-world security protocols. In *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*, pages 1–22, 2009.

[BD90] Thomas Beth and Yvo Desmedt. Identification tokens - or: Solving the chess grandmaster problem. In *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, pages 169–177, 1990.

[Bla11] Bruno Blanchet. Using Horn clauses for analyzing security protocols. *Formal Models and Techniques for Analyzing Security Protocols*, 5:86–111, 2011.

[BMV13] Ioana Cristina Boureanu, Aikaterini Mitrokotsa, and Serge Vaudenay. Towards secure distance bounding. In Shiho Moriai, editor, *Fast Software Encryption – 20th International Workshop, FSE 2013*, volume 8424 of *LNCS*, Singapore, Republic of Singapore, February 2013. Springer. Invited Talk by Serge Vaudenay.

[BRS16] David A. Basin, Sasa Radomirovic, and Lara Schmid. Modeling human errors in security protocols. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*, pages 325–340, 2016.

[BV14] Ioana Boureanu and Serge Vaudenay. Optimal proximity proofs. In *Information Security and Cryptology - 10th International Conference, Inscrypt 2014, Beijing, China, December 13-15, 2014, Revised Selected Papers*, pages 170–190, 2014.

[CBH03] Srdjan Capkun, Levente Buttyán, and Jean-Pierre Hubaux. SECTOR: secure tracking of node encounters in multi-hop wireless networks. In *Proceedings of the 1st ACM Workshop on Security of ad hoc and Sensor Networks, SASN 2003, Fairfax, Virginia, USA, 2003*, pages 21–32, 2003.

[CCG16] Katriel Cohn-Gordon, Cas J. F. Cremers, and Luke Garratt. On post-compromise security. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*, pages 164–178, 2016.

[CCG$^+$18] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1802–1819, 2018.

[CdRS18] Tom Chothia, Joeri de Ruiter, and Ben Smyth. Modelling and analysis of a hierarchy of distance bounding attacks. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.*, pages 1563–1580, 2018.

[CGdR$^+$15] Tom Chothia, Flavio D. Garcia, Joeri de Ruiter, Jordi van den Breekel, and Matthew Thompson. Relay cost bounding for contactless EMV payments. In *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, pages 189–206, 2015.

[CM12] Cas Cremers and Sjouke Mauw. *Operational Semantics and Verification of Security Protocols*. Information Security and Cryptography. Springer, 2012.

[CO99] Ran Canetti and Rafail Ostrovsky. Secure computation with honest-looking parties: What if nobody is truly honest? (extended abstract). In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 255–264, 1999.

[CRSC12] Cas J. F. Cremers, Kasper Bonne Rasmussen, Benedikt Schmidt, and Srdjan Capkun. Distance hijacking attacks on distance bounding protocols. In *IEEE Symposium on Security and Privacy, S&P 2012, 21-23 May 2012, San Francisco, California, USA*, pages 113–127, 2012.

[DD19] Alexandre Debant and Stéphanie Delaune. Symbolic verification of distance bounding protocols. In *International Conference on Principles of Security and Trust*, pages 149–174. Springer, 2019.

[DDW19] Alexandre Debant, Stéphanie Delaune, and Cyrille Wiedling. Symbolic analysis of terrorist fraud resistance. In *European Symposium on Research in Computer Security*, pages 383–403. Springer, 2019.

[Des88] Yvo Desmedt. Major security problems with the “unforgeable" (Feige)-Fiat-Shamir proofs of identity and how to overcome them. In *SECURICOM'88*, pages 15–17, 1988.

[DFKO11] Ulrich Dürholz, Marc Fischlin, Michael Kasper, and Cristina Onete. A formal approach to distance-bounding RFID protocols. In *Information Security, 14th International Conference, ISC 2011, Xi'an, China, October 26-29, 2011. Proceedings*, pages 47–62, 2011.

[DGB87a] Yvo Desmedt, Claude Goutier, and Samy Bengio. Special uses and abuses of the fiat-shamir passport protocol. In *CRYPTO'87*, pages 21–39, 1987.

[DGB87b] Yvo Desmedt, Claude Goutier, and Samy Bengio. Special uses and abuses of the Fiat-Shamir passport protocol. In *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, pages 21–39, 1987.

[DLM04] Nancy A. Durgin, Patrick Lincoln, and John C. Mitchell. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311, 2004.

[EMV18a] EMVCo. *EMV Contactless Specifications for Payment Systems, Book C-2, Kernel 2 Specification, Version 2.7.* April 2018.

[EMV18b] EMVCo. *EMV Contactless Specifications for Payment Systems, Book C-3, Kernel 3 Specification, Version 2.7.* April 2018.

[FDC11] Aurélien Francillon, Boris Danev, and Srdjan Capkun. Relay attacks on passive keyless entry and start systems in modern cars. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011*, 2011.

[FY92] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 699–710, 1992.

[GAA11] Ali Özhan Gürel, Atakan Arslan, and Mete Akgün. Non-uniform stepping approach to RFID distance bounding problem. In *DPM'10/SETOP'10*, volume 6514 of *LNCS*, pages 64–78, 2011.

[HK05] Gerhard P. Hancke and Markus G. Kuhn. An RFID distance bounding protocol. In *SecureComm'05*, pages 67–73, 2005.

[HM97] Martin Hirt and Ueli M. Maurer. Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract). In *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing, Santa Barbara, California, USA, August 21-24, 1997*, pages 25–34, 1997.

[JJ13] Seung Wook Jung and Souhwan Jung. HRP: A HMAC-based RFID mutual authentication protocol using PUF. In *International Conference on Information Networking (ICOIN)*, pages 578–582. IEEE, 2013.

[KA09] Chong Hee Kim and Gildas Avoine. RFID distance bounding protocol with mixed challenges to prevent relay attacks. In *CANS'09*, pages 119–133, 2009.

[KA11] Chong Hee Kim and Gildas Avoine. RFID distance bounding protocols with mixed challenges. *IEEE Trans. on Wireless Comm.*, 10(5):1618–1626, 2011.

[KAK$^+$08] Chong Hee Kim, Gildas Avoine, François Koeune, François-Xavier Standaert, and Olivier Pereira. The Swiss-Knife RFID distance bounding protocol. In *Information Security and Cryptology - ICISC 2008, 11th International Conference, Seoul, Korea, December 3-5, 2008, Revised Selected Papers*, pages 98–115, 2008.

[KKBD12] Süleyman Kardas, Mehmet Sabir Kiraz, Muhammed Ali Bingöl, and Hüseyin Demirci. A novel RFID distance bounding protocol based on physically unclonable functions. In *RFIDSec'11*, volume 7055 of *LNCS*, pages 78–93. Springer, 2012.

[KLT10] Marc Kuhn, Heinrich Luecken, and Nils Ole Tippenhauer. UWB impulse radio based distance bounding. In *7th Workshop on Positioning Navigation and Communication, WPNC 2010, Dresden Germany, 11-12 March 2010, Proceedings*, pages 28–37, 2010.

[KP09] Gaurav Kapoor and Selwyn Piramuthu. Vulnerabilities in some recently proposed RFID ownership transfer protocols. In *First International Conference on Networks & Communications*, pages 354–357. IEEE, 2009.

[Kra05] Hugo Krawczyk. HMQV: A high-performance secure diffie-hellman protocol. In *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, pages 546–566, 2005.

[LLM07] Brian A. LaMacchia, Kristin E. Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In *Provable Security, First International Conference, ProvSec 2007, Wollongong, Australia, November 1-2, 2007, Proceedings*, pages 1–16, 2007.

[LM06] Kristin E. Lauter and Anton Mityagin. Security analysis of KEA authenticated key exchange protocol. In *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings*, pages 378–394, 2006.

[Low97] Gavin Lowe. A hierarchy of authentication specifications. In *10th Computer Security Foundations Workshop (CSFW '97), June 10-12, 1997, Rockport, Massachusetts, USA*, pages 31–44, 1997.

[LW07] Ticyan Li and Guilin Wang. Security analysis of two ultra-lightweight RFID authentication protocols. In *New Approaches for Security, Privacy and Trust in Complex Environments*, pages 109–120. Springer, 2007.

[LXC14] Qing-Shan Li, Xiao Lin Xu, and Zhong Chen. PUF-based RFID ownership transfer protocol in an open environment. In *15th International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 131–137. IEEE, 2014.

[MBK10] Sreekanth Malladi, Bezawada Bruhadeshwar, and Kishore Kothapalli. Automatic analysis of distance bounding protocols. *CoRR*, abs/1003.5383, 2010.

[MP08] Jorge Munilla and Alberto Peinado. Distance bounding protocols for RFID enhanced by using void-challenges and analysis in noisy channels. *Wireless Communications and Mobile Computing*, 8(9):1227–1232, 2008.

[MPP$^+$07] Catherine A. Meadows, Radha Poovendran, Dusko Pavlovic, LiWu Chang, and Paul F. Syverson. Distance bounding protocols: Authentication logic analysis and collusion attacks. In *Secure Localization and Time Synchronization for Wireless Sensor and Ad Hoc Networks*, pages 279–298. 2007.

[MSCB13] Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, pages 696–701, 2013.

[MSTPTR18] Sjouke Mauw, Zach Smith, Jorge Toro-Pozo, and Rolando Trujillo-Rasua. Automated identification of desynchronisation attacks on shared secrets. In *European Symposium on Research in Computer Security*, 2018.

[MSTPTR19] Sjouke Mauw, Zach Smith, Jorge Toro-Pozo, and Rolando Trujillo-Rasua. Post-collusion security and distance bounding. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2019.

[MSTT18] Sjouke Mauw, Zach Smith, Jorge Toro-Pozo, and Rolando Trujillo-Rasua. Distance-bounding protocols: Verification without time and location. In *2018 IEEE Symposium on Security and Privacy, S&P 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 549–566, 2018.

[MTT16a] Sjouke Mauw, Jorge Toro-Pozo, and Rolando Trujillo-Rasua. A class of precomputation-based distance-bounding protocols. In *EuroS&P'16*, pages 97–111, 2016.

[MTT16b] Sjouke Mauw, Jorge Toro-Pozo, and Rolando Trujillo-Rasua. Optimality results on the security of lookup-based protocols. In *Radio Frequency Identification and IoT Security - 12th International Workshop, RFIDSec 2016, Hong Kong, China, November 30 - December 2, 2016, Revised Selected Papers*, pages 137–150, 2016.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[PTDC11] Christina Pöpper, Nils Ole Tippenhauer, Boris Danev, and Srdjan Capkun. Investigation of signal and message manipulations on the wireless channel. In *ESORICS'11*, pages 40–59, 2011.

[RC10] Kasper Bonne Rasmussen and Srdjan Capkun. Realization of RF distance bounding. In *USENIX Security'10*, pages 389–402, 2010.

[RD15] Sasa Radomirovic and Mohammad Torabi Dashti. Derailing attacks. In *Security Protocols XXIII - 23rd International Workshop, Cambridge, UK, March 31 - April 2, 2015, Revised Selected Papers*, pages 41–46, 2015.

[RNTS07] Jason Reid, Juan Manuel González Nieto, Tee Tang, and Bouchra Senadji. Detecting relay attacks with timing-based protocols. In *Proceedings of the 2007 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2007, Singapore, March 20-22, 2007*, pages 204–213, 2007.

[SAKM15] Keerti Srivastava, Amit K. Awasthi, Sonam Devgan Kaul, and R. C. Mittal. A hash based mutual RFID tag authentication protocol in telecare medicine information system. *J. Medical Systems*, 39(1):153, 2015.

[SDPZ14] Saravanan Sundaresan, Robin Doss, Selwyn Piramuthu, and Wanlei Zhou. A robust grouping proof protocol for RFID EPC C1G2 tags. *IEEE Trans. Information Forensics and Security*, 9(6):961–975, 2014.

[SDZ13] Saravanan Sundaresan, Robin Doss, and Wanlei Zhou. Secure ownership transfer in multi-tag/multi-owner passive RFID systems. In *Globecom 2013 - Symposium on selected areas in communications*, pages 2891–2896. IEEE, 2013.

[SDZ15] Saravanan Sundaresan, Robin Doss, and Wanlei Zhou. Zero knowledge grouping proof protocol for RFID EPC C1G2 tags. *IEEE Trans. Computers*, 64(10):2994–3008, 2015.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[SMC00] Paul Syverson, Catherine Meadows, and Iliano Cervesato. Dolev-Yao is no better than Machiavelli. In *First Workshop on Issues in the Theory of Security, WITS'00, Geneva, Switzerland, July 7-8, 2000*, pages 87–92, 2000.

[SSBC09] Patrick Schaller, Benedikt Schmidt, David A. Basin, and Srdjan Capkun. Modeling and verifying physical properties of security protocols for wireless networks. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF 2009, Port Jefferson, New York, USA, July 8-10, 2009*, pages 109–123, 2009.

[SZ16] Da-Zhi Sun and Ji-Dong Zhong. Cryptanalysis of a hash based mutual RFID tag authentication protocol. *Wireless Personal Communications*, 91(3):1085–1093, 2016.

[TDJM$^+$11] Peter Thueringer, Hans De Jong, Bruce Murray, Heike Neumann, Paul Hubmer, and Susanne Stern. Decoupling of measuring the response time of a transponder and its authentication, March 2011. US Patent No. US12994541.

[TMA10] Rolando Trujillo-Rasua, Benjamin Martin, and Gildas Avoine. The poulidor distance-bounding protocol. In *RFIDSec'10*, pages 239–257, 2010.

[TW86] Martin Tompa and Heather Woll. How to share a secret with cheaters. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, pages 261–265, 1986.

[vDMRV09] Ton van Deursen, Sjouke Mauw, Sasa Radomirovic, and Pim Vullers. Secure ownership and ownership transfer in RFID systems. In *ESORICS'09*, pages 637–654, 2009.

Part II

Multiparty Protocols

Chapter 6

Path Protocols

For the second part of this thesis, our attention turns to security goals for multiparty protocols. Traditionally, many such protocols have been modelled as two-party systems, abstracting away the less relevant parties and focusing on the interaction between two principal agents. However, an increasing quantity of modern research has identified the value of avoiding this coarse-grained approach.

This chapter focuses on the concept of "path protocols". Such protocols involve a message being forwarded through a series of connected agents, who may or may not have access to the message contents. Instead of analysing the security of the payload, this chapter discusses the problem of modelling the integrity of the path itself. Though this formally presents itself as an authentication-style property, the implications of an attack against path integrity can lead to a variety of consequences depending on the specific domain.

We highlight three main usecases where path protocols are deployed. In multiparty TLS, middleboxes augment the ecosystem by granting additional functionality such as content filtering or redirection. In mixnet protocols, message forwarding is deployed to grant anonymity to the endpoints. Finally, payment networks represent a relatively modern class of protocols in which the path of a message is constrained by an underlying network topology - the collection of active channels through which a payment can be routed. Although the purpose of each of these protocol families is different, we argue that they share a common underlying structure, such that we can define security goals which are relevant across all domains.

The work in this chapter is based on a paper titled "Taking Shortcuts: Modelling Agent-Skipping Attacks in Message Forwarding Protocols", which is currently unpublished.
One of the case studies, regarding TLS, is delayed until the following chapter, where it has additional relevance.

6.1 Introduction to Path Protocols

The TLS protocol [DR08, Res18] is used for an overwhelming majority of modern web communication. TLS users often deploy services such as firewalls or load balancers, which intercept messages, redirecting or modifying their contents. These entities, known as *TLS middleboxes*, necessitate modifications to the TLS protocol in order to support multiple parties. Currently, the most common approach in handling this is known as Split TLS [JU12], in which the TLS session is “split" into a series of completely disjoint sessions between each pair of intermediate agents. Security concerns about Split TLS [dCdCM16, DMS$^+$17] have lead to new solutions, such as mcTLS [NSV$^+$15], which propose extensions to the TLS protocol suite.

Mixnets and Onion Routing protocols also involve sending messages along a path. Such protocols originate from Chaum [Cha81] (in this case, with email in mind). Modern protocols such as The Onion Router [SDM04] and Sphinx [DG09] are built in a similar way. A novel application of onion routing is shown in the relatively new field of *payment networks*. Perhaps the most notable of these is the Lightning Network [PD16], proposed in 2016. In this system, Bitcoin transactions are made dramatically faster through pairs of agents maintaining off-chain payment channels. A core component of the Lightning Network is the concept of chained payments, in which an agent can send funds to a peer through a series of such channels.

Several attacks have been highlighted on protocols from each of these domains [KW03, MMSS$^+$19]. Intuitively, these attacks arise from the use of *shortcuts*: redirecting or modifying messages in order to bypass one or more agents in the chain. Although the structure of these attacks is similar between use cases, the implications can differ greatly.

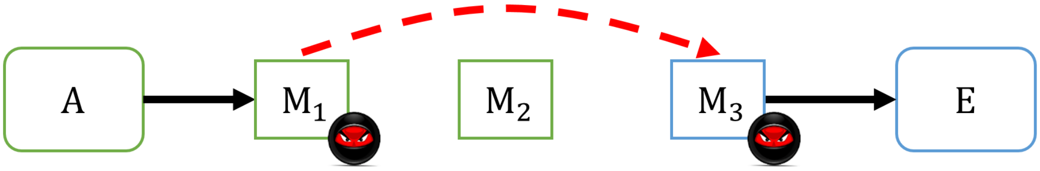


FIGURE 6.1: Agents M_1 and M_3 collude to bypass M_2. Dashed lines indicate out-of-band communication

In order to protect against such attacks, protocols require a notion of *path integrity*. Participating parties must be sure that sent messages can flow only in the way that is intended by the initiating agents. For TLS, the principal agents want to ensure that their middleboxes are being respected. For example, it must not be possible to bypass a content filter, lest malicious injected code could reach an endpoint. For onion-style protocols, the sender wishes to preserve privacy by guaranteeing that the message travels only between the trusted intermediaries. Finally, for payment networks, intermediate agents must be assured that if they assist in a payment by forwarding a message, they are guaranteed to receive their transaction fees during the resolution stage.

Formal methods have been used to analyse security protocols for some time. Many of the core definitions stem from the work of Lowe [Low97]. Symbolic models [Bla12] for analysis of protocols have become popular in recent years due to the increasing effectiveness of automated verification tools such as Tamarin [MSCB13, The18] and ProVerif [Bla13].

In this work we present a theory which links together protocols from each of the different domains under a common framework. Further, we show that each of the attacks found in the literature can be captured as failure to satisfy one of a small set of security goals.

Contributions.
Our contributions are as follows:

- We introduce a symbolic framework, built upon the multiset rewriting model, that can be used to describe the structure of 'path-based' protocols.
- We give a formal definition of the *path integrity* security goal inside this framework.
- We demonstrate the threat posed by message-skipping attacks through case studies on two next-gen protocols: the mbTLS extension to TLS and the Bitcoin Lightning Network.
- We provide a collection of models of a range of protocols from the literature using the Tamarin prover tool, showing the applicability of our framework.

Related Work.
A key focus in the design of Onion routing systems is the consideration of various security tradeoffs with regards to anonymity features [SB08]. However, our review of the literature revealed no formal analysis regarding the integrity of a path in such protocols.

Although significant security analysis has been put into the core TLS protocol suite [CHH+17, BBK17], the security discussion of middlebox-enabled extensions is still somewhat limited. Some formal models have been created for accountable proxying, such as those of Bhargavan et al. [BBD+18, BBF+17]. However, these focus more on end-to-end authentication guarantees in the presence of proxies, avoiding discussion of path integrity. They also use the computational setting, rather than a symbolic model such as the one we present here. The mbTLS [NLG+17] extension for TLS gives an informal description of path integrity, under a different security model; that is, all the agents are trusted based on Intel SGX [Int17]. Another middlebox-oriented TLS extension, maTLS [LSL+19], provides a definition of path integrity for their specific use-case.

When it comes to payment networks, much thought has been put into improving their efficiency – both in optimising the discovery of channels on the network, and in choosing an optimal path for a payment to travel through [RMSKG17, RLT17, PN18]. Malavolta et al. [MMSS+19] describe a wormhole attack on the Lightning network,

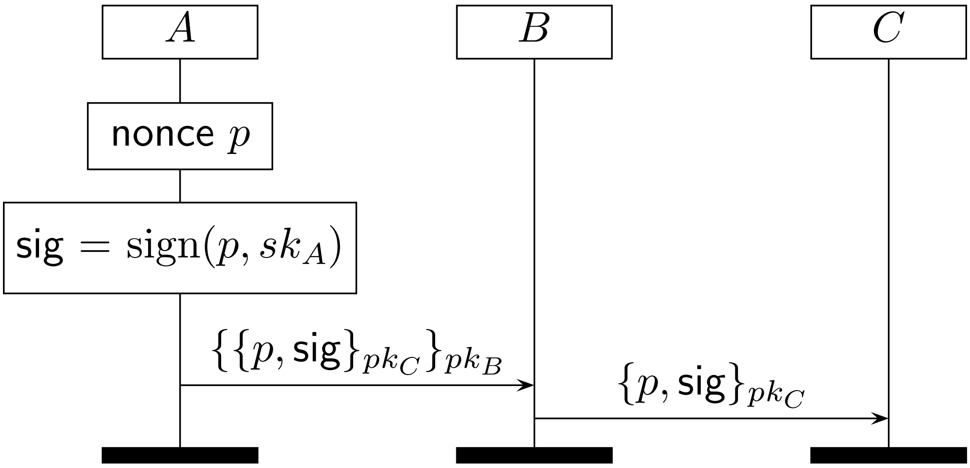

A
B
C
nonce p
sig $= \text{sign}(p, sk_A)$
$\{\{p, \text{sig}\}_{pk_C}\}_{pk_B}$
$\{p, \text{sig}\}_{pk_C}$

FIGURE 6.2: A simple message forwarding protocol.

similar to those demonstrated on wireless sensor networks [KW03], and propose a scheme to fix it. Our model can be used to both detect this attack and prove resistance of their amended version in an automated manner, whilst their proof relies on a lengthy construction in the computational model. Finally, while Malavolta et al. give an impossibility result on the existence of secure two-round payment protocols, we instead show that under our set of assumptions such protocols can indeed be constructed using only standard cryptographic primitives. The full discussion is given in Section 6.4.

Recent work by Bursuc and Kremer [BK19] takes steps into modelling aspects of the Bitcoin cryptocurrency in the symbolic setting. However, to the best of our knowledge, no attempts have been made to analyse the security of the Lightning Network using symbolic models.

6.2 A Framework for Path Protocols

In this section we introduce the notion of a *path protocol*, building on the notation from the previous section. We break down the structure of a path protocol into a set of phases, and describe each of these phases as a collection of generic rules. In the appendices we give several examples of protocols modelled in such a way.

Running Example

Consider the multi-party message forwarding protocol shown in Figure 6.2, which uses Public Key encryption.

Intuitively, the agent A wants an intermediate agent B to forward a message p to C. This is achieved by using nested encryption. The protocol is depicted with one forwarding agent B, but indeed it could be trivially extended for any number of forwarding agents.

It would be relatively straightforward to model this protocol for a fixed number of agents, by specifying the value of the message at each step of execution. However, this design quickly becomes cumbersome as the number of agents grow. Moreover, it requires a separate model for each number of agents. Instead, we construct a single model which accounts for any number of agents.

Modelling Multi-Step Messages

We now build a set of rewriting rules which allow us to specify protocols that use these multi-step messages. Intuitively, such messages are constructed by "wrapping" layers of encryption on top of each other.

This approach will allow us to define *path protocols*, which we break down into a series of phases.

Definition 2.1. *Path Protocols (Notion)*

A Path Protocol is a protocol in which rules (other than those modelling adversary capabilities) can be categorised as belonging to one of the following phases:

- ***Setup Phase**: A preliminary phase in which agents' encryption keys are established*
- ***Construction Phase**: An initial agent creates a message from a payload, by configuring it to pass through a series of intermediate agents*
- ***Forwarding Phase**: Each intermediate agent receives, repackages and forwards the message*
- ***Receive Phase**: The intended recipient receives the final message and retrieves the payload.*

Throughout the course of our discussion, we will generally use the name p to refer to the *payload* – the intended value for the final recipient. The name m will be used to refer to *messages* sent between agents – including things like encryption (which we do model) and header or miscellaneous data (which we do not). We make the assumption that for each individual session, the payload will always contain some session data or randomness that makes it unique, and thus suitable as a session identifier.

Over the course of this section, we build a framework of generic rules which is sufficient to cover each of the individual phases in Definition 2.1. This resulting set of rules can be used to describe a large majority of path protocols. We finish with a full specification of the example protocol from Figure 6.2 in this framework. In Section 6.3 we will discuss extending this model with additional phases to handle certain classes of protocols with additional requirements.

Setup Phase.

Our execution model begins with the empty multiset. In order for the protocol to begin, agents must be instantiated and assigned encryption keys. This includes asymmetric keys owned by individuals, as well as shared keys between pairs of agents.

Our models will make use of the `Gen_ShKey` and `Gen_Ltk` rules for generating encryption keys, specified as follows:

$$\texttt{Gen\_ShKey} := \Big[\ \mathrm{Fr}(k)\ \Big] \longrightarrow \Big[\ !\mathrm{ShKey}(A, B, k)\ \Big]$$

$$\texttt{Gen\_Ltk} := \Big[\ \mathrm{Fr}(\mathrm{ltk})\ \Big] \longrightarrow \begin{bmatrix} !\mathrm{Ltk}(A, \mathrm{ltk}) \\ !\mathrm{Pk}(A, \mathrm{pk}(\mathrm{ltk})) \\ \mathrm{Net}(\mathrm{pk}(\mathrm{ltk})) \end{bmatrix}$$

We also allow for the corruption of agents created during the setup phase. We will assume that the initiating agent in each session is honest, but allow all other agents to be under full adversarial control.

Construction Phase.

The Construction Phase represents the beginning of a session of the protocol. We break this down into three rules – a `Create` rule which determines the payload, a `Wrap` rule which modifies the message to pass through an intermediate agent, and finally a `Send` rule, in which the message is released onto the communication network.

Implicit in the application of these rules is the notion of a *path order*. Intuitively, this is a relation that models the (intended) order in which the message will pass between protocol participants.

Definition 2.2. *Path Order*

A Path Order is a total order $<_\pi$ on a (finite) set of public terms. We call the minimal element A of $<_\pi$ the initial agent, the maximal element E the final agent, and all other elements M_i intermediate agents.

The construction phase that takes place in each run of the protocol will define the path order for that execution. As a result, this enables a different path to be chosen for each session. We assume that our protocols are designed such that paths are non-repeating (i.e. the path order is always well-defined). A path order need not include all agent identifiers, and so the path can be of any length. The idea of a path order allows us to present an informal notion for the Path Integrity security goal that we will define in the next section.

Definition 2.3. *Path Integrity (Notion)*

A protocol satisfies path integrity if for every session, for all M_i and M_j such that $M_i <_\pi M_j$, if M_j has forwarded the message, then M_i has also forwarded the message.

In order to formalise this notion, we will make use of *expected messages*. We define the linear fact $\mathrm{Build}(p, M, msg)$, which represents the message as it is being constructed: the payload p is used as a unique path identifier, while the second and third terms indicate the current agent being considered as well as the current value of the message (for example, as successive layers of cryptography are applied). For more complex protocols, additional parameters may be added to the ShKey fact to track state. The event fact Add represents that an agent has been added to the path. It is parameterised by the path identifier, the agent who has been added to the path, and how the initiating party anticipates the message will be altered as it passes through them (for example, through de- or re-encryption). The StartBuild event fact is used to mark the beginning of the protocol execution.

These facts are used by three rules during this phase. In the first rule, the initiating agent determines the payload to be sent to the other endpoint.The second rule is repeatedly applied to add new intermediate agents to the path in order, each time replacing the current ShKey fact with a new version containing any required changes to the message. Finally, the last rule sends the message on to the network:

$$\begin{bmatrix} \mathrm{Fr}(p) \\ !\mathrm{Pk}(E, pkE) \end{bmatrix} \xrightarrow{\substack{\mathrm{Add}(p,E,f(p),'') \\ \mathrm{StartBuild}(A,p)}} \begin{bmatrix} \mathrm{Build}(p, E, f(p)) \end{bmatrix}$$

$$\begin{bmatrix} \mathrm{Build}(p, M_i, m) \\ !\mathrm{Pk}(M_j, pkJ) \end{bmatrix} \xrightarrow{\mathrm{Add}(p,M_i,m,g(m))} \begin{bmatrix} \mathrm{Build}(p, M_j, g(m)) \end{bmatrix}$$

$$\begin{bmatrix} \mathrm{Build}(p, M_1, m) \end{bmatrix} \longrightarrow \begin{bmatrix} \mathrm{Net}(m) \end{bmatrix}$$

We use anonymous functions f and g to depict how the message is changed – these are instantiated for each specification based on the protocol in question. For our example protocol, f is pairing with a signature, and g is asymmetric encryption using a public key.

The ordering of Add facts in any given trace establishes a path order $<_\pi$: If the Add fact for M_i is added to the trace before that of M_j, we have that $M_i <_\pi M_j$.

Forwarding Phase.

The forwarding phase of the protocol occurs as intermediate agents transceive the message We model this with the use of the `Unwrap` rule, in which each intermediate agent forwards the message.

The exact nature of this forwarding is dependent on the protocol – it may involve de- or re-encryption, or reading information about how to route the forwarded message.

$$\begin{bmatrix} \mathrm{Net}(m) \\ !\mathrm{Ltk}(M_i, \mathrm{ltk}) \end{bmatrix} \xrightarrow{\mathrm{Forward}(M_i,m,f(m))} \begin{bmatrix} \mathrm{Net}(f(m)) \end{bmatrix}$$

We introduce the Forward fact to denote that the agent has forwarded the message, including the values it has changed from and to. In a faithful execution of the protocol, the parameters of these facts should agree with those in the Add facts created in the Construction phase.

Receive Phase.

The last step of a protocol is upon the successful receipt of the payload by the endpoint.

$$\begin{bmatrix} \text{Net}(f(p)) \\ \text{!Ltk}(E, ltkE)) \end{bmatrix} \xrightarrow{\text{Forward}(E,f(p),'')} \begin{bmatrix} - \end{bmatrix}$$

The `Receive` rule may include validation of the final payload. For example, in the example protocol, the final agent must ensure that the attached signature matches the payload. More advanced validation is discussed in the following section.

Rule Summary

Figure 6.3 shows a set of rewriting rules which model the simple example protocol from Figure 6.2.

Setup

$$\texttt{Gen\_Ltk} := \begin{bmatrix} \text{Fr}(\text{ltk}) \end{bmatrix} \longrightarrow \begin{bmatrix} \text{!Ltk}(A, \text{ltk}) \\ \text{!Pk}(A, \text{pk}(\text{ltk})) \\ \text{Net}(\text{pk}(\text{ltk})) \end{bmatrix}$$

Construction

$$\texttt{Create} := \begin{bmatrix} \text{Fr}(p) \\ \text{!Pk}(E, pkE) \\ \text{!Ltk}(A, ltkA) \end{bmatrix} \xrightarrow{\substack{\text{Add}(p,E,\{p,\text{sign}(p,ltkA)\}_{pkE},'') \\ \text{StartBuild}(A,p)}} \begin{bmatrix} \text{Build}(p, E, \{p, \text{sign}(p, ltkA)\}_{pkE}) \end{bmatrix}$$

$$\texttt{Wrap} := \begin{bmatrix} \text{Build}(p, M_i, m) \\ \text{!Pk}(M_j, pk) \end{bmatrix} \xrightarrow{\text{Add}(p,E,m,\{m\}_{pk})} \begin{bmatrix} \text{Build}(p, M_j, \{m\}_{pk}) \end{bmatrix}$$

$$\texttt{Send} := \begin{bmatrix} \text{Build}(p, M_1, m) \end{bmatrix} \longrightarrow \begin{bmatrix} \text{Net}(m) \end{bmatrix}$$

Forwarding

$$\texttt{Unwrap} := \begin{bmatrix} \text{Net}(\{m\}_{\text{pk}(\text{ltk})}) \\ \text{!Ltk}(M_i, \text{ltk}) \end{bmatrix} \xrightarrow{\text{Forward}(M_i,\{m\}_{\text{pk}(\text{ltk})},m)} \begin{bmatrix} \text{Net}(m) \end{bmatrix}$$

Receive

$$\texttt{Receive} := \begin{bmatrix} \text{Net}(\{p, sig\}_{\text{pk}(ltkE)}) \\ \text{!Ltk}(E, ltkE)) \\ \text{!Pk}(A, pkA) \end{bmatrix} \xrightarrow{\substack{\text{Forward}(E,\{p,sig\}_{\text{pk}(ltkE)},'') \\ \text{Equal}(\text{verify}(sig,p,pkA),\text{true})}} \begin{bmatrix} - \end{bmatrix}$$

FIGURE 6.3: Full set of rewriting rules for the example protocol given in Figure 6.2.

6.3 Security Goals for Path Protocols

We introduce three security goals to cover the range of protocols built in the framework from the previous section. The first, *Path Integrity*, covers the simplest case of message forwarding protocols. We then extend the framework with *Verification-dependent Path Integrity*, in which Path Integrity is made conditional on the receiving party validating the received message.

Finally, for protocols in which a response message is sent along the original path, we define *Path Symmetry*.

Intuition

The intuition behind the structure of our security goals is as follows. Given a specific protocol session, we assume that a path order $<_\pi$ has been defined by a sequence of events. We then examine the case that some agent M_i has successfully forwarded the message. Our goal is satisfied if there is (fundamentally) only one way to fill the gap between these events: that each intermediate agent M_j such that $M_j <_\pi M_i$ has also forwarded the message.

These claims can be verified (or falsified) by considering a partial trace that contains the indicated facts added at specific times. We then reason 'backwards', attempting to reconstruct a sequence of rules from the empty multiset to this state. The security property holds if all trace reconstructions necessarily satisfy the postconditions indicated in the claim.

Path Integrity

We begin by formalising Definition 2.3, the idea of *path integrity*. This goal represents the initiating agent's belief that a sent message will indeed travel through the list of intermediate agents in the intended path in the correct order. Intuitively, this requires a correspondence between the order in which agents were named in applications of the `Wrap` and `Unwrap` rules.

Definition 3.1. *Path Integrity*

We say that a protocol P satisfies Path Integrity if and only if all traces $\tau \in Traces(P)$ satisfy the property displayed in Figure 6.4.

Figure 6.4 includes a breakdown of the intuition matching the security property. Note that we must split the case that an agent has forwarded a message to also consider that the agent may be corrupt. In this case, it is possible that the adversary could have forwarded the message on their behalf. In such a situation, the path order is still preserved, as the message was still forwarded at the correct time.

This definition considers the case where the wrapping functions are applied in the opposite order to the path through the agents (i.e. the first agent added to the path is the last to receive the message). This is in line with many onion-based protocols, but can be trivially amended to consider the other case by changing the ordering of $\#ta_i$ and $\#ta_j$.

Verification-Dependent Path Integrity

The security property defined in the previous section can be seen as *on-the-fly* security: if verified, it ensures that path integrity holds even while a message is in-flight. This is common for onion-style protocols, which use layered encryption such that the message can only be decrypted in a specific order. However, for many protocols, this requirement might be too strict – we may want to loosen it to only consider *completed* sessions.

This approach is typical in TLS-style protocols [NSV+15, LSL+19, LCS+19], where an additional *verification phase* exists to ensure that a session has executed completely. To account for this case, we extend our model by introducing a new set of rules. Our security definition is amended to only apply to protocols in which the verification phase has completed.

Verification Phase.

The verification phase takes place at the end of the protocol execution. We assume that the final message received by the endpoint can be broken into two main parts: one containing the session payload (or some function thereof), and another which includes any additional validation data, such as signatures appended by the intermediate agents.

We modify the `Receive` rule to include an instance of the Check$(E, f(p), m)$ fact, separating these two components. Successive applications of a `Verify` rule then

$\forall A, M_i, M_j, p_{ID}, f_i, t_i, f_j, t_j,$	
$\quad \#ta_i, \#ta_j, \#tk_i, \#ts.$	
$(\nexists \#ta_c.\ \mathrm{Corrupt}(A)@ta_c)$	Suppose A is an honest agent
$\wedge\, \mathrm{StartBuild}(A, p_{ID})@ts$	who starts a session ID p_{ID}
$\wedge\, \mathrm{Add}(p_{ID}, M_i, f_i, t_i)@ta_i$	adding agents M_i...
$\wedge\, \mathrm{Add}(p_{ID}, M_j, f_j, t_j)@ta_j$	...and M_j to the path
$\wedge(\#ts < \#ta_i < \#ta_j)$	such that $M_j <_\pi M_i$
$\wedge(\, \mathrm{Forward}(M_i, f_i, t_i)@tk_i)$	and M_i has successfully forwarded the message
$\implies$	then
$\exists \#tk_j.\ (\#tk_j < \#tk_i)$	at some earlier time
$\wedge($	either:
$\quad (\, \mathrm{Forward}(M_j, f_j, t_j)@tk_j)$	M_j forwarded the message
$\quad \vee (\, \exists\, \#tc_j.\ \mathrm{Corrupt}(M_j)@tc_j \wedge$	or M_j is corrupt, and
$\quad\quad \mathrm{K}(\langle f_j, t_j \rangle)@tk_j)$	the adversary had the necessary knowledge
$\quad)$	to forward the message

FIGURE 6.4: Statement of the Path Integrity security goal, along with corresponding intuition

check the validation data added by each intermediate agent in turn. A final rule runs after all verification steps to confirm successful completion.

$$\begin{bmatrix}\text{Check}(E,f(p),m)\\ !\text{Pk}(M_i,pkM_i))\end{bmatrix}\longrightarrow\begin{bmatrix}\text{Check}(E,g(p),l(m))\end{bmatrix}$$

$$\begin{bmatrix}\text{Check}(E,p,'')\\ !\text{Pk}(A,pkA))\end{bmatrix}\xrightarrow{\text{Complete}(E,f(p))}\begin{bmatrix}\;-\;\end{bmatrix}$$

As in similar rules, anonymous functions f, g, l are used to denote the changing values of the payload and validation portions of the message as it is decomposed between verification steps. These functions are instantiated for individual protocols – e.g., by stripping off a signature from a chain.

The security definition for Verification-Dependent Path Integrity differs only in the addition of an event marking that verification was successful at the end of the protocol's execution. This is achieved with the Complete event fact, which appears only in the final rule of the protocol. As it includes the path identifier it can be readily associated with the corresponding StartBuild fact.

If necessary, additional security requirements could be placed on this verification phase – for example, that agents are verified in the same order that the message was forwarded. This requirement can typically be encoded in the structure of the `Verify` rule (e.g. by including a list of agents by intended order alongside the payload in the packet received by the final party).

Path Symmetry

So far, we have considered only protocols in which a single message is sent between two endpoints. We now consider protocols containing a response message. For such protocols, one desired goal may be *Path Symmetry*. Intuitively, this is the notion that the reply should follow the same path as the original message. This is particularly relevant for payment network systems, in which an intermediate agent (who aids in forwarding a payment) is often only able to claim their transaction fee by reading a component of the response message. While Path Integrity focuses on ensuring that a protocol proceeds according to the initiating agent's plan, Path Symmetry is a requirement on the execution order for the reply in relation to the original message.

Our model can be readily extended to capture Path Symmetry with the following approach.

- The `Receive` rule is replaced with a new `Reply` rule. If necessary, this rule can be broken down into several wrapping rules, similar to the Construction phase.
- A new set of rules are defined to represent message forwarding in a **Response** phase. These rules make use of Backward facts to represent the expected and actual messages sent (in the same vein as Forward facts).

- We assume that honest agents are aware of the "direction" of a message, and will only assist in the **Response** phase if they have already forwarded the original message. This is a consequence of the structure of protocols for which Path Symmetry is an important goal – in cases where there is no relation between the original message and its response, the protocol can instead be divided into two separate executions. For example, a response message might necessarily contain a hash or signature of the original request. This is modelled by adding a new linear fact to the `Forward` rule to model the internal state the forwarding agent. This rule is used to enforce a functional relationship between the original message and the reply.

With these modifications in place, we can express path symmetry. Intuitively, this requires that there be a relationship between the order messages are sent in the **Forwarding** and **Response** phases. Unlike Path Integrity, our definition considers only pairs of adjacent agents; universal quantification ensures that it indeed applies to the total path order.

Definition 3.2. *Path Symmetry*

We say that a protocol P satisfies Path Symmetry if and only if for all traces $\tau \in Traces(P)$, the following property holds:

$$
\begin{array}{l}
\forall M_i, M_j, f_i, t_j, t_i, x, y, \\
\qquad\qquad \#tf_i, \#tf_j, \#tr_i, . \\
(\nexists \#tc_j.\ \mathrm{Corrupt}(M_j)@tc_j) \\
\ \wedge \mathrm{Forward}(M_j, x, t_j)@tf_j \\
\ \wedge (\\
\qquad (\quad \mathrm{Forward}(M_i, f_i, x)@tf_i \\
\qquad\ \ \wedge \mathrm{Backward}(M_i, y, t_i)@tr_i \\
\qquad) \vee (\\
\qquad\ \ \exists \#tc_i. \\
\qquad\qquad\quad \mathrm{Corrupt}(M_i)@tc_i \\
\qquad\qquad \wedge \mathrm{K}(\langle f_i, x\rangle)@tf_i \\
\qquad\qquad \wedge \mathrm{K}(\langle y, t_i\rangle)@tr_i \\
\quad)) \\
\ \wedge (\#tf_i < \#tf_j < \#tr_i) \\
\quad \Longrightarrow \\
\exists f_j, \#tr_j.\ (\#tf_j < \#tr_j < \#tr_i) \\
\quad \wedge \mathrm{Backward}(M_j, f_j, y)@tr_j
\end{array}
$$

The security claim can be seen as being made from the perspective of an intermediate agent (here, M_j), who wishes to ensure that if they assist in forwarding an incoming message from M_i, then M_i cannot later forward the return message without M_j's continued participation (i.e., a $\mathrm{Backward}(M_j, \ldots)$ event). As such, security claims do not directly refer to a path identifier, as an intermediate agent may not be aware of

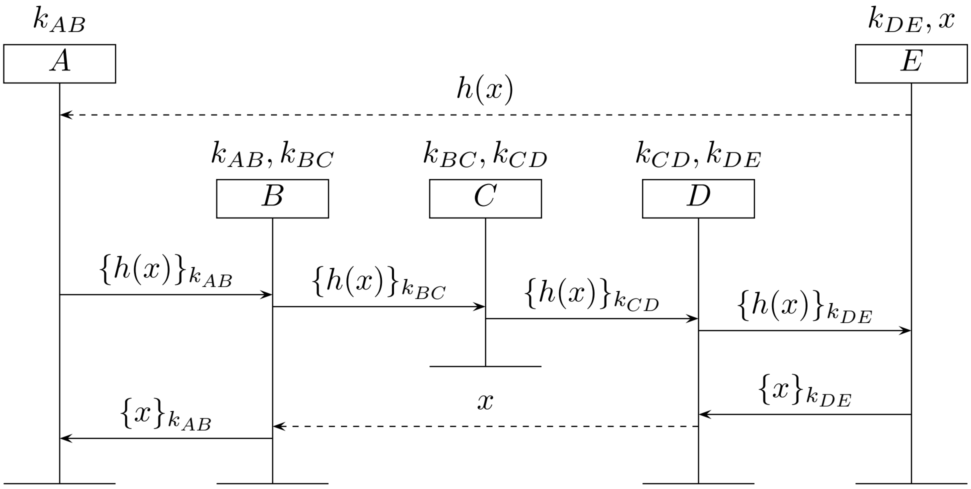


FIGURE 6.5: Simplified attack on the Lightning payment forwarding protocol. Dashed lines indicate out-of-band communication

this value. As with Path Integrity, we split M_i's actions into two cases. Either M_i is honest, and performs the message forwarding actions, as expected, or they are corrupt, and the adversary has the necessary knowledge to perform the message forwarding on their behalf.

6.4 Case Study: Lightning Network

We now look at the Lightning Network, a second-layer protocol designed to run on top of the Bitcoin cryptocurrency. The aim of the Lightning Network is to increase network speeds by resolving many transactions "off-chain". At its core, this is done by pairs of agents maintaining an unpublished contract, which describes how funds are to be allocated between them. Instead of performing transactions on the Bitcoin ledger, these agents can instead opt to update their shared contract to redistribute the funds appropriately. In case of payment disputes, the contract can be published to the blockchain, which contains resolution mechanisms.

First, we give an overview of the payment forwarding protocol. We then demonstrate a fee-skimming attack, originally discovered by Malavolta et al. [MMSS+19]. Next we extend their work, by demonstrating that it is possible to build a secure two-round payment protocol using only simple cryptographic primitives.

Lightning Payment Protocol.

The Lightning Network consists of a collection of nodes which communicate using a gossip protocol. Any pair of nodes can open a *payment channel* through a channel establishment protocol, and may choose to broadcast this information using the gossip network. As a result, we can consider a graph of all active participants of the network, where two nodes are connected by an edge if and only if there exists an open public payment channel between them. The gossip protocol aims to give agents a complete view of this graph, allowing them to route payments along these edges.

We assume that all agents have perfect information about the structure of the payment network, that all open channels are accessible to all honest participants, and that agents can freely publish to and read from the underlying blockchain whenever desired. This follows from the security requirements of the gossip protocol and blockchain (analysis of whether these goals are indeed achieved is considered out of scope of this work).

The Lightning payment protocol makes use of a cryptographic scheme named *Hash Time-Locked Contracts* (HTLC). Fundamentally, HTLCs form a promise that A will pay B some amount of money if B is able to produce the preimage of a hashed value $h(x)$ within a certain time limit. Most commonly, the value x is chosen by B, who then supplies A with the hash as a payment invoice through some off-band communication. A can then be sure that they are paying the correct person, as only B can redeem the associated HTLC.

This scheme is central to the design of the *chained payment protocol*. Suppose A wishes to send funds to some agent D with whom they do not currently have an active payment channel. The protocol proceeds as follows:

1. D chooses a value x and sends $h(x)$ to A
2. A chooses some path on the network to D – say, via B and C
3. A opens a HTLC with B, agreeing to pay B if B can produce x from $h(x)$, and including forwarding data to the remaining agents
4. B creates a new HTLC with C with the same parameters, who then in turn creates a HTLC with D
5. D is able to reproduce x (as they are the one who chose it), and provides evidence of this to C, closing the HTLC and claiming the funds
6. C now also knows x, and so can claim the funds from B, who can in turn claim them from A

The ability of agents to "share" their payment channels to other members of the system is a key design feature of the lightning network. Generally, this is assumed to come with *fees* for using the channel - in particular, if A wishes to send D an amount of money m, they will instead initiate the protocol with some funds $m + t$, and each intermediate agent claims some small share of the transmitted funds as a transaction fee. Note that funds are committed during the first round of the protocol (the 'locking' phase), but are not distributed until the second round (the 'release' phase).

As such, the initiating party is concerned with ensuring that the payment follows their chosen path, in order to minimize fees. Moreover, honest forwarding agents also benefit from this: if they are skipped over, they are at risk of being denied their earned transaction fees.

The Lightning Network chained payment protocol also exhibits a skipping attack. It is detailed by Malavolta et al. [MMSS+19], who name it the "wormhole" attack.

In this attack, a corrupt agent M_i sends a non-adjacent agent M_j the hash preimage x through an off-band channel on the return journey. Although M_i loses funds (the HTLC against them is redeemed but they do not turn in x to the next agent in the chain), M_j gains the equivalent amount, since noone can redeem the HTLC against M_j them. Further, M_j also gains the transaction fees of every agent in between the two corrupt agents. The net result is that M_i and M_j skim the transaction fees of the other agents on the path. These agents may not be aware the attack has even taken place, instead assuming that the payment failed, allowing the associated HTLCs to time out.

Figure 6.5 contains a simplified diagram of the skipping attack on the lightning payment protocol. Note that we abstract the instantiation and later resolution of HTLCs into individual messages – the actual process requires several messages in which agents update their shared contracts and issue revocation keys for the previous versions.

'Wormhole'-Free Payment Protocols Do Exist.

In addition to demonstrating this attack and designing a new cryptographic primitive (named an Anonymous Multi-Hop Lock), which is used to amend the protocol, Malavolta et al. prove 'wormholes' cannot be avoided. More specifically:

Theorem 4.1. *(Malavolta et al. [MMSS+19, Theorem 1])*

For all two-round (without broadcast channels) multi-hop payment protocols there exists a path prone to the wormhole attack.

Here, "round" refers to interactions between adjacent agents. Thus, a round-trip is considered to be two "rounds": one for the forward trip and one for the return trip.

The model used in this paper differs in several aspects from that of Malavolta et al. (the full model, along with a more formal expression of this result, labelled Theorem 5, is in their paper [MMSS+19]).

The most significant difference relates to agents' knowledge of the path – in their model the only starting knowledge is the hashed value $h(x)$ shared between the two endpoints, and encryption keys shared between adjacent agents on the network. In our model, we additionally assume that the network topology is known by the initiating agent. Indeed, this is an intended consequence of the gossip protocol which underlies the Lightning Network.

Under these assumptions, we present the following results:

Lemma 4.2. *Sufficiency*

Path Integrity and Path Symmetry are sufficient to ensure that no wormhole attack exists on a round-trip payment path protocol.

Theorem 4.3. *Possibility*

There exists a payment protocol with one round trip which does not admit a wormhole attack.

We argue[1] the validity of Lemma 4.2. The security of a payment protocol is conditional on the following:

- The initiating agent must be able to choose a valid payment path in accordance with the network topology
- The initiating agent must be assured that the payment proceeds along their chosen path
- Intermediate agents must be assured that if they forward a payment request (by opening the associated contracts), they will be compensated if the payment is successful (by being able to close the contract).

The first point follows from our assumptions about the gossip protocol. The second is an immediate consequence of Path Integrity. Finally, Path Symmetry ensures the last point.

Our framework allows us to identify an attack on Path Symmetry for the Lightning Network. The identified attack is indeed the fee-skimming attack discussed above. Malavolta et al. present a wormhole-free protocol using an additional round of communication. They prove the security of their protocol using a game-hopping proof in the computational model.

We provide a payment network protocol, named Lightning-Sig. This protocol achieves Path Integrity and Path Symmetry, and thus is not susceptible to a skipping attack. The design capitalises on the underlying Sphinx [DG09] protocol, which includes per-hop payloads used to transmit anonymous routing data to each agent. In particular, we propose extending these per-hop payloads with ephemeral (one-time) keys used to create a chain of signatures that can be verified by each agent on the return journey.

Lightning-Sig is presented for exposition rather than as a suggested implementation – it does not adapt directly to the underlying blockchain structure, and would impose additional computational and storage costs.

6.5 A Path-Integral Payment Network Protocol

We introduce a structure for a Payment Network protocol which satisfies Path Symmetry. As with the model used in the rest of the paper, we assume that the initiating agent A is aware of the full network topology, and chooses the path accordingly. A diagram showing the intended execution of the protocol is given in Figure 6.6, for the case of 4 agents. The transition rules used for this protocol are given in Figure 6.7. It differs from the Lightning payment forwarding protocol as follows:

- At the start of the protocol, A generates an ephemeral asymmetric keypair $(e_i, \mathrm{pk}(e_i))$ to be used by each intermediate agent M_i as well as the endpoint E.

[1] A full formal proof necessitates a definition of payment, which is beyond the scope of the current paper.

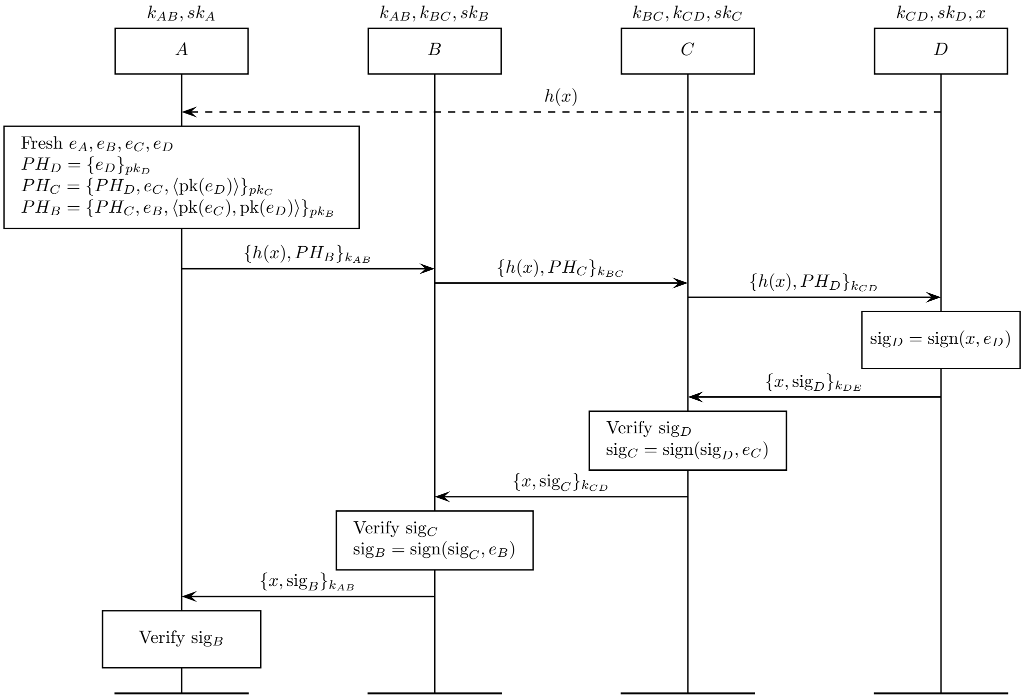

k_{AB}, sk_A
k_{AB}, k_{BC}, sk_B
k_{BC}, k_{CD}, sk_C
k_{CD}, sk_D, x
A
B
C
D
$h(x)$
Fresh e_A, e_B, e_C, e_D
$PH_D = \{e_D\}_{pk_D}$
$PH_C = \{PH_D, e_C, \langle \mathrm{pk}(e_D)\rangle\}_{pk_C}$
$PH_B = \{PH_C, e_B, \langle \mathrm{pk}(e_C), \mathrm{pk}(e_D)\rangle\}_{pk_B}$
$\{h(x), PH_B\}_{k_{AB}}$
$\{h(x), PH_C\}_{k_{BC}}$
$\{h(x), PH_D\}_{k_{CD}}$
$\mathrm{sig}_D = \mathrm{sign}(x, e_D)$
$\{x, \mathrm{sig}_D\}_{k_{DE}}$
Verify sig_D
$\mathrm{sig}_C = \mathrm{sign}(\mathrm{sig}_D, e_C)$
$\{x, \mathrm{sig}_C\}_{k_{CD}}$
Verify sig_C
$\mathrm{sig}_B = \mathrm{sign}(\mathrm{sig}_C, e_B)$
$\{x, \mathrm{sig}_B\}_{k_{AB}}$
Verify sig_B

FIGURE 6.6: MSC for Path-Symmetric payment network protocol

- The structure of the underlying Sphinx protocol allows A to send secure packets to each intermediate agent. To each M_i, A sends the private key generated for them, as well as an ordered list of the public keys *after* them on the path. For example, M_i stores the public keys for M_{i+1}, M_{i+2}, etc.

- Each agent M_i appends a signature using their ephemeral private key signature to the hash preimage x on the return journey. The HTLC contract is amended such that M_i will only disperse payment to M_{i+1} if they receive the hash preimage x along with a chained signature with each of the keys for $M_{i+1}, M_{i+2} \ldots$ in turn.

Explicitly, in the forward direction, each message consists of two components: the hash $h(x)$, and a per-hop payload (PH_i), which is defined inductively (and reverse to the path order) as follows:

$$\begin{aligned} PH_n &:= \{e_n\}_{\mathrm{pk}_{M_n}} \\ PH_i &:= \{\ PH_{i+1}, \\ &\qquad\quad \langle \mathrm{pk}(e_{i+1}), \mathrm{pk}(e_{i+2}), \ldots \rangle, \\ &\qquad\quad e_i \\ &\quad\ \}_{\mathrm{pk}_{M_i}} \end{aligned}$$

The first component is the per-hop payload for the agent after them in the path, the second is the set of ephemeral public keys after them in the path, and the last is their ephemeral private key.

The addition of (ordered) signatures to the hash preimage ensures that the return trip must follow the same path as in the forward direction. Intuitively, the forward

Setup

$$\texttt{Gen\_Ltk} := \begin{bmatrix}\mathrm{Fr}(ltk)\end{bmatrix} \longrightarrow \begin{bmatrix}!\mathrm{Ltk}(A, ltk)\\ !\mathrm{Pk}(A, pk(ltk))\\ \mathrm{Net}(pk(ltk))\end{bmatrix}$$

$$\texttt{Gen\_ShKey} := \begin{bmatrix}\mathrm{Fr}(k)\end{bmatrix} \longrightarrow \begin{bmatrix}!\mathrm{ShKey}(A, B, k)\end{bmatrix}$$

Construction

$$\texttt{Create} := \begin{bmatrix}\mathrm{Fr}(p), \mathrm{Fr}(e_E)\\ !\mathrm{ShKey}(D, E, k_{DE})\\ !\mathrm{Pk}(E, pk_E)\end{bmatrix} \xrightarrow{\substack{\mathrm{Add}(p,E,\{h(p),PH_E\}_{k_{DE}},'')\\ \mathrm{StartBuild}(A,p)}} \begin{bmatrix}\mathrm{Build}(p, A, E, \{h(p), \{e_E\}_{pk_E}\}_{k_{DE}})\end{bmatrix}$$

$$\texttt{Wrap} := \begin{bmatrix}\mathrm{Build}(p, A, N, \{m, PH_X\}_{k_{NX}})\\ !\mathrm{Pk}(N, pk_N), \mathrm{Fr}(e_N)\\ !\mathrm{ShKey}(M, N, k_{MN})\end{bmatrix} \xrightarrow{\mathrm{Add}(p,N,m,m)} \begin{bmatrix}\mathrm{Build}(p, A, M, \{m, PH_N\}_{k_{MN}})\end{bmatrix}$$

$$\texttt{Send} := \begin{bmatrix}\mathrm{Build}(p, A, M, \{m\}_k)\\ !\mathrm{ShKey}(A, M, k)\end{bmatrix} \longrightarrow \begin{bmatrix}\mathrm{Net}(\{m\}_k)\\ \mathrm{InitState}(A, p, pkeys)\end{bmatrix}$$

Forwarding

$$\texttt{Unwrap} := \begin{bmatrix}\mathrm{Net}(\{h(x), PH_C\}_{k_{BC}})\\ !\mathrm{Ltk}(C, ltk))\\ !\mathrm{ShKey}(B, C, k_{BC})\\ !\mathrm{ShKey}(C, D, k_{CD})\end{bmatrix} \xrightarrow{\mathrm{Forward}(C,m,m)} \begin{bmatrix}\mathrm{Net}(\{h(x), PH_D\}_{k_{CD}})\\ \mathrm{FwdState}(C, h(x), e_C, \mathrm{pk}(keys))\end{bmatrix}$$

Reply

$$\texttt{Reply} := \begin{bmatrix}\mathrm{Net}(\{h(p), PH_E\}_{k_{DE}})\\ !\mathrm{Ltk}(E, ltkE, pkE))\\ !\mathrm{ShKey}(D, E, k_{DE})\end{bmatrix} \xrightarrow{\mathrm{Forward}(E,h(p),'')\ \mathrm{Backward}(E,'',p)} \begin{bmatrix}\mathrm{Net}(\{p, \mathrm{sign}(p, e_E)\}_{k_{DE}})\end{bmatrix}$$

Response

$$\texttt{Backward} := \begin{bmatrix}\mathrm{Net}(\{p, \mathrm{sign}(p, keys)\}_{k_{CD}})\\ !\mathrm{ShKey}(B, C, k_{BC})\\ !\mathrm{ShKey}(C, D, k_{CD})\\ \mathrm{FwdState}(C, h(p), e_C, pkeys)\end{bmatrix} \xrightarrow{\substack{\mathrm{Backward}(C,p,p)\\ \mathrm{Equal}(\mathrm{verify}(sig,p,pkeys),\mathrm{true})}} \begin{bmatrix}\mathrm{Net}(\{p, \mathrm{sign}(p, \langle keys, e_C\rangle\}_{k_{BC}})\end{bmatrix}$$

$$\texttt{Finish} := \begin{bmatrix}\mathrm{Net}(\{p, sig\}_{pk})\\ !\mathrm{ShKey}(A, B, k_{AB}))\\ \mathrm{InitState}(A, p, pkeys)\end{bmatrix} \xrightarrow{\mathrm{Equal}(\mathrm{verify}(sig,p,pkeys),\mathrm{true})} \begin{bmatrix}\quad - \quad\end{bmatrix}$$

FIGURE 6.7: Rewriting rules for a payment network protocol that satisfies path symmetry. Some terms are compressed for space.

journey ensures that only M_i is able to generate the signature from e_i, and so any agent after M_i on the return journey can be assured that M_i did indeed forward the message.

In addition, this does not significantly affect the privacy of the protocol – the generated keys are not tied to the identities of the forwarding agents. If necessary, the initiating agent can also "pad" the message with additional keys for the endpoint to use, such that agents can not be fully aware of where in the path order they are.

For modelling reasons, we represent signatures with multiple keys by $\mathrm{sign}(x, \langle k_1, k_2, ...\rangle)$, allowing agents to insert an additional key into the list used in the signature. We also assume that the body of a signature is included alongside the signature itself in order to save space.

We emphasise two key contributions from our approach over the prior work of Malavolta et al. :

- We prove that *no additional round trips* are needed to avoid wormhole attacks on payment networks
- We demonstrate that *no additional cryptographic primitives* are needed to achieve this security

Our models for the Lightning Network are built from the Lightning "Bolt" specification [Tea20], which is a continually-updated document. We model communication between pairs of agents using symmetric encryption, with per-hop payloads encrypted using nested asymmetric encryption. The endpoint is assumed to be able to retrieve the session secret from the hashed value.

6.6 Tamarin Implementations

We implemented our approach from Section 6.3 to perform a security survey of several path protocols of note from the literature. We created abstract models of these protocols into our framework, creating adapted implementations inside the Tamarin prover tool [MSCB13].

We split our analysis into three main families: **Middlebox-Enabled TLS**, **Mixnets** and **Payment Networks**. We consider onion-style protocols (such as those demonstrated by Chaum [Cha81]) as part of the Mixnet family – although there can be some differences on the network layer (such as by batching messages), the message structure is often very similar. Our reasons for choosing these specific protocols are as follows:

- **Middlebox-Enabled TLS**. mcTLS [NSV+15] uses session keys shared between multiple parties based on their permissions (read, write), rather than their location in the path. mbTLS [NLG+17] uses unique session keys for each pair of adjacent agents. maTLS [LSL+19] furthers this scheme with chained signatures (forming a *modification log*).
- **Mixnets**. The original models by Chaum [Cha81] use chained asymmetric encryption. The TOR [SDM04] ecosystem establishes connections using symmetric encryption (with public keys only for key-establishment) - paths are defined by a connection ID. HORNET [CAB+15] reduces the use of state compared to TOR, by including routing data as part of the message in place of a connection ID, using a construction based on Sphinx [DG09].

  Mixnets generally offer several options for the return path of a message - they can be symmetrical, or chosen by the initiating agent or recipient. For Path Symmetry we consider the former case (the latter is treated as two separate executions).
- **Payment Networks**. The Lightning Network [PD16] uses per-hop payloads to conceal routing data, with a term $h(x)$ (and later x) that is shared between agents. Our exhibited Lightning-Sig protocol extends the return journey with a chain of signatures created using per-hop data transmitted during the forward journey.

Protocol Name	Integrity	Symmetry
Middlebox-Enabled TLS		
- mcTLS [NSV+15]	×	×
- mbTLS [NLG+17]	×	×
- maTLS [LSL+19]	✓*	✓*
Mixnet		
- Chaum [Cha81]	✓	✓
- TOR – Establishment [SDM04]	✓	✓
- TOR – Data Exchange [SDM04]	✓	✓
- HORNET [CAB+15]	✓	✓
Payment Network		
- Lightning [PD16]	✓	×
- Lightning-Sig	✓	✓

TABLE 6.1: Results of case studies on message-forwarding protocols

The results of our analysis can be found in Table 6.1. We give results for Path Integrity (as per Definition 3.1) as well as Path Symmetry (as per Definition 3.2). For each protocol we indicate if the security goal is met (✓) or violated (×). An asterisk * indicates the goal is dependent on a Verification phase (as in Verification-Dependent Path Integrity).

We note that for different protocols the most desirable (and indeed the intended) security goals may differ. For example, Path Symmetry is an important goal for Payment Networks, but may not be as relevant for Onion-style protocols.

Tamarin Implementation Notes.

We embedded the protocols considered in our case studies into the Tamarin prover tool. We note several implementation details arising from this translation. Some decisions were often made in order to reduce the complexity of analysis within the automated tool, in order to ensure termination and improve execution time. This comes at a tradeoff of the faithfulness of the model to the intended specification: unfortunately, some loss of detail is an inevitable consequence of symbolic modelling and the use of verification tools.

Analysis was performed on a 28-core Xeon server with a 2.6Ghz CPU and 64gb of RAM. In some cases, additional RAM was required (up to 1TB). An Oracle file was used to help guide Tamarin's built-in heuristics.

Firstly, we add some restrictions to follow the intuition laid out in earlier sections. This includes preventing duplicate agents on a path (i.e. enforcing well-formedness of the path order).

Secondly, we make use of binding of message types (for example, assuming agents can deduce that a term is fresh or whether it is headed in the forwards- or backwards-directions). This reduces the adversary's capability to introduce confusion-style attacks, which can significantly complicate analysis.

Finally, we enforce some bounds on the complexity of each execution – in particular, the number of agents and number of parallel sessions. In most cases this is set to 5 agents (two endpoints and three intermediate agents) running a single session. Although the Tamarin prover includes tools for induction arguments, it still struggles to handle protocols with multiple loops. As such, restricting to the finite case is unfortunately a necessity. We present the following conjecture:

Conjecture 6.1. *Sufficiency of Finite Analysis*

There exists $n \in \mathbb{Z}$ such that for all path protocols, if there is a trace containing a path of length $m > n$ which violates Path Integrity, then there is also a trace containing a path of length n which violates Path Integrity.

The intuition for this conjecture is as follows:

- We can view the path as a sequence of agents, and whether or not each is corrupted
- The structure of the message does not fundamentally change between intermediate agents: hence two adjacent honest agents combined exhibit similar behaviour to a single honest agent (and likewise for corrupt agents)
- An attack requires at most two corrupt agents, with an honest agent in between to be skipped

Following this reasoning, the most significant path for analysis is one in which a single honest intermediate agent is sandwiched in between two corrupt parties. In theory, other paths which admit attacks can be "reduced" into one of this shape.

Chapter 7

TLS and Middleboxes

Middleboxes are widely used for various in-network functionalities, becoming indispensable in the modern internet ecosystem. They can be deployed for various benefits in terms of performance (e.g., proxies, DNS interception boxes, transcoders), security (e.g., firewalls, anti-virus software), or content filtering (e.g., parental controls). However, the practice of using middleboxes is not immediately compatible with Transport Layer Security (TLS) — the de-facto standard for securing end-to-end connections. Since TLS is designed to provide end-to-end authentication and confidential communication, middleboxes are not supposed to read or modify any TLS traffic.

A well-known method for overcoming this limitation is SplitTLS, in which a TLS session between two endpoints is split into two separate segments so that a middlebox can decrypt, encrypt, and forward the traffic as a man-in-the-middle. While SplitTLS allows us to use middleboxes with TLS, it poses security and privacy risks on both the client and server sides.

In this chapter, several concerns with the current middlebox-enabled TLS ecosystem are presented. A new solution - named "middlebox-aware TLS" is presented, which aims to address these concerns. Its feasibility is demonstrated through experimental analysis, and an analysis is performed against both traditional end-to-end authentication goals and some protocol-specific requirements.

The work in this chapter is based on the NDSS 2019 paper "maTLS: How to Make TLS middlebox-aware" [LSL+19].
My main contribution to this topic was in the formalisation of the security goals. In addition, I aided in identifying several potential attacks in the protocol during the design process.
A case study is added (originally written for the paper in the previous chapter), which demonstrates the necessity of the security features included in the design.

7.1 The TLS Protocol Suite

HTTPS – HTTP over TLS [Res00] is becoming increasingly common, with more than 50% of HTTP traffic being encrypted using TLS [FBK+17, NFL+14]). Middleboxes enhance HTTP traffic through a variety of methods, but often require access to the unencrypted message body. As such, there is a need to allow middleboxes to participate in a TLS session. However, the current de-facto approach remains the usage of Split TLS: breaking the overall TLS session into a series of completely disjoint segments.

As a result of this, users are often required to install custom root certificates, which allows a middlebox to impersonate any server in order to read and modify all HTTPS traffic. For servers, websites are often required to share their private keys with some middlebox service providers (e.g., content delivery networks (CDNs)), so that middleboxes can provide their content to clients with better performance. These imply that *a compromised middlebox may be used to perform critical attacks*, either by abusing custom root certificates to impersonate someone else or by using a shared private key to impersonate a particular server.

Such vulnerabilities of middleboxes have been reported in several studies [dCdCM16, DMS+17, WMY18, ORSZ16, TII+18]; for instance, some middleboxes accept *nearly all certificates* in spite of certificate validation failures, which gives a chance for another compromised or malicious middlebox to meddle in the TLS session [dCdCM16, DMS+17, WMY18]. Similarly, a middlebox that splits a TLS session may support only weak ciphersuites, which are vulnerable to known attacks such as the Logjam attack [ABD+15] or the FREAK attack [BBDL+15]. Even worse, it has been reported that middleboxes are being used to inject malicious code [TII+18, ORSZ16, CCM16]; for example, Giorgos et al. [TII+18] found that 5.15% of proxies inject malicious or unwanted content into web pages.

Nevertheless, as middleboxes provide crucial benefits to users, content providers, and network operators, there has been a long thread of studies aiming to accommodate for middleboxes in secure networking between two endpoints [SLPR15, LSP+16, PLPR18, HKHH17, LMS+12, Nir12, NSV+15]. These studies can be largely classified into three main categories: encryption-based, trusted execution environment (TEE)-based, and TLS extension-based. First, BlindBox [SLPR15] and Embark [LSP+16] propose the usage of special encryption schemes such as order-preserving encryption to allow middleboxes to perform their functionality over encrypted packets. Second, SafeBricks [PLPR18] and SGX-Box [HKHH17] leverage TEEs such as Intel SGX to make middleboxes trustworthy. Finally, several studies seek to extend the TLS protocol [NSV+15, LMS+12, Nir12, MWNG12, NLG+17] in order to let middleboxes intervene during the TLS handshake and perform their functionalities within the session.

However, these approaches pose several technical challenges and limitations. The encryption-based approaches depend greatly on their encryption mechanisms; as a result, their functionalities are limited to pattern-matching or range-filtering. The proposals leveraging TEEs are only applicable to the middleboxes with specific hardware that provides secure enclaves. What is worse, neither of them are backward-compatible (i.e., current middleboxes have to be replaced to adopt such approaches). The TLS extension approaches are most feasible in the sense that TLS software can be extended to support the backward compatibility. However, these approaches leave three issues that have not been comprehensively solved.

First, the proposal of using explicit proxies in IETF [LMS+12] introduces a proxy certificate to indicate that the certificate holder is a middlebox. However, the client can only authenticate the next middlebox, not the server or other middleboxes intervening in the session. Thus, there is still a risk of an unknown middlebox meddling in the session. Second, mcTLS [NSV+15], TLMSP[1], and TLS Keyshare extension[2] [Nir12] use the same symmetric key (and hence the same ciphersuite) across all the split TLS segments between the two endpoints. As a result, middleboxes that do not support the specific ciphersuite chosen will not be able to process the TLS traffic. Furthermore, the middleboxes share the same keystream, which may undermine security [MWNG12]. Third, none of these proposals except TLMSP allow the client to know who has sent TLS traffic as well as who has modified it. For example, in mcTLS [NSV+15], the client cannot check whether the TLS traffic he received originated from a valid endpoint (e.g., a cache or an endpoint) if there is a middlebox that modified the message during transit.

Contributions.
In this chapter, we propose an extension to TLS, which ensures middleboxes are *visible* and *auditable*. The starting point is to enable a client to authenticate all the middleboxes. We first define *middlebox certificates*, which are signed by certificate authorities (CAs), and used to encrypt the channel for each TLS segment (e.g., between a client and a middlebox, between middleboxes, and between a middlebox and a server). The use of middlebox certificates eliminates the insecure practice of users installing custom root certificates or servers sharing their private keys with third parties (like CDNs). We also introduce them with middlebox transparency log servers to make middleboxes auditable. Along with auditable middleboxes, we design the middlebox-aware TLS (maTLS) protocol, a TLS extension auditing the security behaviors of middleboxes. The maTLS protocol is designed to satisfy the following security goals (to be detailed later): server authentication, middlebox authentication, segment secrecy, individual secrecy, data source authentication, modification accountability, and path integrity.

To satisfy these goals, a client authenticates all participants of its maTLS session. That is, the client verifies the certificates of all the participating middleboxes to prevent any arbitrary middleboxes from intervening in the session, which we will refer to as *explicit authentication*. Moreover, the two endpoints confirm the negotiated security association of every segment to ensure its confidentiality and integrity, which is called

[1] Transport Layer Middlebox Security Protocol (`https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=52930`). The protocol is being discussed in ETSI, and the draft of the protocol specification is currently unavailable. We refer to the document in a web archive: https://docplayer.net/88122390-Announcement-of-middlebox-security-protocol-msp-draft-parts.html

[2] Note that this is different from the `keyshare` extension used to negotiate a Diffie-Hellman shared key in TLS 1.3.

security parameter verification. Note that a security association consists of a TLS version, a ciphersuite, and a confirmation of encryption key establishment. Lastly, maTLS performs *valid modification checks*, which allows the endpoints of a maTLS session to verify whether the received messages have been modified only by authorized middleboxes. This way, maTLS provides auditability of all participants in the session.

We also evaluate the security and performance of maTLS. We define a set of security goals that middlebox-enabled TLS protocols should aim to achieve. Finally, we evaluate the feasibility of the maTLS protocol by building an implementation using OpenSSL and comparing its performance against prior proposals.

Organisation.
In Section 7.2 we discuss the Split TLS approach for enabling middleboxes, and how it undermines the security goals of TLS. We also discuss some of the other approaches for overcoming the shortcomings of Split TLS. Section 7.3 contains a case study of the mbTLS protocol, another proposed approach for enhancing TLS. In Section 7.4 we discuss the requirements (both in terms of security and implementation) of a middlebox-enabled TLS extension, then present the maTLS extension. Section 7.5 contains a discussion of the formal security goals that should be achieved by such a protocol. Finally, in Section 7.6, we evaluate an implementation of the maTLS protocol, and conclude, discussing the hurdles in deployment of such a scheme.

7.2 Middlebox-Enabled TLS Schemes

The TLS protocol [Die08, E. 18], coupled with a Public Key Infrastructure (PKI), is designed to authenticate endpoints, establishing a secure communication channel between them. The security goals of TLS are authentication, confidentiality, and integrity: *authentication* is confirmation of the identity of the other party, by validating a certificate chain and verifying a proof-of-possession of the corresponding private key. In practice, the server is always authenticated from its certificate, while authenticating the client is optional. *Confidentiality* is a guarantee that the data sent over the channel is secret to all but the endpoints. *Integrity* ensures that any third parties do not modify data on the network.

These security goals are achieved by two components of the TLS protocol suite, called the handshake and record protocols. The main purpose of the TLS handshake protocol is to establish a master secret, which will be used for an authenticated encryption and decryption of the data between two endpoints.

X.509 Certificates.
A digital certificate is an attestation that binds a subject (e.g., a domain name) to its public key. This binding is guaranteed by a Certificate Authority (CA) with its signature in the certificate. The certificate authority also possesses its own certificate, issued by another CA. This results in a chain of certificates terminated with a self-signed certificate called a root certificate. A certificate receiver validates the certificate if the receiver trusts the root certificate in the chain and all the signatures in the

certificates can be verified using the public key of the next certificate in the chain (up to the root certificate).

CAs also indicate that a domain owner satisfies specific suggested requirements. For example, a domain validation (DV) certificate is issued when a domain owner has successfully proved its ownership of the domain. To provide stronger assurance to clients that a certificate has been adequately issued, CAs can require domain owners to follow a set of stricter criteria in order to obtain extended validation (EV) certificates.

On the Internet, X.509 [ITU00] is the most widely used format for certificates, which typically include fields such as the subject, its public key, a serial number, and the certificate's validity period. The current version of X.509, version 3, supports extensions that CAs can add for a variety of purposes; for example, the Server Alternative Name (SAN) field [CSF+08] is used to allow alternative names of the certificate holder.

Certificate Transparency.
The PKI trust model has a severe drawback in reality: any CA can issue a certificate for *any* domain, potentially exposing users to high risk. There have been security incidents in which commercial CAs were compromised and issued fraudulent certificates, allowing attackers to impersonate the actual certificate owner or perform man-in-the-middle attacks [Com11, Wil15].

To mitigate the risks from CA compromises, Google introduced the Certificate Transparency (CT) system [LLK13], which aims to provide *accountability* to a PKI. This is achieved by archiving every certificate into multiple append-only public log servers so that any entity can monitor and audit a CAs' operations. Upon submission of a certificate chain, the log servers return a signed proof called a signed certificate timestamp (SCT), which can be verified using the public keys of the log servers. An SCT can be delivered from web servers to the browsers separately or embedded in the web server's certificate, via a TLS extension or through OCSP. For example, a browser might display a lower security indicator if the server's certificate is not logged on the CT servers. CT logging became mandatory in Chrome for all certificates issued after April 2018 [O'B18]. A third party (e.g., a CA) can keep track of CT log servers to see if there is any mis-issuance of certificates, thus providing *auditability* of certificates and *accountability* of CAs' certificate issuance. For example, TLSMate's CertSpotter [SSL] and Facebook's CT Monitor [Fac] monitor each log server and alert a domain owner if a new certificate that binds to her domain name has been issued.

Middleboxes in SplitTLS

We will focus our attention on middleboxes which inspect application data sent over HTTPS, for the purpose of security or performance. Figure 7.1 illustrates how they typically intervene in a TLS session. A middlebox intercepts the TLS session, splitting it into two segments. The middlebox then pretends to be the client while communicating with the server and in turn impersonates the server in its communication with the client. In the case of multiple middleboxes, they form a chain of TLS segments

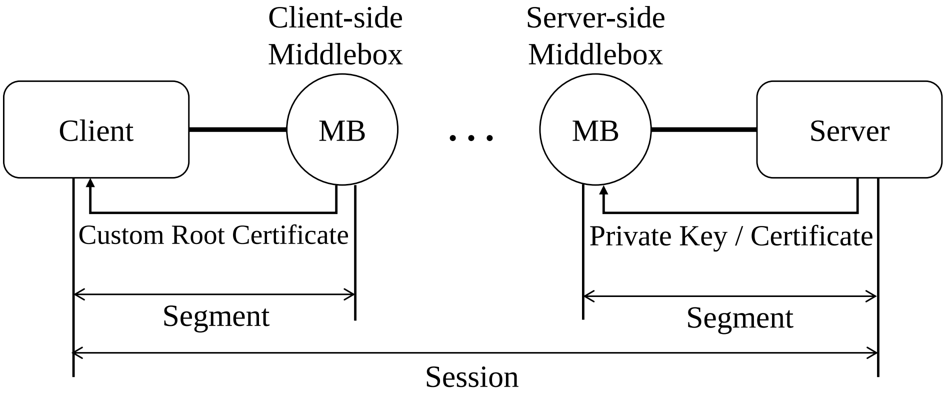


FIGURE 7.1: **Overview of SplitTLS:** A client sets up a TLS session with a server involving multiple middleboxes in-between. During a TLS handshake, each middlebox splits the TLS session into two TLS segments.

between the client and server, with each middlebox ultimately playing both the roles of client and server during each round trip.

Once the end-to-end session is established, the client and the server communicate via the middleboxes. When a middlebox receives an encrypted message over a segment, it decrypts the message using the key of the segment. Then, the middlebox performs its functionality on the decrypted message. Finally, the middlebox encrypts the message with the key for the next segment and forwards it to the next middlebox (or the endpoint). Note that we are interested only in those middleboxes that participate in two *segments* simultaneously; for instance, we do not consider middleboxes that play the role of the intended servers to service the content such as edge servers in CDNs, since they do not always participate in two segments.

Depending on which entity installs the middleboxes and where they are deployed, we can classify middleboxes into two categories: client-side and server-side. **Client-side middleboxes** are employed by users (e.g., anti-virus software) or operators of client-side networks (e.g., intrusion detection systems). They are located at vantage points which packets always pass through. For example, a secure gateway, such as Bluecoat system[3], can be situated at the edge of a corporate network to inspect all the incoming and outgoing packets. **Server-side middleboxes** are deployed by web servers or by the contracts between the web servers and middlebox service providers. They are deployed on a server's networks, or in clouds that provide middlebox-as-a-service [SHS+12]. A client typically accesses server-side middleboxes through DNS routing. For example, when a server employs an outsourced web application firewall, such as Cloudbric[4], he changes the DNS zone file in his authoritative name server to direct traffic from clients to the firewall. After the firewall's inspection, the traffic is then forwarded to web servers or to further middleboxes based on the IP address configuration in the firewall settings.

[3] https://www.symantec.com/products/proxy-sg-and-advanced-secure-gateway

[4] https://www.cloudbric.com/

Also, different techniques are used to intercept TLS sessions, depending on the middlebox type. For a client-side middlebox, clients are often required to install custom root certificates into the trusted root certificate store on their devices. Whenever a middlebox receives a TCP SYN packet sent to the server from the client, it intercepts the packet, executing a TCP handshake and then performing a TLS handshake with the client. During the TLS handshake, the middlebox generates a new certificate on-the-fly with the same common name as the intended server, which is signed by the private key that corresponds to the custom root certificate. Thus, if an attacker learns any private key of a custom root certificate, he can impersonate any server to which the client that trusts the custom root certificate wishes to connect. Furthermore, as the certificate is not issued by CAs, clients cannot verify its legitimacy by other means, such as through CT or DANE [SH12]. For server-side middleboxes, web servers are required to hand over their private keys along with the certificates so that the middleboxes can service their content. This breaks the fundamental principle of authentication and weakens the security of the servers, which makes middleboxes attractive targets for attackers [MWNG12, CCC+16].

Security Problems in SplitTLS.
Although SplitTLS complies with the current TLS practice, several studies have reported that some middleboxes fail to correctly validate certificates, degrade to weaker ciphersuites, or insert malicious scripts [dCdCM16, DMS+17, TII+18, CCM16]. This means that fundamental security properties (i.e., authentication, confidentiality, and integrity) between two endpoints are broken. The client is forced to trust the behavior of middleboxes, since the security of the session is highly dependent on whether the middleboxes correctly operate the TLS protocol. We summarize how SplitTLS breaks the security goals of TLS.

- **Authentication:** A client cannot authenticate the intended server, as the middlebox replaces the server's certificate with a certificate forged by the middlebox. Even worse, recent studies have shown that some middleboxes do not validate the certificate of the intended server. For example, PrivDog [Ash15] was known to accept *every* certificate without checking its validity, and some anti-virus software *always* generates valid certificates even when it received invalid certificates from the intended servers (or another middlebox) [dCdCM16, CCM16].
- **Confidentiality:** Because a middlebox splits the original session into two segments, the client negotiates the key for the segment with the middlebox, not the intended server. Thus the middlebox can read or modify all traffic between the client and the server. Further, the client has no idea of whether the data has been encrypted (with a strong ciphersuite) after it passes through the middlebox. For example, when a client sends an HTTPS request to a server by using Nokia's Xpress Browser, it forcibly sends all messages to the Nokia's forward proxy. Then, this proxy delivers the messages on behalf of the client to the server. However, the Xpress Browser does not notify the clients that their information can be read or modified by the proxy [Mey13, Gau13].
- **Integrity:** SplitTLS cannot guarantee the integrity as a client cannot detect any modification by a middlebox on her messages with the intended server.

> For example, Lenovo laptops performed a man-in-the-middle attack to inject sponsored links on web pages (delivered over TLS) using Superfish [Sep15], but this injection behavior was not noticeable by the ordinary client.

The above problems take place mainly because it is difficult for a client to detect which middleboxes meddle in the session and what they do to the traffic. Several works have demonstrated how these issues, along with the underlying internet infrastructure, can lead to large-scale vulnerabilities. Frack et al. [CCC+16] showed that content providers sharing a private key with a hosting provider (such as CDNs) may significantly affect the security of the HTTPS ecosystem; an attacker who compromises ten hosting providers is estimated to obtain the control of 45% of all content providers. Lin-Shung et al. [HREJ14] demonstrated that there were a large number of forged certificates in the wild, most of which were generated by client-side middleboxes. They also showed that these certificates can be used to trick victims, who had installed the root certificates of the forged certificates.

There have been two IETF drafts that highlight the problems with HTTPS middleboxes and propose new design principles. Both Nottingham [Not14] and Narayanan [Nar13] emphasize that endpoints should be aware of middleboxes, and that their modifications on the messages should be detectable. Therefore, it is necessary to make middleboxes *visible* to clients and publicly *auditable* in order to address the above security and privacy challenges.

Related Work

There have been multiple proposed schemes for enhancing TLS sessions to support middleboxes:

1. `Explicit Trusted Proxy [LMS+12]`: This work proposes that middleboxes should have their own certificates for authentication. Each middlebox certificate should be an EV certificate with proxyAuthentication value in the ExtendedKeyUsage field. This makes middleboxes visible with their certificates; however, endpoints can only authenticate the immediately adjacent middleboxes, and cannot get any information about the other middleboxes.

2. `TLS Keyshare extension [Nir12]`: In this protocol, the client initiates a TLS handshake by sending information about authorized middleboxes to the server. During the handshake, the middleboxes inspect the TLS handshake message and notify the endpoints of any unsupported ciphersuites. After the session is established by the endpoints, the authorized middleboxes receive the session key from the endpoints, allowing them to perform their functionality. Since the same key is shared across all the segments, the keystream is reused, which weakens overall security. Furthermore, this work does not consider modification-related properties.

3. `TLS ProxyInfo extension [TWE+04]`: Each split segment is separately established, as in the maTLS protocol. All the middleboxes pass their certificates

and negotiated security parameters with their signatures to the endpoints, who can authenticate all the middleboxes and confirm security parameters. However, in this protocol, the endpoints must blindly trust the information about each segment from each middlebox. Furthermore, data source authentication, modification accountability, and path integrity are not considered.

4. `Multi-context TLS (mcTLS) [NSV+15]:` mcTLS aims to restrict the behavior of middleboxes by applying the least privilege principle. Endpoints generate two MAC keys for middleboxes: read and write. If a middlebox is authorized to read and write, it obtains both MAC keys. If it can only read the TLS traffic, it gets only the read MAC key. All the middleboxes are authenticated from their certificates. However, as mcTLS uses one session key, it undermines the security of the session if any of middleboxes involved is a writer. Furthermore, after modification by a writer middlebox, the receiver cannot know who has sent the data.

5. `Transport Layer Middlebox Security Protocol (TLMSP):` TLMSP is an improved version of mcTLS, which is being standardized in ETSI. Based on mcTLS, it optionally introduces an audit trail that records each middlebox's inbound HMAC and outbound HMAC to check the modification by the middlebox and the order of the middleboxes in the chain. However, TLMSP uses a top-down approach, which is not suitable for incremental deployment.

6. `BlindBox` [SLPR15] and `Embark` [LSP+16] allow a monitoring gateway (in the client's network) to read TLS traffic without revealing its content to middleboxes on a third party cloud. To this end, they introduce a secondary channel using a special encryption technique (such as searchable encryption or order-preserving encryption). The client communicates with the server over a TLS session, and delivers the packets to the middleboxes via the secondary channel before the client sends packets to the server. Private data are not leaked to the middleboxes in these proposals, but they have two main drawbacks. First, the possible functionality of middleboxes is limited by the encryption techniques. Second, they require another round trip to middleboxes over the secondary channel before sending the data to the other endpoint.

7. `SafeBricks` [PLPR18], `ShieldBox` [TKG+18], and `SGX-Box` [HKHH17] focus on guaranteeing security and protecting privacy (from middleboxes) by building middleboxes over a trusted execution environment. The three schemes have different properties. For example, `SafeBricks` aims to apply the least privilege principle to middleboxes by using a type-safe language. `ShieldBox` seeks to supports syscalls in an enclave, and `SGX-Box` offers programmability to middlebox developers for easy deployment.

7.3 Case Study: mbTLS

In this section we perform a case study of the proposed middlebox-enabled TLS scheme mbTLS [NLG+17]. We highlight a skipping attack, akin to those identified in Chapter 6, which can be found in several similar related works. This problem arises from the problem of distributing (and verifying the usage of) keys in a multiparty setting.

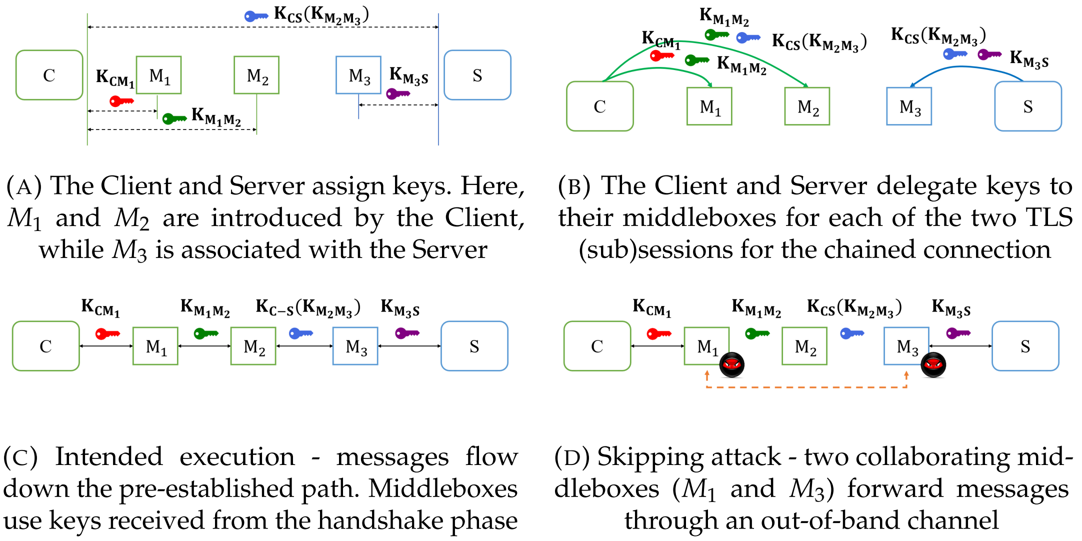
The core structure of the mbTLS protocol is as follows:

- During the mbTLS handshake between a client and a server, middleboxes report their presence to the endpoints with an additional handshake.
- At the end of the handshake phase, each middlebox is delegated two session keys for its associated sessions (i.e. one where it acts as a client, and one as a server) from the endpoint which has deployed them, rather than generating the keys themselves.
- During the record phase, messages between the endpoints are passed down the chain of middleboxes, which decrypt then re-encrypt each message (whilst performing any of their functions on the message body)

The mbTLS scheme assumes that middleboxes are running on hardware enclaves, such as the Intel SGX framework [Int17]. As a result, the authors assert that middleboxes can be seen as trusted agents for the purpose of security analysis. However, we argue that this is not a realistic security model. Software enclaves are designed to ensure that software is loaded without modifications after distribution, and is run in a secure environment. This does not provide any guarantees about security of the software itself. As such, maliciously designed middlebox software is not protected against. This means that an attacker could write malware which poses as a service-providing middlebox (such as an ad-blocker), but instead fulfills some other purpose (such as leaking data). In addition, we note the existence of several attacks on trusted execution environments [VBMW+18, BMD+17, VBWK+17, HCP17], suggesting that even well-intending participants may accidentally leak secret data. Because of this, we argue that the Dolev-Yao model is more appropriate.

Middlebox Skipping Attack on mbTLS.
The mbTLS protocol admits an attack. In this scenario, a pair of (non-adjacent) adversary-controlled middleboxes transmit messages between each other, skipping over other TLS segments in the path. As a result, honest middleboxes are prevented from performing their functionality.

The structure of the attack is shown in Figure 7.2. The cause of the vulnerability is that an endpoint cannot detect the difference between a middlebox choosing not to modify a message, and the middlebox never having received the message at all. As such, an attacker who controls two (non-adjacent) middleboxes on the path can simply forward the body of messages from the first to the second. This allows for the bypassing of service-providing software such as adblockers or content filters.



(A) The Client and Server assign keys. Here, M_1 and M_2 are introduced by the Client, while M_3 is associated with the Server

(B) The Client and Server delegate keys to their middleboxes for each of the two TLS (sub)sessions for the chained connection

(C) Intended execution - messages flow down the pre-established path. Middleboxes use keys received from the handshake phase

(D) Skipping attack - two collaborating middleboxes (M_1 and M_3) forward messages through an out-of-band channel

FIGURE 7.2: mbTLS handshake phase conclusion and skipping attack

To prevent such an attack, a simple approach is to add some form of read-receipt to messages, in the form of a MAC or signature from each middlebox in the path. With such an approach, the endpoints can confirm at the end of the execution that the path was followed faithfully.

7.4 The maTLS Protocol

In this section we introduce the threat model we consider for designing middlebox-enabled TLS extensions. With this model in mind, we present the maTLS protocol – our proposed solution. We also discuss some of the factors to consider when rolling out such a scheme.

The maTLS protocol consists of two main components: the handshake phase, in which middleboxes declare their presence to both endpoints and keys are established, and the record phase, in which data is exchanged.

Trust and Threat Models

Before introducing our threat model, we describe five entities in the networking architecture.

(1) *Client (C)*: A client refers to a machine or a piece of software (e.g., web browsers), used by a *user*, that communicates with middleboxes. We assume the client correctly performs protocols and is not compromised.

(2) *Server (S)*: A server refers to a machine or a piece of software, operated by a *content provider*, that services content based on a client's request. We assume that the server to which a client wishes to connect is not malicious or compromised. The client and the server are collectively referred to as *endpoints*.

(3) *Middlebox (MB)*: a middlebox is a machine or a piece of software, made by a *middlebox service provider*. A middlebox is deployed by a network operator, a content provider, or a user and is located between the client and the server. The endpoints may not be aware of the middleboxes, their functions, or their states. If the middleboxes are mis-configured or incorrectly implemented, they may accept invalid certificates, use deprecated ciphersuites, or attempt to inject unwanted or malicious content [TII+18, ORSZ16].

(4) *Certificate Authority (CA)*: An organization that issues and revokes certificates. A CA issues a certificate to a requester after a validation process. In our model, A CA can be compromised; thus, fraudulent certificates can be issued to an adversary who can impersonate the server.

(5) *Middlebox transparency (MT)*: A system (similar to CT [LLK13]) that logs certificates, which can be publicly monitored and audited by any interested parties. Any *trusted* CT operator, such as Google, can operate an MT system. The only difference from CT is that the MT system targets *middlebox certificates*, which will be detailed in Section (§7.4). Alternatively, the CT system can be assumed to accommodate middlebox certificates as well.

We accept the Dolev-Yao model [DY83] in which an active adversary can fully control the network; that is, the network is untrusted. The adversary can not only capture messages on-the-fly, but also modify, drop, reorder, or inject messages. Specifically, he can manipulate middleboxes (e.g., TLS-intercepting WiFi access points), which then can capture packets, perform crypt-analysis, or patch software to inject malicious scripts. We do not consider other attacks such as side-channel attacks or denial-of-service attacks.

Note that we consider both endpoints to be honest, as our primary security goals revolve around their ability to establish a secure channel. We assume that the client has chosen their intended communication partner S at the time of initiating the protocol. However, we do allow for corrupt servers who may attempt to falsely convince the client that they are S.

maTLS Extension Overview

We now provide an overview of the maTLS protocol, before we later go into some of the specific details. Table 7.1 contains an overview of the notation that will be used in this section.

The maTLS Handshake Protocol.
A client performs a maTLS handshake to negotiate accountability keys, to authenticate the server and middleboxes, and to perform security parameter verification. The maTLS handshake protocol, which extends TLS 1.2, is shown in Figure 7.4a. In the first round-trip, the client expresses its preference to perform the maTLS protocol by adding the *Middlebox_Aware* extension to the `ClientHello` message. The client generates its DH key pair (say, (a, g^a)) and inserts the DH public key (g^a) into the extension. Then, the client sends the `ClientHello` message with the highest possible

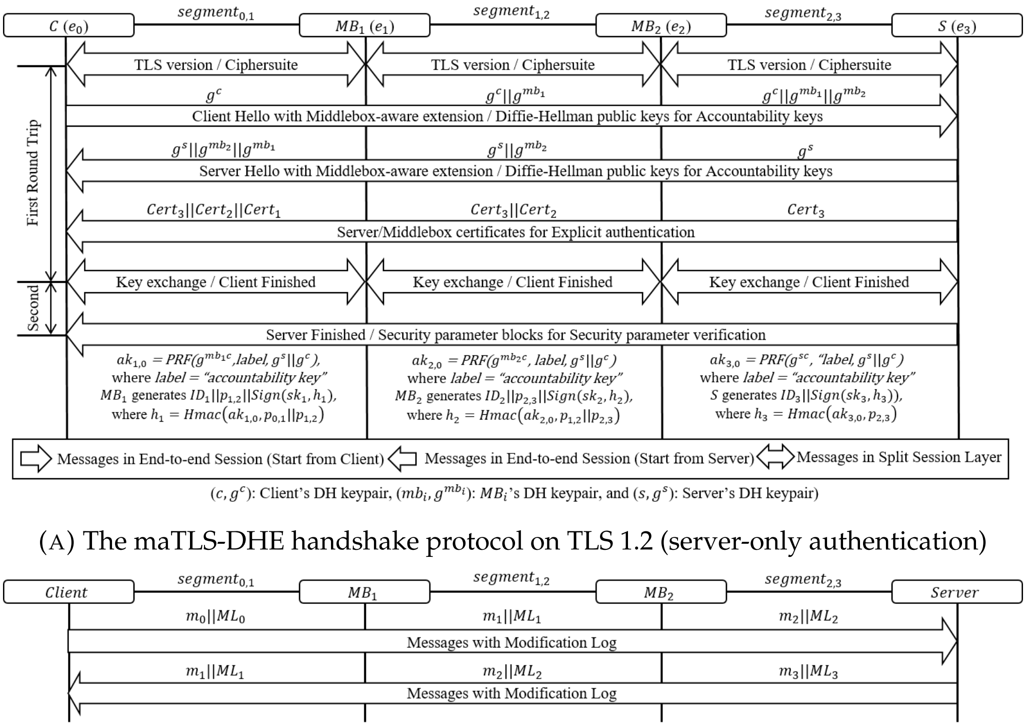
Notation	Meaning
C	Client
S	Server
MB_i	ith Middlebox in the session $(1 \leq i \leq n-1)$
e_i	ith Entity in the session where $(e_0 = C, e_n = S)$
$segment_{i,j}$	The maTLS segment between e_i and e_j
m_i	Message sent from e_i
$a\|\|b$	a concatenated with b
$PRF(a,b,c)$	Pseudorandom function in [Die08] to derive keys $(a: \ secret, b: \ label, c: \ seed)$
$Sign(k,m)$	Signature function on m with a key k
$H(m)$	Hash function on m
$Hmac(k,m)$	Keyed hash-based MAC function with a key k on m
$Ae(k,m)$	Authenticated encryption on m with a key k
(sk_i, pk_i)	Entity e_i's (secret key, public key) pair
$Cert_i$	Entity e_i's certificate
ID_i	Identity of e_i. $ID_i = H(pk_i)$
g	Generator of a DH group
(a, g^a)	Ephemeral DH key pair
$p_{i,j}$	Security parameters that includes the negotiated version, the negotiated ciphersuite, the hashed master secret, and the transcript between e_i and e_j
$ak_{i,j}$	Accountability key of e_i established with e_j (We simply write ak_i when j is fixed in the context)
$HMAC_i$	The result of $Hmac(k,m)$ by e_i
ML_i	Modification log generated by e_i

TABLE 7.1: Notation used in describing the maTLS extension

TLS version and a set of supporting ciphersuites. On receiving the `ClientHello`, each middlebox finds the client's maTLS extension, generates its own DH key pair, and extracts the list of DH public keys from the maTLS extension. After that, it appends its own DH public key, and sends the new `ClientHello` with the DH public keys toward the client's intended server. This process is repeated at every middlebox on the way to the server.

The server generates its own DH key pair (say, (b, g^b)) and sends the `ServerHello` message with the DH public key (g^b) and the selected TLS version and ciphersuite for the maTLS segment. On receiving `ServerHello`, each middlebox processes the message as the middlebox do on `ClientHello` and determines the TLS version and the ciphersuite to be used in the maTLS segment.

Then, each entity negotiates the TLS version and the ciphersuite with its neighbor entity for each maTLS segment. Furthermore, both endpoints receive the DH public keys from all entities and each middlebox has two DH public keys (i.e. the client's and the server's). With their own DH private keys, all entities generate the accountability keys by using the `PRF` function defined in [Die08] with the server's DH public key and the client's DH public key as seeds. For a `label`, one of the input parameters of the `PRF` function, we use the string, "accountability key."


segment_{0,1}
segment_{1,2}
segment_{2,3}
C (e_0)
MB_1 (e_1)
MB_2 (e_2)
S (e_3)
First Round Trip
Second
TLS version / Ciphersuite
g^c
g^c||g^{mb_1}
g^c||g^{mb_1}||g^{mb_2}
Client Hello with Middlebox-aware extension / Diffie-Hellman public keys for Accountability keys
g^s||g^{mb_2}||g^{mb_1}
g^s||g^{mb_2}
g^s
Server Hello with Middlebox-aware extension / Diffie-Hellman public keys for Accountability keys
Cert_3||Cert_2||Cert_1
Cert_3||Cert_2
Cert_3
Server/Middlebox certificates for Explicit authentication
Key exchange / Client Finished
Server Finished / Security parameter blocks for Security parameter verification
ak_{1,0} = PRF(g^{mb_1c}, label, g^s||g^c), where label = "accountability key" MB_1 generates ID_1||p_{1,2}||Sign(sk_1, h_1), where h_1 = Hmac(ak_{1,0}, p_{0,1}||p_{1,2})
ak_{2,0} = PRF(g^{mb_2c}, label, g^s||g^c) where label = "accountability key" MB_2 generates ID_2||p_{2,3}||Sign(sk_2, h_2), where h_2 = Hmac(ak_{2,0}, p_{1,2}||p_{2,3})
ak_{3,0} = PRF(g^{sc}, "label, g^s||g^c) where label = "accountability key" S generates ID_3||Sign(sk_3, h_3)), where h_3 = Hmac(ak_{3,0}, p_{2,3})
Messages in End-to-end Session (Start from Client)
Messages in End-to-end Session (Start from Server)
Messages in Split Session Layer
(c, g^c): Client's DH keypair, (mb_i, g^{mb_i}): MB_i's DH keypair, and (s, g^s): Server's DH keypair)
Client
MB_1
MB_2
Server
m_0||ML_0
m_1||ML_1
m_2||ML_2
m_3||ML_3
Messages with Modification Log

(A) The maTLS-DHE handshake protocol on TLS 1.2 (server-only authentication)

(B) The maTLS record protocol with a modification log.

FIGURE 7.3: **The maTLS protocol.** The maTLS handshake protocol is responsible for explicit authentication and security parameter verification, while the maTLS record protocol executes valid modification checks.

The ServerCertificate message is sent after the Hello messages. The server sends its own certificate and each middlebox appends its middlebox certificate. The client performs explicit authentication in order to accept the server and the middleboxes. Then, the client maps each accountability key to the corresponding identity, where an identity is a digest of an entity's public key. Although the server does not receive the certificates, the server can identify the client from the accountability key.

After receiving the certificates, each maTLS segment exchanges key materials via the ServerKeyExchange and ClientKeyExchange messages. Using the key material, all entities generate shared secrets of the segment.

Finally, Finished messages are exchanged to verify the handshake between two peers in each segment, followed by a newly defined ExtendedFinished message that includes security parameter blocks from the server to the client. The client performs security parameter verification and confirms the proofs of private key possession by verifying the signatures by processing the ExtendedFinished message.

The maTLS Record Protocol.
The maTLS record protocol provides data source authentication, modification accountability, and path integrity during data exchange. The maTLS record protocol is illustrated in Figure 7.4b. For each message, the record protocol generates the data source, initializes an ML, and inserts its source MAC. On receiving the message and its ML, each middlebox processes the ML as mentioned earlier. A read-only

middlebox extracts the final HMAC from the ML, performs the HMAC operation over the previous HMAC to put its fingerprint, and updates the MAC. A writer middlebox appends the modification MAC to the ML.

Upon receipt of the message, the destination performs valid modification checks by validating the ML, aborting the connection if there has been an invalid modification by middleboxes. The destination also verifies the source of the incoming message; for example, a server can abort the connection if the HTTP request message (over maTLS) did not originate from the client. Furthermore, since all the middleboxes in the session leave their own MACs in the ML whenever the data is passed the middleboxes, the endpoints can confirm whether the order of the middleboxes is preserved by verifying the MACs with the accountability keys in sequence.

Auditable middleboxes

We now describe an architecture to make middleboxes visible to the endpoints of TLS sessions. To this end, we define the notion of an *auditable middlebox* that has its own *middlebox certificate* logged in *middlebox transparency* (MT) servers. Middlebox certificates are written based on the X.509 format, and then signed by CAs, which may require middlebox service providers to follow a set of established criteria for certificate issuance. Like TLS certificates, middlebox certificates could also be mis-issued, mis-configured, or exploited. To mitigate those attacks, we also introduce *MT log servers* where any middlebox certificates can be publicly logged so that interested parties can monitor and detect unexpected behaviors.

Middlebox Certificates.
The primary purpose of middlebox certificates is to help users authenticate middleboxes, by providing information about behaviour of the middlebox. For example, the role of the middleboxes (e.g., firewall) or permissions (e.g., read or write) can be included. This information can be added into the format of X.509 certificate without any modification to the existing infrastructure. Below, we itemize the required information for a middlebox certificate along with the names of the fields.

- **Name(s) of the Middlebox Service Provider** indicates the name(s) of the middlebox service provider, which can be specified at the `CommonName` field.
- **Subject (Middlebox) Public Key Info** carries the public key and the cryptographic algorithm (e.g., ECC) used to generate the key, which can be specified at the `Subject Public Key Info` field.
- **Middlebox Information Access** contains additional information that can help a user trust the middlebox. To this end, we define an extension, `Middlebox_InfoAccess` where its ASN.1 syntax is defined as follows.

Certificate Extension Syntax

```
Middlebox_InfoAccess :: =
  SEQUENCE SIZE (1..MAX) OF Middlebox_Description

Middlebox_Description::=  SEQUENCE {
  Middlebox_InfoType     OBJECT IDENTIFIER,
  Middlebox_Info         GeneralName}
```

For example, *permission* can be one of the `Middlebox_InfoType` fields, used to indicate the read or write permission required by the middlebox for TLS traffic. Similarly, the *TypeofService* and *URL* fields can provide additional information about the middlebox as a form of `Middlebox_Description`.

Middlebox Transparency

We introduce the notion of a middlebox transparency log server, which publicly records middlebox certificates. The operation of MT is similar to that of certificate transparency [LLK13]. It encourages middlebox service providers or CAs to submit middlebox certificates to the MT log server. Further, once a middlebox certificate is accepted at the MT log server, the log server returns a Signed Certificate Timestamp (SCT). A client can check its membership by verifying the SCT with the public key of the log server.

Properties of Auditable Middleboxes.
We call a middlebox that has a middlebox certificate logged in an MT log server an *auditable middlebox*. It provides the following benefits regarding the *trustworthiness* of middleboxes:

First, middleboxes now have their own key pairs and can be authenticated from the endpoints by presenting their *valid* certificate. Thus, middleboxes now no longer require (1) content providers to share their private keys or (2) users to install their custom root certificate.

Second, clients can be assured of the names and properties of middleboxes or middlebox service providers. This will hold middlebox service providers accountable. Further, with the help of maTLS, which will be detailed in §7.4, clients can detect if a middlebox has modified traffic without any authorization. This can be done by checking the *Permission* item in the `Middlebox_InfoAccess` field of the middlebox certificate, which would encourage middleboxes to have *least* privileges. For example, anti-virus software can be issued with a middlebox certificate with only read permission to assure users that it will not modify any traffic.

Third, middlebox certificates may require some of the essential X.509 extensions such as *Permission* field to be set to `critical` [ITU00], which explicitly indicates that clients must refuse the connection if they cannot interpret the extension.

Fourth, the MT system provides a global set of auditable middleboxes; any interested parties, such as monitors, auditors, and clients, can check any mis-issued, mis-configured, or fraudulent certificates.

Fifth, when a middlebox certificate's corresponding private key is no longer safe due to security breaches, the middlebox certificate can be revoked, and the revocation status can be disseminated through existing revocation mechanisms such as CRL [CSF+08] or OCSP [MAM+99]. Thus, clients can be protected from middleboxes with security risks by leveraging the existing revocation mechanisms.

Given that the PKI has been suffered from many security issues regarding certificate management, one might be concerned that introducing additional infrastructure (i.e., MT system) could exacerbate the current situation. However, we believe that the middlebox certificate by itself *does not* introduce new management problems as it can be easily integrated into the existing CT architecture. Rather, the use of middlebox certificates can mitigate the current insecure practices of middleboxes splitting TLS connections such as installing custom root certificates or sharing private

Middlebox-aware TLS Design

In this section, we describe the maTLS protocol, which is designed to allow middleboxes to participate in a TLS session. As we have middleboxes equipped with certificates, we extend the security goals of TLS to the seven objectives below, divided into three categories. For the sake of exposition, we explain maTLS based on TLS 1.2 with ephemeral Diffie-Hellman (DHE) key exchange in the server-only authentication mode.

Deployment

In this subsection we discuss some of the strategies for deploying a middlebox-enabled TLS scheme such as maTLS.

Session Establishment Approaches.
First of all, we explain how a client establishes a maTLS session with the server through multiple middleboxes. There are two possible approaches to establish a maTLS session and its segments, as shown in Figure 7.4. In the top-down approach, the client first establishes a TLS session directly with the server, and the server determines the security parameters of the session. After that, either or both of the endpoints should pass the segment keys to the authorized middleboxes via separate TLS connections. In the bottom-up approach, the client and middleboxes first initiate TLS segments sequentially up to the server. In this approach, the two participants of each segment negotiate their security parameters individually, and the session is eventually constructed from these segments.

In maTLS, we adopt a *bottom-up approach* for the following reasons. First, an maTLS session can be partially established even if not all entities support maTLS. For example, even if the server does not support maTLS, the client and the next middlebox

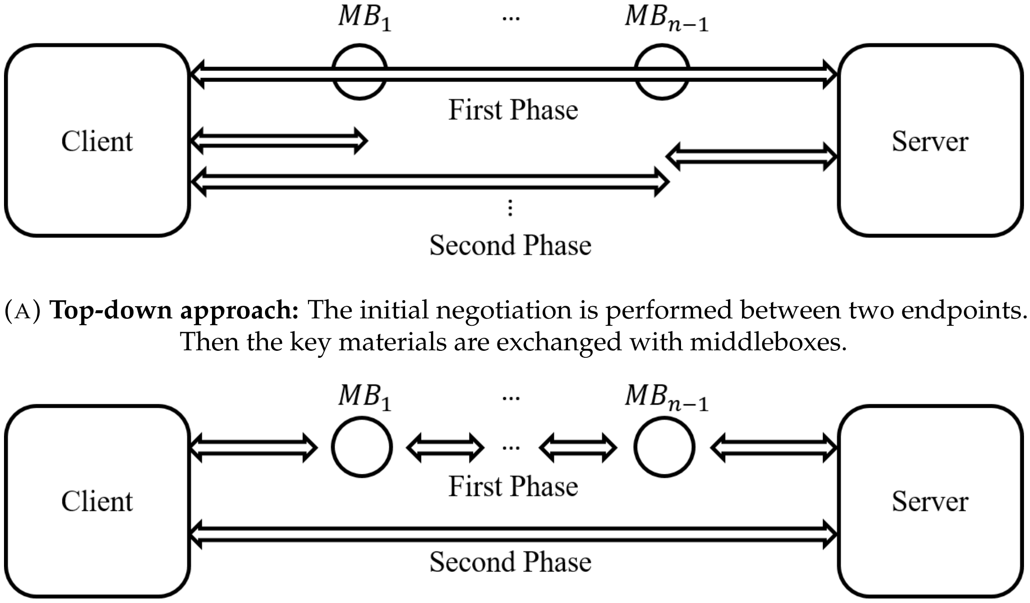
Mechanism	Proof Data Structure	Description & Advantages
Explicit authentication	A sequence of certificate blocks, including the server certificate and any middlebox certificates with their signed certificate timestamps.	The client authenticates the server and middleboxes by checking their certificates, and confirms their names and the middleboxes' permissions • No custom root certificate and no private key sharing • EV certificates are not degraded due to fabricated certificates • Support for Certificate Transparency [LLK13] and DANE [SH12]
Security parameter verification	Security parameters of each maTLS segment including the TLS version, ciphersuite, and a transcript of the handshake	The client confirms the confidentiality of each segment • Neither a low TLS version nor a weak ciphersuite is permitted without the client's knowledge • The two points of each segment perform a TLS handshake and establish a segment key
Valid modification checks	A modification log that keeps track of the modifications of a packet	The client confirms that only authorized entities can generate or modify messages • Only an authorized data origin (a server or a cache proxy) can generate messages • Only trusted writer middleboxes can modify messages • The order of middleboxes is always preserved

TABLE 7.2: **Three audit mechanisms of endpoints in maTLS:** Explicit authentication guarantees the authentication of all the participants. Security parameter verification ensures the confidentiality of all the maTLS segments. Valid modification checks ensure that only authorized entities can modify messages.

that supports maTLS can still negotiate security parameters for their segment and establish a maTLS session. Second, each different maTLS segment can benefit from using strong ciphersuites or newer TLS version independently because maTLS does not require all entities to share the same ciphersuite or TLS version. Third, the bottom-up approach efficiently achieves *Individual Secrecy*. This is because the two entities involved in each segment use different random numbers to establish a master secret; thus, the probability that all the segment keys are identical is negligible.

It is worth noting that most of the top-down approach schemes, such as mcTLS [NSV+15], TLMSP, and TLS Keyshare extension [Nir12], do not support incremental deployment. This is mainly because only the server picks the version, ciphersuite, and extensions that are supported across all entities (i.e., both endpoints as well as middleboxes), which makes it challenging to deploy them incrementally. Even worse, it is highly likely that the security level of the session will be decided by the "intersection" of the security parameters supported by all the entities. Furthermore, the entire session needs to use the same shared secret, which undermines the security of the communication as well.

Among the top-down approach schemes, the only solution that supports incremental deployment is mbTLS [NLG+17]. If the server does not support mbTLS, the client first establishes a standard TLS session with the server. Then, the client sends the segment keys to each middlebox that does support mbTLS. To achieve individual secrecy, the client generates the different segment keys for all the segments and distributes keys to the corresponding middleboxes (two segment keys per one middlebox), which is inefficient.



(A) **Top-down approach:** The initial negotiation is performed between two endpoints. Then the key materials are exchanged with middleboxes.

(B) **Bottom-up approach:** The two participants of each maTLS segment negotiate security parameters independently, and then the maTLS session is established by connecting the maTLS segments.

FIGURE 7.4: Two approaches to establish a TLS session with middleboxes. We adopt the bottom-up approach since it efficiently supports incremental deployment.

Audit Mechanisms.
We propose three audit mechanisms for the clients to audit middleboxes while performing an maTLS session: *Explicit Authentication*, *Security Parameter Verification*, and *Valid Modification Checks*.

These mechanisms necessitate some data structures for middleboxes, such as signatures or message authentication codes (MACs), to demonstrate accountability for every message. We prefer to use MACs, as signatures require higher computation overhead on their generation. Thus, entities will use hash-based message authentication codes (HMACs) when signatures are not necessary. To this end, we introduce *accountability keys* that are to be used as HMAC keys. The accountability key is established between the endpoints and middleboxes; thus, each middlebox should establish one accountability key with each endpoint (two in total), while the client and the server each need one accountability key for each middlebox, and share one more key between them.

We overview the audit mechanisms in Table 7.2.

(1) *Explicit Authentication* guarantees authentication of the server as well as the middleboxes by validating received certificates. If there are any suspicious middleboxes, the maTLS session can be aborted. The server sends its certificate in the `ServerCertificate` message during the maTLS handshake. Whenever the middleboxes receive this message, each of them simply appends its certificate, so that the client can receive all the certificates up to the server. As the client receives all the certificates, she does not need to worry about the degradation of certificate-level due

to forged certificates by middleboxes. Similarly, DANE or CT can also be supported with middleboxes.

When receiving a sequence of certificates, the client should validate all of the certificates as well as recording the order of the certificates, up to the server.

(2) *Security Parameter Verification* allows the client to audit the security association of each maTLS segment, and to confirm the accountability keys as well as their order. To this end, the middleboxes have to present the security parameters (of each segment), that is, the chosen TLS version, the negotiated ciphersuite, the hashed master secret, and a (hashed) transcript of the TLS handshake (i.e., the `verify_data` in the `Finished` message). The selected TLS version and ciphersuite show the degree of confidentiality of the corresponding maTLS segment. The hashed master secret demonstrates the uniqueness of segment keys. The transcript, a digest of handshake messages in the maTLS segment, is used to prove that two entities involved in the segment performed the handshake without any modification by an attacker.

However, middleboxes could potentially give false information to the client. To avoid such misbehavior, we propose a *security parameter block* – an unforgeable cryptographic proof of security information for each segment. Each block contains the security parameters and their HMAC value.

The two entities of a maTLS segment, say $segment_{i,i+1}$, present the security parameters of the segment, respectively for cross-verification.

All the entities except the client in the maTLS session generate the security parameter block. The basic structure of the block is of the following form:

$$ID_i||p_{i,i+1}||Sign(sk_i, Hmac(ak_{i,0}, p_{i-1,i}||p_{i,i+1}))$$

One entity e_i first generates an HMAC over the security parameters in its two segments, namely $segment_{i-1,i}$ and $segment_{i,i+1}$, and signs on the resultant HMAC. Then, e_i prepends its identifier and the security parameters of the segment in the direction of the server with the signature. When the block is generated, e_i forwards it toward the client.

For a server $(S = e_n)$ that is only involved in one segment, i.e., $segment_{n-1,n}$, the server sends $ID_n||Sign(sk_n, Hmac(ak_{n,0}, p_{n-1,n}))$ in which the term corresponding to $p_{i,i+1}$ in the above expression is removed.

When the client receives a series of security parameter blocks, it can confirm all security parameters negotiated between each entity by verifying the signature of signed HMACs. Verification fails could be due to modified security parameters, missing or incorrect order of the middleboxes; thus the client must abort the negotiation process. Once the client can successfully verify all the security parameters, accountability keys, the order of the middleboxes in the maTLS session, it can further decide whether to accept the session based on its policy. For example, the client might abort the connection if any of the segments is established with a weak algorithm such as an RC4 [Pop15].

(3) *Valid Modification Checks* allow a client to audit which entity has modified the message.

When an entity forwards a message to the next entity it also generates a cryptographic proof, called a *modification log* (ML). Basically, it is to compare the incoming and outgoing message from the entity by attaching (1) a HMAC generated from both received and sending message using its accountability keys (ak_i), (2) a digest of the received message ($H(m_{i+1})$), and its identifier (ID_i). Assuming that the message is coming from the server (e_n) to the client (e_0), we can define the ML generated from the e_i, which is denoted as ML_i:

$$ID_i||H(m_{i+1})||Hmac(ak_{i,0}, H(m_i)||H(m_{i+1}))||ML_{i+1}$$

Here, we can apply some optimization techniques to reduce the size of the MLs in specific scenarios. First, the server does not have a prior message, thus the ML_n can be defined as $ID_n||Hmac(ak_{n,0}, H(m_n))$. Second, when an entity (e_i) does not modify any message (i.e., in the case of a read-only middlebox), we can further reduce the size of the ML_i by (1) simply generating a $HMAC_i$ from the previous $HMAC_{i+1}$ and (2) omitting its received digest ($H(m_{i+1})$) and even its ID (ID_i). Thus, if the client detects a omitted ID while parsing the received ML, it can assume that the message has not been modified among the middleboxes with the omitted IDs. For example, if an entity (e_i) receives a message that has never been modified, the ML that the entity received will be

$$ID_n||Hmac(ak_{i+1,0}, Hmac(ak_{i+2,0}, \cdots, Hmac(ak_{n,0}, H(m_n))).$$

Once e_i modifies the message, however, the ML produced from e_i will be

$$ID_i||H(m_{i+1})||Hmac(ak_{i,0}, H(m_i)||H(m_{i+1}))||ID_n||HMAC_{i+1},$$

which implies that the message between the middlebox e_{i+1} and e_n has never been modified.

Once the receiver (i.e., the client in this example) obtains the series of MLs, it can extract the digests of all the modified messages, track the identifiers of the middleboxes that performed the write operation, and finally verify each ML using its HMAC.

7.5 Security Verification

In this section we discuss the security goals that we aim to achieve with the maTLS protocol, as well as the process of building a model for analysis.

Security Goal	Description
Server Authentication	When a client believes she has finished a maTLS handshake, the corresponding server also believes he has established a session with the client, sharing the same accountability key data
Segment Secrecy	When a maTLS session is established, the client has correctly verified the security parameters used in each segment
Data Authentication	When a client receives a message during the maTLS record phase, the hash value from the server is a faithful digest of the original message
Modification Accountability	When an endpoint receives a message during the maTLS record phase, the agent believes that a middlebox has changed the message if and only if that middlebox did make a change

TABLE 7.3: Core security goals of the maTLS handshake and record phase protocols.

Security Goals

Authentication: Similar to the authentication process of TLS certificates, clients should be able to receive and check the validity of the certificate of the server that the clients intended to connect. This should hold even when there are middleboxes splitting the TLS connection between them. Thus, we extend the notion of the authentication to cover both the intended server and middleboxes, and we call this property of the maTLS protocol (1) *Server Authentication* (and equivalently, *Middlebox Authentication*).

Confidentiality: Browsers warn a user if her session is negotiated with a low TLS version or a weak ciphersuite. Thus, each maTLS segment should be encrypted with a sufficiently high version of TLS and a strong ciphersuite; we apply this requirement to each maTLS segments, which is called (2) *Segment Secrecy*. Further, each maTLS segment should have its own security association (e.g., a unique session key) to prevent the same keystream from being reused across the overall maTLS session.

Integrity: The notion of integrity can be extended such that only authorized entities can generate or modify messages depending on their permissions. To this end, we define (3) *Data Authentication*, which means that a client should be able to confirm that a received message has originated from a valid endpoint such as a web server or cache proxy. Moreover, a client should be able to figure out which middleboxes have made each modification to the message, ensuring accountability. We call this (4) *Modification Accountability*. Moreover, not only the integrity of the messages should be preserved, but also the order of the middleboxes; the network attacker could also capture and redirect packets, or bypass some middleboxes. We saw the consequences of this in Section 7.3 – as such, we require Path Integrity, as from Chapter 6.

Protocol Rules

The protocol rules for maTLS can be divided broadly into three categories. The first handles the *setup* rules of the protocol. These represent events such as the registration of server or middlebox certificates. Second, a set of *corruption* rules describe the main ways in which an agent may violate their specification — for example, giving their long-term private key to the adversary. Finally, the *protocol* rules describe the actual actions of the participants.

In order to handle the complexity of analysis, we consider the handshake and record protocols separately for analysis. That is, we assume that the record phase cannot begin unless the handshake phase has successfully completed exactly as per the specification. This assumption arises from the Server Authentication security claims, which assert that there is full agreement on all terms by the end of the handshake.

Security Claims

With the protocol rules, we modeled the core security goals of maTLS. We describe our security goals in the form of the first order logic formulae, the intuition behind which are shown in Table 7.3.

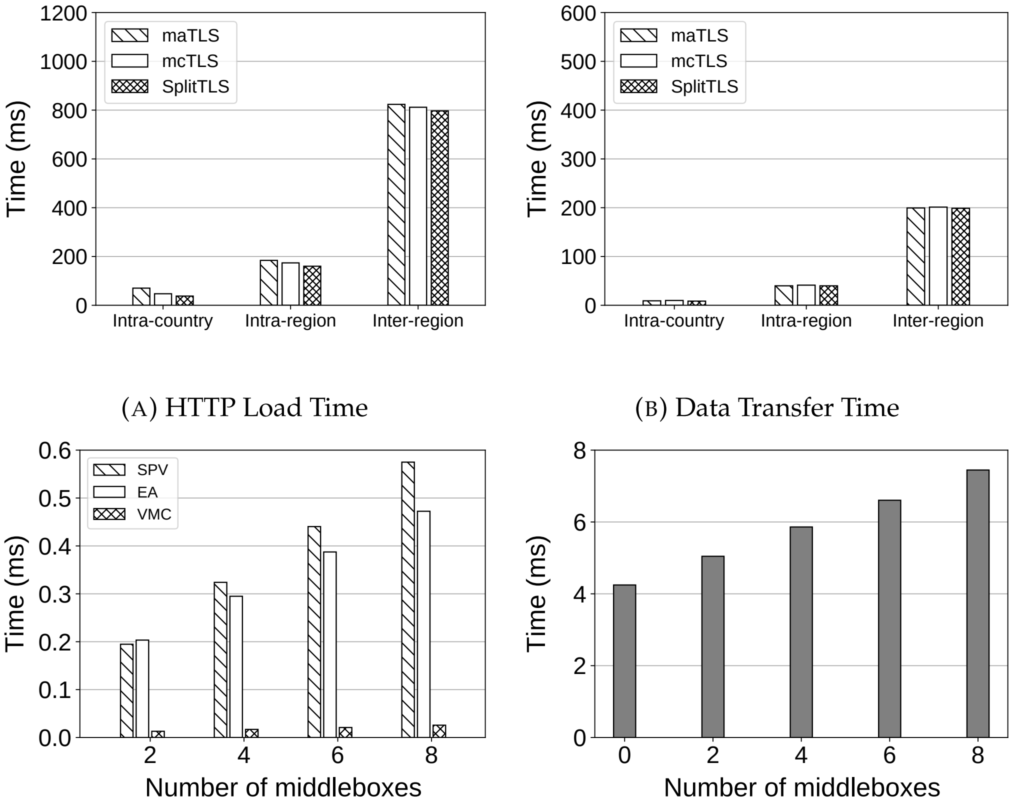
7.6 maTLS Implementation & Evaluation

In this section we evalute an implementation of the maTLS extension, as well as discussing the implications of attempting to deploy such a scheme.

Experiment Settings.
To demonstrate the feasibility of the maTLS protocol, we implemented it using the OpenSSL library. Our testbed consists of a client (C), a client-side middlebox (MB_C), a server-side middlebox (MB_S), and a server (S). The server-side middlebox and the server were equipped with an Intel Xeon CPU E5-2676 at 2.40GHz with 1GB memory. We used a virtual machine with an Intel Core i7 at 2.30GHz and 1GB memory for the client-side middlebox, and a virtual machine with an Intel Broadwell CPU at 3.30GHz and 1GB memory for the client.

During our experiments, the client and the client-side middlebox were located on a campus network. We ran tests with the server (and the server-side middlebox) located at three different locations: in the same country (intra-country testbed), in different countries but the same region (intra-region testbed), and in different continents (inter-region testbed). The round-trip times between the two entities in each scenario are shown in Table 7.4.

After establishing an maTLS session, the client requests an HTML page of 1KB with an HTTP GET message, respectively, terminating the connection after completing the download of the corresponding HTTP response. Each plotted value is the average of 100 measurements. We compare the performance overhead of maTLS with those of SplitTLS and mcTLS [NSV+15].



(A) HTTP Load Time

(B) Data Transfer Time

(C) Integrity Verification Time

(D) CPU Processing Time

FIGURE 7.5: Numerical results reveal that maTLS incurs slightly more delay, ranging from 10.22ms to 32.52ms against mcTLS and SplitTLS, mainly due to the signature verification and key generation needed in the maTLS handshake. (EA: Explicit Authentication, SPV: Security Parameter Verification, VMC: Valid Modification Checks)

We used an ECDH key exchange algorithm over the secp256r1 elliptic curve for the accountability keys, the SHA256 function for the hash algorithm, and a SHA256-based ECDSA for the signature algorithm.

HTTPS Page Load Time.

We first evaluate the time elapsed to fetch an 1KB file from the server in the maTLS protocol, which is compared to the SplitTLS and mcTLS protocols. Figure 7.5a summarizes the time taken from starting a TCP handshake to finishing the download of the content. We observe that the maTLS protocol introduces a slight delay (10.22ms – 32.52ms) compared to SplitTLS and mcTLS in the general case.

We believe this is mainly due to the message order dependency in maTLS. Unlike SplitTLS, where each TLS segment is established *completely independently*, the maTLS segments are established piecewise *sequentially* as some signaling messages (e.g., `ClientHello, ServerHello, ServerCertificate`) must be exchanged between the client and the server through the middleboxes in sequence. Thus, in maTLS, each middlebox needs to wait until these messages arrive while performing the handshake.

To quantify the overhead that the maTLS record protocol requires, Figure 7.5b shows the data transfer time, which starts at the client sending an HTTP GET (a single packet)

Testbed	C-MB_C	MB_C-MB_S	$MB_S - S$
Intra-country	1.136ms	4.944ms	0.551ms
Intra-region	1.136ms	35.896ms	0.537ms
Inter-region	1.136ms	192.818ms	0.610ms

TABLE 7.4: **Networking Settings.** The round-trip times between two points in each scenario are shown, where C and MB_C are in the same campus, and MB_S and S are in the same data center.

and ends at the client receiving an HTTP RESPONSE (a single packet). Interestingly, we notice that the delay time of the maTLS record protocol is similar to that of the SplitTLS and mcTLS record protocols. For example, in the intra-region testbed scenario, the data transfer time is 39.92ms, 39.90ms, and 41.28ms in maTLS, SplitTLS, and mcTLS, respectively.

From Figures 7.5a and 7.5b, we conclude that the additional maTLS overhead is mainly due to the setup of a maTLS session, which implies that once the session is established, maTLS provides similar performance to the others while *preserving all security merits that we have discussed.*

Scalability of Three Audit Mechanisms.
Next, we evaluate the scalability of the maTLS audit mechanisms: Explicit Authentication (EA), Security Parameter Verification (SPV), and Valid Modification Checks (VMC). Note that the number of required HMAC operations increases linearly with the number of the middleboxes. Thus we now wish to check the scalability of the HMAC operations in maTLS for its feasibility. To this end, we increase the number of middleboxes in the same data center to quantify the computational overhead due to the audit mechanisms by measuring the validation time for each arriving packet (Figure 7.5c).

We observe that the overhead of the three audit mechanisms is almost negligible. For example, it takes 0.195ms to verify security parameter blocks, 0.203ms to validate certificates, and 0.013ms to check the modification record for two middleboxes. Also, we observe that the overhead increases *linearly* with the number of middleboxes; for each incoming packet, only an extra 0.045ms and 0.063ms overhead is required for the explicit authentication checks and security parameter verification, respectively. It is worth noting that the delay of explicit authentication is mainly due to certificate validation, which accounts for around 95% of the delay. Likewise, signature verification accounts for more than 91% of the delay of the security parameter verification. The overhead for valid modification checks is marginal as it uses HMAC operations to verify the ML, which turns out to be only 0.026ms, even with 8 middleboxes. We believe that the auditing mechanisms of maTLS can achieve their goals without incurring a substantial delay.

CPU Processing Time.
Next, we evaluate the CPU processing time for a maTLS handshake as the number of middleboxes increases. We place all the middleboxes and the endpoints in the same data center to minimize the impact of networking delay. As shown in Figure 7.5d, the

CPU processing time for the maTLS handshake also linearly increases by on average 0.398ms for each middlebox. This increment is mainly due to the multiplication operations required to add an ECDH shared secret, and generating accountability keys using a `PRF`, which account for 0.367ms (92.2% of the increment) and 0.016ms (4.0% of the increment), respectively.

Other Implementation Considerations

Incremental Deployment.

The maTLS protocol can be executed even if not all the entities support it. In other words, a session can have both maTLS segments and TLS segments at the same time. For example, when a client and two middleboxes support maTLS and the server does not, maTLS segments can be set up between the client and the two middleboxes. In this case, the middlebox farthest from the client in the maTLS segments establishes a standard TLS segment with the server. Following the maTLS protocol, all the middleboxes in the maTLS segments send their own certificate to the client. Therefore, the client will receive a bundle of middlebox certificates, but not the certificate including the server's name. This will cause the client to issue a warning message.

To resolve the problem, we require that the farthest middlebox in the maTLS segments should send not only its middlebox certificate but also the received certificate from the standard TLS segment. This allows the client to receive the server's certificate and thus validate it. Unfortunately, this requires that the client must trust that the middlebox sent the certificate that it received, and correctly validated the server certificate in the standard TLS handshake. However, the client can still authenticate the participating middleboxes and verify their security parameters, which is not be supported by the current practice.

Abbreviated Handshake.

maTLS supports abbreviated handshakes using session IDs/tickets in TLS 1.2, or pre-shared keys in TLS 1.3, which need not extend the handshake. A client can resume a maTLS session using the abbreviated handshake protocol. The middlebox (closest to the server) can resume its maTLS segment with the server, as it knows the session ID, pre-shared key, or session ticket. The middlebox, however, does not have the accountability key shared between the client and the server; thus, the server is able to detect incorrect session resumptions by verifying the modification log if an adversary attempts to impersonate the middlebox.

Mutual Authentication.

Like the standard TLS protocol, maTLS also supports mutual authentication by sending a `CertificateRequest` message to the client during the TLS handshake. In this case, the client also sends her certificate upon receipt of the `CertificateRequest` message from the server. The middleboxes can simply append their certificates to her certificates while being forwarded to the server so that both the client and the server authenticate each other's certificates. After that, the client and the server each send a `ExtendedFinished` message to verify the possession of their private keys.

TLS 1.3 Compatibility.
TLS 1.3 [E. 18] has been recently approved and is expected to be widely deployed. The maTLS protocol can support TLS 1.3 by adding a `ExtendedFinished` message after a server's `Finished` message in the server-only authentication mode. The only difference is that TLS 1.2 requires two round-trips for session establishment, while TLS 1.3 only requires one and a half round trips. Unfortunately, this means that individual segments running TLS 1.2 will negate some of the speed-up benefits from TLS 1.3.

Conclusion

This chapter looked at some of the real-world complications involved in the usage of the TLS protocol suite. Because of the complexity of such a protocol, no solution will be intrinsically "simple" – instead, the problem exists in identifying a high-level solution that meets the needs of users.

We also looked at the idea of agreement between two parties who are not directly communicating to each other. This kind of property, although simple to define, can be surprisingly difficult to verify. This is because there are more "points of failure" during transit: especially in the TLS setting, where messages are usually delivered in plaintext, and so security must be verified by tracing backwards on the signed transcript in the final message.

In the next chapter, we will see several strategies towards making the problem of verifying "indirect agreement" tractable.

Chapter 8

Accountable Proxying

The LoRa Alliance represents a collaboration between over 500 companies, producing a set of standards for IoT devices. The most notable of their contributions is the LoRaWAN specification for Long Range devices over a Wide Area Network. Over 80 million connected devices follow the LoRaWAN specification, meaning that its security is of key interest.

In this chapter, we closely analyse the security of the LoRa Join Procedure, in which an End Device uses a Join Server to establish channel-keys used to communicate securely with an Application Server. This protocol is of particular interest because it is a proxied key exchange: the two endpoints never directly communicate. In fact, during the LoRa 1.1 Join procedure, the Application Server is not active at all. Vulnerabilities are highlighted and discussed with both the LoRa 1.02 and 1.1 specifications. Importantly, our analysis considers a variety of threat models under differing assumptions, in order to cover different deployment scenarios or ambiguities in the specification.

This chapter demonstrates the complexities of analysing a real-world protocol at a very fine granularity: models are derived directly from the specification, including details such as counters which can increment and reset. To overcome the complexity of automated analysis, several techniques are deployed, including the use of specialised security goals which capture specific aspects of agreement properties.

The work in this chapter is based on the EuroS&P 2020 paper "Extensive Security Verification of the LoRaWAN Key-Establishment: Insecurities & Patches". Some parts have been reordered, in order to present the case studies as complete sections. The figures and code snippets have been enhanced.
I contributed to this work through manual analysis of the technical specification of the protocol, as well as hand-construction of some of the attack traces which were later verified through Tamarin.

8.1 Introduction to LoRaWAN

In October 2017, the LoRaWAN specification was updated from version 1.02 to version 1.1, containing significant updates to several sections. However, because many IoT devices run specialised hardware, the upgrade path has been very slow, with many devices remaining un-updated.

As such, both the LoRa 1.0 and the LoRa 1.1 specifications remain of interest to many communities. Although certain security concerns regarding the LoRa Join have been raised in academic chapters [AF17], in general, there has not been significant engagement of the LoRa Alliance with the academic, security community. This is surprising, considering the success of such collaborations in recent years, such as the formal modelling of TLS 1.3 [CHH+17] before its official release.

Related Work.
There are several published discussions of vulnerabilities of LoRa 1.0 Join [EBPG19, AF17], using multiple different approaches. On the one hand, one of these [EBPG19] is semi-formal (using symbolic verification), yet it is arguably not a faithful analysis of LoRaWAN 1.0, as their modelling is too simplified. On the other hand, [AF17] performs an empirical analysis of the LoRa 1.0 Join, signalling multiple security issues. If the LoRa 1.0 Join is not widely scrutinised, there are even fewer assessments of the security of LoRa 1.1 Join [BPG18, CF19]. The frameworks used in these works do not always faithfully capture the additional details added in the latest version of the specification. Concretely, neither [BPG18] nor [CF19], fully consider that the 1.1 Join is a four party authenticated key-exchange (AKE) protocol. In the case of [BPG18], not only is it the case that the analysis is treated at the two-party AKE level and that Join Server is not modelled at all, but the choices of modelling abstractions remove important details and their specification files found online are incomplete.

At the other end of the spectrum, Canard et al. [CF19] provide a formal, rigorous cryptographic model. However, they build on the 3ACCE model [BBD+18] (which is for proxied 3-party AKEs), and fail to fully address the fourth party (i.e., the Application Server is not fully modelled). Moreover, the 3ACCE model [BBD+18] is suited for PKI-based AKEs, not for AKEs based on a trusted-server (like the LoRa Join is).

Objectives.
We aim to analyse the AKE protocols in the LoRa 1.0 and 1.1 Join Procedures, as faithfully as possible, considering several threat models which accurately portray the real-world possibilities.

Contributions.
The main contributions of this chapter are as follows:

- We further the symbolic analysis of the LoRaWAN 1.0 AKA protocol, creating automated proofs of the existence of attacks against the specification that previously had to be demonstrated by hand [AF17].

- We present a faithful symbolic analysis, far beyond preexisting efforts of this type [EBPG19, BPG18], of the LoRaWAN 1.1 Join *closely following its specification*, in the threat model declared by the LoRa Alliance.

  We analyse the of LoRa1.1 Join under multiple threat models, considering the possibility that the Network Server (which acts as a proxy) may be partially dishonest.

  To our knowledge, this is the *first full symbolic analysis of the LoRa 1.1 Join*. Using this analysis, we demonstrate several security violations, including those in a model which closely follows that of the specification. Following from this, we discuss the shortcomings, inexactitudes and implications of the current specification for LoRa1.1 Join.

- Drawing on the above, we propose a new design of LoRa 1.1 Join, which we call *LoRaWAN1.1 `LoRA 3-AKA`*$^{+}$. This is as backwards-compatible as possible with the current LoRa 1.1 Join, and we prove using the Tamarin proving tool that `LoRA 3-AKA`$^{+}$ is secure even in our strongest threat model.

Structure.

This chapter is organised as follows. In Section 8.2 the threat model and security goals are introduced. Sections 8.3 and 8.4 contain case studies of the LoRa 1.02 and 1.1 Join procedures, respectively. In Section 8.5 we present `LoRA 3-AKA`$^{+}$, a proposal for a new Join procedure which addresses the vulnerabilities identified. Section 8.6 contains a discussion of our interactions with the LoRa Alliance during the course of this security analysis, as well as our conclusions.

8.2 Threat Model and Security Goals

In this section we introduce the threat models used for our analysis, and use them to define our security goals. The complexity of the LoRa protocol suite means that it is not tractable to fully model traditional synchronisation or agreement goals, and so we use our understanding of the protocol structure to define approximations to these goals that are sufficient to identify attacks on standard authentication goals.

Throughout the rest of this chapter we will refer to the Network Server, Application Server, Join Server and End Device as *NS*, *AS*, *JS*and *ED*, respectively.

Threat Models

We consider three threat models, which we name $\mathcal{M}_{\texttt{AS-NS-Secure}}$, $\mathcal{M}_{\texttt{LoRa1.1Spec}}$, and $\mathcal{M}_{\texttt{NS-weakCorrupt}}$. The intent of these models is to balance, as well as probe, the trust assumptions and the security requirements made by the LoRa specifications. Concretely, we differentiate these models based on the amount of trust placed in the Network Server, as well as the security of channels between *NS* and the backend. Figure 8.1 contains a summary of the different threat models.

		Channel Security	
		High	Low
NS trust	High	$\mathcal{M}_{\texttt{AS-NS-Secure}}$	$\mathcal{M}_{\texttt{LoRa1.1Spec}}$
	Low	–	$\mathcal{M}_{\texttt{NS-weakCorrupt}}$

FIGURE 8.1: Threat models considered in our analysis

The intent of each of these models is as follows:

- $\mathcal{M}_{\texttt{LoRa1.1Spec}}$ is the threat model exactly implied by the LoRa 1.1 specifications for the Join Procedure. In this case, there is a high level of trust on *NS*. More specifically, as per the specs, we consider all parties be honest. As per the specs, we model the the channels between *NS* and *JS* and between *AS* and *JS* as secure, but the channel between *NS* and *AS* as insecure, i.e., as the specs do not require it be secure.

- $\mathcal{M}_{\texttt{AS-NS-Secure}}$ has the same level of trust as $\mathcal{M}_{\texttt{LoRa1.1Spec}}$, i.e., the *NS* and all other parties are considered honest. However, for $\mathcal{M}_{\texttt{AS-NS-Secure}}$, we consider the channel between *NS* and *AS* to be secure. As a result, $\mathcal{M}_{\texttt{AS-NS-Secure}}$ is a weaker threat model than $\mathcal{M}_{\texttt{LoRa1.1Spec}}$. Security analysis in this model is arguably adequate, as a proprietary *AS* might be commissioned with specific security measures (i.e., authentication, confidentiality and integrity) in its communication with the *NS*.

- $\mathcal{M}_{\texttt{NS-weakCorrupt}}$ is a stronger threat model than $\mathcal{M}_{\texttt{LoRa1.1Spec}}$. Concretely, $\mathcal{M}_{\texttt{NS-weakCorrupt}}$ is in fact the same as the $\mathcal{M}_{\texttt{LoRa1.1Spec}}$ model except that the channel between *NS* and *JS* has become insecure. In particular, in $\mathcal{M}_{\texttt{NS-weakCorrupt}}$, the *NS* behaves as per its specification, and the attacker has extra powers only in that it has compromised the security of the channel between the *NS* and the *JS*. Arguably, this is a weak form of compromising the *NS*. That is, the model $\mathcal{M}_{\texttt{NS-weakCorrupt}}$ does not take away this entire trust assumption in the LoRa 1.1 Join (i.e., *NS* does continue to behave follow the protocol); instead, $\mathcal{M}_{\texttt{NS-weakCorrupt}}$ probes the implications of even the smallest compromise of this assumption.

In our formal modelling (in Tamarin), insecure channels are implicitly public and thus accessible to the Dolev-Yao attacker for active manipulation.

So, the $\mathcal{M}_{\texttt{LoRa1.1Spec}}$ model is the baseline (in line with the trust and security of the LoRa specifications), $\mathcal{M}_{\texttt{NS-weakCorrupt}}$ is a weakening of trust assumptions in $\mathcal{M}_{\texttt{LoRa1.1Spec}}$, whereas $\mathcal{M}_{\texttt{AS-NS-Secure}}$ is a strengthening of the secure-setup requirements in $\mathcal{M}_{\texttt{LoRa1.1Spec}}$.

Threat Models for LoRa 1.0. Note that whilst all the above models apply to the LoRa 1.1 Join, they do not apply to the LoRa 1.0 Join. Concretely, $\mathcal{M}_{\texttt{NS-weakCorrupt}}$ rests on the security of the channel between *NS* and *JS*, but this channel is not present in the LoRa 1.0 Join (as *JS* does not exist therein); and in the 1.0 Join, the *AS* is completely inactive, so $\mathcal{M}_{\texttt{LoRa1.1Spec}}$ and $\mathcal{M}_{\texttt{AS-NS-Secure}}$ do not apply.

Properties Analysed

For our verification, we encode various agreement and synchronisation properties [Low97]. Such goals are commonplace for AKE protocols: e.g., weak agreement, non-injective agreement, injective agreement, synchronisation and secrecy of the established key (key secrecy, for short). The main difference is that, for LoRa 1.1, these are extended to encompass not the standard two parties, but rather three or four parties, as demanded by the LoRa Join procedures.

As we give details of our models, which themselves make the properties more specific, we further explain the precise nature of certain formulations of our properties.

8.3 Case Study: LoRaWAN 1.0

In this section we perform a case study of the LoRa 1.0 Join procedure. Although the LoRa specification has been updated to version 1.1, the 1.0 Join is still actively used by many devices. This is partially due to the difficulty in rolling out updates to IoT devices, many of which have specialised hardware.

Although the LoRa 1.0 and 1.1 protocols have fundamentally similar goals, the granularity of our modelling leads to a divergence in our security properties. Notably, the two protocols have different network layouts, with the latter adding the Join Server as a separate entity (in 1.0 the Network Server arguably performs the role of both parties).

Our analysis allows for the first fully formal proof of an attack on the LoRa 1.0 Join procedure, thanks to the detailed modelling of mechanisms such as resetting counters.

The Join Procedure in LoRa 1.0

We begin by explaining the LoRa 1.0 Join procedure protocol. We use as a reference the LoRa 1.0.2 version of the Join Procedure [SLE$^+$]).

The LoRa 1.0 Join considers only three active roles, *End Devices* (*ED*), a *Network Server* (*NS*), and the backend *Application Server* (*AS*). The protocol makes use of a single long-term shared key, denoted AppKey. The Application Server is not active during the Join protocol, instead receiving the derived AppSKey from the Network Server.

In the LoRa 1.0 Join the DevNonce and JoinNonce are sampled uniformly at random over their domain. Yet, DevNonce is only 16-bit long, leading to a birthday paradox-style attack for LoRa 1.0 (i.e., high chance of repetitions over sessions) [AF17].

Figure 8.2 shows the messages exchanged in the LoRaWAN 1.0 Join, which are as follows:

1. The End Device sends a “Join Request” message, consisting of a nonce DevNonce along with identifiers for *ED* and *AS*. The message is MAC-ed using the long term key AppKey.

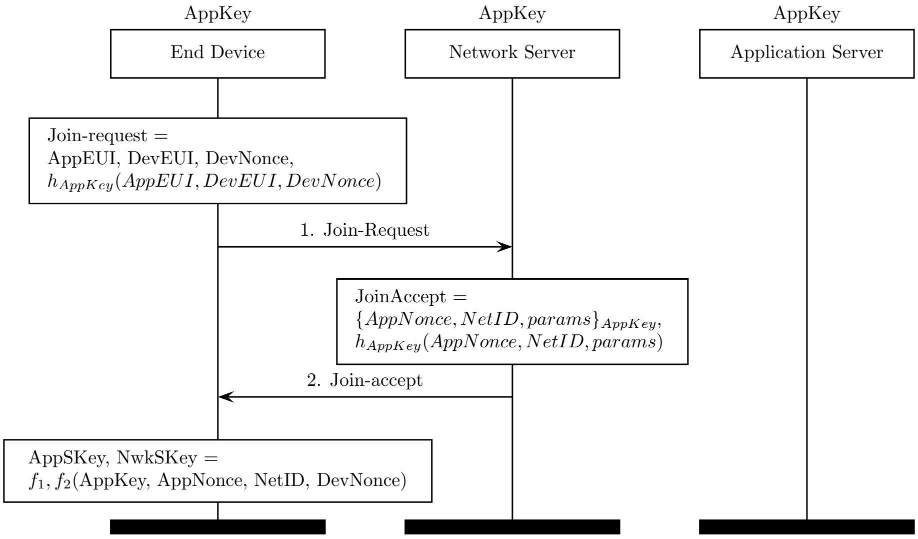

AppKey
End Device
AppKey
Network Server
AppKey
Application Server
Join-request =
AppEUI, DevEUI, DevNonce,
$h_{AppKey}(AppEUI, DevEUI, DevNonce)$
1. Join-Request
JoinAccept =
$\{AppNonce, NetID, params\}_{AppKey}$,
$h_{AppKey}(AppNonce, NetID, params)$
2. Join-accept
AppSKey, NwkSKey =
f_1, f_2(AppKey, AppNonce, NetID, DevNonce)

FIGURE 8.2: LoRa 1.02 Join Procedure

2. After checking the freshness of DevNonce*ED*[1], *NS* replies with a “Join Accept” message. This message contains identifiers for *NS*, a fresh nonce AppNonce, and a collection of parameters to be used for further communication. This message is encrypted using AppKey[2], and a MAC of the plaintext is sent back alongside this encryption.

3. *ED* and *NS* can now calculate the session keys NwkSKey and AppSKey, as AES-128 encryptions under AppKey of AppNonce and DevNonce together with specific tagging (e.g., 0x01, 0x02). To make the figure more readable, we write f_1 and f_2 respectively instead of aes(AppKey; 0x01||...), aes(AppKey; 0x02||...).

Modelling of LoRa 1.0 Join

We first go over our analysis of the LoRa 1.0 Join procedure. Note that for this protocol we consider only the $\mathcal{M}_{\texttt{LoRa1.1Spec}}$ threat model. This is because there is no Join Server, changing the assumptions on secure channels.

We also do not model confusion freeness, as in this case it is subsumed entirely by the synchronisation goal, sync_ED_NS, defined in the following subsection.

Comprehensively & Closely Modelling the LoRa 1.0 Specs in Tamarin.
Our Tamarin model follows the LoRa 1.0 specification extremely closely. For instance, in the code-snippet below, one can immediately see that we encode details down to the level of the padding and optional parameters:

[1] “For each end-device, the network server keeps track of a certain number of DevNonce values used by the end-device in the past, and ignores join requests with any of these DevNonce values from that end-device.” [SLE$^+$]

[2] Technically, the message is *decrypted* using this key, so that the end device needs only implement the primitive for encryption

Example LoRa 1.0 Tamarin Rule

```
rule Device_Receive_JoinAccept:
let
    NS='NetworkServer'
    //inputs - ED_Store_02
    DevEUI=ClientID(~random64)

    //inputs -  ans
    decoded=sdec(ans,AppKey)
    AppNonce=fst(decoded)
    NetID=fst(snd(decoded))
    DevAddr=fst(snd(snd(decoded)))
    opt_params=fst(snd(snd(snd(decoded))))
    tau_s=snd(snd(snd(snd(decoded))))

    //compute session keys
    pad16='pad_with_0s'
    NwkSkey=senc(<'0x01', AppNonce, NetID, DevNonce, pad16>, AppKey)
    AppSkey=senc(<'0x02', AppNonce, NetID, DevNonce, pad16>, AppKey)
```

Nonce Freshness & Short Nonces.
The specification states that *NS* should keep a tally of nonces sent by *ED*, performing necessary freshness checks. Our model fully captures this.

We also simulate the fact that nonces are only over generated small domains ($\mathsf{DevNonce} \in \{0,1\}^{16}$ and $\mathsf{JoinNonce} \in \{0,1\}^{24}$). We devise a manner of producing nonces in a cyclic manner. This clearly under-approximates true nonce generation (which would be done by simply using a "Fresh" sort), however it encapsulates the fact that collisions of nonces are likely. It is arguably unusual for symbolic methods to encode this fact, however LoRa 1.0 insecurities stemming from these collisions are known [AF17]. So, our model aims to see if we can find attacks based on this same short-nonces shortcoming.

Our modelling approach in this case is to encode the set of possible values as constant terms (i.e. '1', '2', ...) rather than abstract fresh terms. This set of values is encoded into the specification of *ED*. When a value is to be chosen, we take advantage of Tamarin's multiset builtin, which permits nondeterministic selection of an element from a multiset.

To verify the soundness of this "small-domain" encoding, we prove two lemmas:
(a) `two_join_requests_distinct_nonces_device`, stating a device can always send two (or more) Join Requests with different nonces. This ensures that our encoding does not restrict the behaviour too coarsely;
(b) `two_join_requests_same_nonce_device`, stating that a device can send two Join Requests with the same nonce, demonstrating that our modelling can indeed capture the shortcomings of short-nonce repetition.

Analysis of LoRa 1.0 Join

We first present the main security properties we encode and verify.

Key-Agreement Properties..
Firstly, we focus on standard agreement properties for AKE protocols, as seminally introduced by Lowe [Low97]. To this end, we encode the following properties with corresponding intuitions:

(1) *weak agreement (*wa_ED_NS*)* – whenever a device *ED* has completed the Join Procedure with a network server *NS*, then some execution of *NS* has "recorded" to have run the Join with said *ED*;

(2) *non-injective agreement (*nia_ED_NS*)* – whenever a device *ED* has completed the Join Procedure with a Network Server *NS*, and *ED*'s transcript contains certain messages, then some execution of *NS* has "recorded" to have run the Join with said *ED* with the same said messages;

(3) *injective agreement (*ia_ED_NS*)* – whenever a device *ED* has completed an execution d of the Join Procedure with a network server *NS* and *ED*'s transcript for execution d contains certain messages, then there is a unique execution n of *NS*, which records a Join Procedure with said *ED*, and the same said messages that appear in *ED*'s execution.

(4) *synchronisation (*sync_ED_NS*)* – in any full Join execution there exists an *ED* and a *NS* who both have matching views of the transcript of the execution.

(5) *session-key secrecy (*key_secrecy*)* – if a secret key x is established at timepoint i, then either the adversary does not know x, or the agent who established said key has been compromised beforehand.

Freshness Failure.
The main specification encoded for checking freshness and identifying potential replay attacks is the following fifth property:

(6) *replay-attack existence (*replay*)* – there can exist two Join executions, i and j, in which $\mathsf{DevNonce}_i = \mathsf{DevNonce}_j$ and $\mathsf{AppNonce}_i = \mathsf{AppNonce}_j$ (i.e., in which both *ED*'s nonce and *NS*' nonce repeat themselves).

Analysis Results for LoRa 1.0..
We now present some details and the results of our analysis for the properties above, performed using the Tamarin prover tool. To carry out the verification, we created an oracle to improve the automatic navigation of the search-space.

The results show that 4 out of the 6 properties examined for agreement and synchronisation in LoRa 1.0 Join are violated. We provide a summary in Table 8.1.

Interpretation of Attacks..
We note that the failings of of non-injective agreement and synchronisation come

Security Goal	$\mathcal{M}_{\texttt{LoRaSpec}}$
key_secrecy	✓
sync_ED_NS	✗
wa_ED_NS	✓
nia_ED_NS	✗
ia_ED_NS	✗
replay resistance	✗

TABLE 8.1: LoRa 1.0 Results

down to equivalent traces. This trace shows that the main crux of the failure is that DevNonce is not used in the LoRa 1.0 Join Response. We note that this behaviour is rectified in the LoRa 1.1 Join.

The fact that replay-attack resistance fails indicates that even with the checks of the *NS* on the freshness of DevNonce, due to the small-domain of the nonces, the protocol does fail to catch replay attacks. Namely, the trace shows that it is possible for there to be a session i in which the attacker can replay an old DevNonce from a session j and, in session i, the *NS'* AppNonce also coincides with the AppNonce in session j. As a result, the AppSKey established in each of two traces is the same. Given how the encryption of the record-layer messages works in LoRa 1.0, this means that the attacker can retrieve elements of the plaintext of the record-layer messages from the two sessions i and j. Note that the replay attack we exhibit by this was shown by hand in 2018 [AF17]; however, *this is the first time this replay was automatically found by a protocol-verification tool.*

As well as verifying the soundness of our "small-domain" nonce approach, we also verified that this modelling is complete - that the attacker cannot always perform this replay attack. To this end, we prove two additional lemmas, which demonstrate that there are also traces in which only one nonce (be it the DevNonce or the AppNonce) repeats itself.

8.4 Case Study: LoRaWAN 1.1

We now investigate the LoRa 1.1 Join procedure. The 1.1 version of the specification addresses some problems with LoRa 1.0, by reducing trust in the proxy. In accordance with this, our analysis is extended to include this additional party. Again, we aim to draw our model in fine-grained detail from the specification, which allows us to produce attacks that are directly verifiable.

The Join Procedure in LoRaWAN 1.1

Protocol Description. The LoRaWAN 1.1 "Join procedure" is a four-party protocol [Sor], involving an *end device ED*, a *Join Server JS*, a *Network Server NS* and an *Application Server AS*. The protocol relies primarily on the fact that *ED* and *JS* share

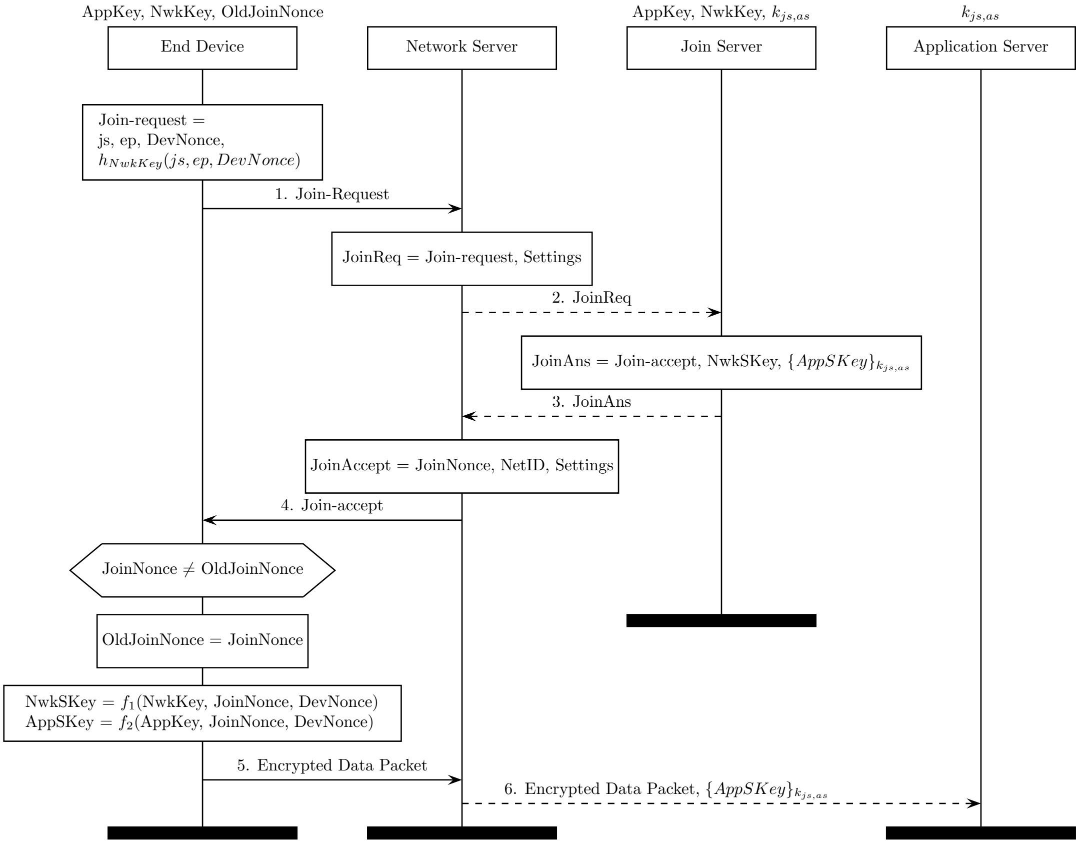

AppKey, NwkKey, OldJoinNonce
End Device
Network Server
AppKey, NwkKey, $k_{js,as}$
Join Server
$k_{js,as}$
Application Server
Join-request = js, ep, DevNonce, $h_{NwkKey}(js, ep, DevNonce)$
1. Join-Request
JoinReq = Join-request, Settings
2. JoinReq
JoinAns = Join-accept, NwkSKey, $\{AppSKey\}_{k_{js,as}}$
3. JoinAns
JoinAccept = JoinNonce, NetID, Settings
4. Join-accept
JoinNonce ≠ OldJoinNonce
OldJoinNonce = JoinNonce
NwkSKey = f_1(NwkKey, JoinNonce, DevNonce)
AppSKey = f_2(AppKey, JoinNonce, DevNonce)
5. Encrypted Data Packet
6. Encrypted Data Packet, $\{AppSKey\}_{k_{js,as}}$

FIGURE 8.3: LoRa 1.1 Join Procedure – Simplified View. Secure channels are displayed with dashed lines - - →

two[3] long-term symmetric keys: AppKey and NwkKey. The aim of the protocol is to use these long-term keys to establish a set of session keys for use in the record layer.

In Fig. 8.3, we depict the Join Procedure in LoRaWan 1.1, where –for simplicity– we omit certain low-level details, such as certain identifiers, message padding, and headers. The protocol proceeds as follows:

1. *ED* sends to *JS* a "Join Request" formed of a nonce DevNonce, relevant identifiers, along with a MAC of this information using the long-term key NwkKey.

2. This request is received by *NS*, who forwards it as is (apart from network-level headers) to *JS*.

3. Upon successful verification of the MAC, *JS* produces a "Join Answer", which includes a "Join Accept", and a number of session keys. The crucial session keys are AppSKey and NwkSKey. We will discuss these keys' generation later.

   – To create the "Join Accept", *JS* generates a new nonce JoinNonce, which it MACs and encrypts using the long-term keys shared with *ED*.

   – In the "Join Answer", the AppSKey is encrypted with a long-term key that *JS* shares with *AS*.

   – The "Join Answer" is sent by *JS* to *NS*.

[3] There is a third, long-term shared key called JSIntKey.

4. The Network Server retains NwkSKey and forwards only the "Join Accept" part. From the "Join Accept", the device retrieves JoinNonce, and is then able to re-compute the session keys.

5–6) Later, in the first encrypted application-level message sent by *ED*, the *AS* either gets the encrypted AppSKey from the *NS* or a session-id with which to obtain the AppSKey from the *JS*. The *NS* forwards this information along with the first application-level message. This will be further discussed later.

Keys Generated During the Join Procedure.
During the Join Procedure, the Join Server (and later, the End Device) generates a set of session keys. The most significant of these are named AppSKey and NwkSKey[4]. These sessions keys are produced by encrypting with AES-128 using the long-term keys AppKey and NwkKey, respectively. The following data is used in the key generation: DevNonce, JoinNonce and *JS*'s identifier JoinEUI, along with specific message tagging[5](e.g., 0x02, 0x04, respectively). Each session key is used at the record layer for encrypting/decrypting application messages[6]: NwkSKey is to be used by *NS* and *ED*, and AppSKey by *AS* and *ED*.

Remarks on the LoRa 1.1. Join Procedure.

There are a few other aspects of the LoRa 1.1 Join Procedure worth mentioning, which we detail below.

LoRa 1.1 Channel Requirements.
During the Join Procedure, as per the LoRa specifications, the security of channels is assumed to be as follows:

- an insecure channel between *ED* and *NS*
- an insecure channel between *NS* and *AS*
- a secure channel between *NS* and *JS*

Upon the end of this AKE, the intended channels are as follows:

- a confidential, authenticated and integral channel between *ED* and *NS* (obtained via encryption and MAC-ings);
- an insecure channel[7] between *NS* and *AS*, on which a correct behaviour is one where *NS* is proxying AppSKey-encrypted messages between *ED* and *AS*.

[4] There are two additional MAC keys for *NS* and *ED*, namely, SNwkSIntKey and FNwkSIntKey. Our formal models capture all session keys.

[5] To not overload notation with different AES keys and different padding, on Fig. 8.3 , we simply write NwkSKey is calculated using a function f_1 and AppSKey is calculated using a function f_2.

[6] LoRa record-layer messages are not encrypted directly with the AppSKey, instead there is a bespoke KDF which uses AppSKey.

[7] Concretely, line 1437 of the LoRA1.1 specifications mentions that there is a-priori no secure channel between *NS* and *AS*.

There is also a secure channel between *AS* and *JS*, which can optionally be used to deliver AppSKey. Note the insecure channels between *NS* and *AS*, both during the Join Procedure as well as afterwards, at the application level. In the LoRa1.1 specifications, there is no requirement of integrity or of confidentiality made with respect to the channel between *NS* and *AS*.

Bespoke Nonce-generation in LoRa 1.1.
In LoRa 1.1, DevNonce is not sampled uniformly at random over its domain, instead it is produced as a strictly-increasing counter. *NS* keeps a tally of DevNonces per device, and checks that it is increasing (see page 52 of the specifications). Similarly, the JoinNonce (which is 24 bits long) is produced by the increasing a counter, kept up-to-date for each device.

Alternative AppSKey-deliveries in LoRa 1.1.
There are in fact two ways in which AppSKey can be delivered to *AS*. In the Join Answer message, *JS* may choose to generate a SessionKeyID instead of (or as well as) the encrypted AppSKey. If a SessionKeyID is given, the *NS* may forward this alongside any encrypted data packets sent by *ED*. The specification does not indicate which approach is preferred.

In this second case, *AS* cannot immediately decrypt messages sent from *ED*. In order to retrieve the encryption key, *AS* contacts *JS*, who delivers the corresponding AppSKey. In both cases, it is not clear from the specifications whether *AS* is to check if the first data-packet/application-level message does indeed decrypt correctly with the delivered AppSKey and how the protocol continues in case of failure.

LoRa 1.1 Join – a Non-standard AKE..
Although the Join Procedure views four entities, the protocol is in fact *actively* run just between two of the four parties: the end-device *ED* and the Join Server *JS*. In particular, *ED* and *JS* have pre-shared symmetric-keys, and –based on these– they establish new session keys, in what constitutes a symmetric-key authenticated key-establishment (AKE). Using the the Join Server *JS* as a trusted third party (TTP) in an symmetric-key AKE protocol is not necessarily unusual; this type of AKE has been used in protocols such as Kerberos [Ada11]. To this end, like in a TTP-based AKE, *ED* and *JS* establish session keys not for their use, as it would be the case in a standard AKE, but for the use of *ED* with the Network Server and the Application Server, respectively. However, there are other aspects that are non-standard, even for a TTP-based AKE protocol:

– *ED* and *JS* do not have a direct connection, as in a standard TTP-based AKE, and instead it is the Network Server who acts as a proxy for the AKE messages sent between *ED* and *JS*.

– Even though, as a result of this AKE, the Application Server receives a session key to use with *ED* to encrypt/decrypt application-level messages, the Application Server is not active during the Join procedure at all. In fact, *AS* cannot verify any meaningful properties (e.g., integrity w.r.t. what *JS* delivered) about the session key it is given.

Modelling of LoRa 1.1 Join

Comprehensively & Closely Modelling the LoRa 1.1 Specs in Tamarin.
We follow the specifications of the LoRa 1.1 Join closely, encoding in Tamarin levels of details far beyond the simplified description given in Figure 8.3. For instance, the Tamarin code-snippet shown in Figure 8.4 shows part of the specification of the JoinServer, which contains details about sessions keys, padding, and counters.

LoRa 1.1 Code Snippet

```
rule JoinServer_Receive_JoinRequest_Generate_Response_with_Key:
let
...
    //verify tau_c
    tau_c_dash=<'MHDR', $JoinEUI, $DevEUI, DevNonce>

    //generate response - assuming OptNeg is set, ie we are using v1.1
    ctr_JS=ctr_JS_in+'2' //counter for nonces
    JoinNonce=Nonce(<$DevEUI, ctr_JS>) //nonce  device specific;
     ↪  counter-based
    DevAddr=Nonce(~rnd32DevAddr) //address assigned by the Join Server

    //compute various keys
    pad16='pad_with_0s'
    //the key used for the mac during the initial Join-Accept answer
    JSIntKey=SessionKey(senc(<'0x06', $DevEUI, pad16>, NwkKey))
    //the key used for the Join-Accept triggered by a Rejoin-Request
    //JSEncKey=senc(<'0x05', DevEUI, pad16>, NwkKey) not needed atm

    //compute various network session keys
    //Forwarding Network session integrity key (p55, l. 1604)
    FNwkSIntKey=SessionKey(senc(<'0x01',JoinNonce, $JoinEUI,DevNonce,
     ↪  pad16>, NwkKey))
    //Serving Network session integrity key (p55, l. 1605)
    SNwkSIntKey=SessionKey(senc(<'0x03',JoinNonce, $JoinEUI,DevNonce,
     ↪  pad16>, NwkKey))
    //Network session encryption key (pp55, l. 1606)
    NwkSEncKey=SessionKey(senc(<'0x04',JoinNonce, $JoinEUI,DevNonce,
     ↪  pad16>, NwkKey))
    //compute the secret Application Server Key (pp55, l. 1601)
    AppSKey=SessionKey(senc(<'0x02', JoinNonce, $JoinEUI, DevNonce,
     ↪  pad16>, AppKey))
...
```

FIGURE 8.4: Code Snippet of Tamarin rule for LoRa 1.1 Join Procedure

In order to justify the choices made in our modelling, our Tamarin files contain references to the line/page of the LoRa specifications which justify our choice of encoding.

Modelling the Join & Data Packets in the LoRa 1.1..
The LoRa 1.1 Join Procedure finishes before the first data packet is sent (i.e., the keys

are established before this packet is sent). However, as explained in Subsection 8.2, we wish to verify key-agreement properties with respect to all four parties involved, but *AS* receives the AppSKey only when first data packet is delivered (or shortly afterwards, depending on the delivery method). To this end, we model the full Join Procedure, plus the first data packet.

Variations of the LoRa 1.1 Specifications Modelled.
The specification contains no clear requirement on the *AS* being able to decrypt the first data-packet sent to it. To this end, we model two variations of the LoRa 1.1 Join:

- The *"Desync-Model"*, which encodes the LoRa 1.1 specification precisely (where the *AS* will not end in "error" if it receives the first application message malformed);
- The *"Sync-Model"*, which encodes a slight tightening of the LoRa 1.1 specifications, whereby the *AS* will end in "error" if it receives a malformed first application message.

For each such "sync"/"desync" setting, we encode the two cases discussed in Subsection 8.4 as to how *AS* receives the AppSKey. This yields:

- *"AppSKey-from-NS"*, which encodes that the *NS* delivers the (JS-encrypted) AppSKey directly to the *AS*, with the first application-level message;
- *"AppSKey-from-JS"*, which encodes that the *JS* delivers the AppSKey to the *AS*, upon the latter's request yielded after the first application-level message arrived at the *AS*.

Encoding Different Threat Models.
For both the Desync-Model and the Sync-Model, as well as for both the Appskey-from-NS and Appskey-from-JS variations, we create four different Tamarin files, each encapsulating one of the three threat models $\mathcal{M}_{\texttt{LoRa1.1Spec}}$, $\mathcal{M}_{\texttt{NS-weakCorrupt}}$ and $\mathcal{M}_{\texttt{AS-NS-Secure}}$, presented in Subsection 8.2.

Consequently, to capture all these different scenarios, we include a total of 12 Tamarin files for LoRa 1.1. Note that this is mainly done for convenience, with the vast majority of the Tamarin code shared between models, and only minor changes in key locations. This separation allows us to pinpoint which "configurations" of LoRa 1.1 lead to attacks.

Restrictions.
In order to improve the tractability of our analysis in the most severe threat model, $\mathcal{M}_{\texttt{NS-weakCorrupt}}$, we include a series of restrictions on the maximum number of Join Requests and the maximum number of Join Responses. This means that, in this strongest model and for this property, we look to see if there is any attack within a maximum number of spanned sessions for one given Device and one given Join Server. However, we do not bound the attacker's powers in other ways, such as number of nonces or messages. Concretely, for the LoRa 1.1 Join in the $\mathcal{M}_{\texttt{NS-weakCorrupt}}$ model,

proofs were carried out with a maximum number of Join Responses and Requests set to 4 and 3 respectively. These values were chosen with the tradeoff between computational cost and faithfulness of the model in mind. However, for exhibiting an attack, we found counterexamples of the properties in $\mathcal{M}_{\texttt{NS-weakCorrupt}}$ even with this set to 2.

Weakly-typed Models vs. Source Lemmas.
Specifically for the case of proving properties on the $\mathcal{M}_{\texttt{NS-weakCorrupt}}$ models, we wrote several Tamarin source lemmas. However, these proved ineffective. Instead, we achieved better performance by weakly-typing[8] the models. Nonetheless, we did this in a parsimonious way, i.e., we only declared: (a) a specific sort Nonce as a subsort of Fresh, to exclude typing-attacks on nonces; (b) a specific format for data-packages after the Join finished, to stop injection of *any* possible term therein. Arguably, these two weak-typings are realistic and cannot exclude mainstream key-agreement attacks. These together with the aforementioned restrictions allowed us to disprove (different versions) of confusion freeness in the strongest attacker model $\mathcal{M}_{\texttt{NS-weakCorrupt}}$.

Counter-based Nonces.
As described in Subsection 8.4, the LoRa 1.1 specifications replace nonces with counter-based bitstrings both on the *ED* and on the *JS*, which the latter having device-specific counters. We model this in Tamarin, using the multiset builtin. Counters are modelled as multisets over a domain of a single element (i.e. the counter can have values '1', '1'+'1'...), incrementing on each use. Both *ED* and *JS* record the last accepted value of their partner's counters, and will only accept a message if the counter is increasing.

Security Goals of LoRa 1.1 Join

We now present the main security properties we encode and verify.

Firstly, to encode equivalent properties as in the LoRa 1.0 Join, we look at agreement properties between *ED* and *JS*. To this end, recall that the role of *NS* in the LoRa 1.0 Join is assumed in LoRa 1.1 by *JS*. So, with the equivalent meaning as those presented in Subsection 8.3, we encode:

(1) *key secrecy for the* NwkSKey *and* AppSKey *(*key_secrecy*)*

(2) *weak agreement between ED and JS (*wa_ED_JS*)*

(3) *non-injective agreement between ED and JS (*nia_ED_JS*)*

(4) *injective agreement between ED and JS (*nia_ED_JS*)*

Secondly, we move to encoding AKE requirements linked specifically to the LoRa 1.1 Join, i.e., between the Devices and the Application Server. We specify the following lemmas:

[8] Such techniques have been used before in Tamarin-verification when large systems (such as TLS1.3) were encoded and analysed [CHH+17].

(4) *weak agreement between ED and AS (*wa_ED_AS*)* – which encodes that whenever a device *ED* has completed the LoRa 1.1 Join Procedure allegedly with *AS* then some execution of *AS* has "recorded" running with *ED*.

We note, once again, that (in our models and in the LoRa 1.1 specification), an *AS* "ascertains" that it is communicating with a specific *ED* only when said *AS* receives the first (encrypted) data packet and the corresponding AppSKey to decrypt it, as –beforehand– the *AS* is not alive during the actual key-establishment run by the *ED* and *JS* (via *NS*). Recall that, to capture error-handling if the packet-decoding by the *AS* fails, we encode the two variations "`Desync-Model`" and "`Sync-Model`" of the LoRa specifications. In both of these models, we look at the synchronisation sync_ED_AS property below:

(5) *synchronisation between ED and AS (*sync_ED_AS*)* – which checks if there is an execution in which *ED* sends a data-packet, yet the data-packet received by *AS* cannot be decoded.

Moreover, we also encode agreement properties similar to sync_ED_AS but slightly stronger. Namely, we check 2-party and 3-party agreement properties with respect to data packets and/or AppSKeys, to check not only if the *AS* is (un)able to decode a data-packet (as per sync_ED_AS), but also if an *AS* could potentially be adversarially confused as to which data-packet and/or AppSKey is associated with which *ED*. We call these agreement properties ***confusion freeness*** and describe several of their flavours below.

(6) *two-party confusion-freeness w.r.t. packets* (CF_ED_AS_packet) – which encodes that if a device *ED* finishes a session *i* by sending an encrypted data-packet p_i to an application server *AS*, then if the application server *AS* gets this data-packet p_i it believes it originates from this device *ED*.

(7) *two-party confusion-freeness w.r.t.* AppSKey (CF_ED_AS_packet_key) – which encodes the same as CF_ED_AS_packet w.r.t. the AppSKey: i.e., if a device *ED* finishes a session *i* with a given $\mathsf{AppSKey}_i$, then if the application server *AS* receives $\mathsf{AppSKey}_i$, it correctly believes it is meant for communicating with *ED*.

(8) *three-party confusion-freeness w.r.t.* AppSKey (CF_ED_JS_AS_key) – which encodes the same as CF_ED_AS_packet_key, factoring in the *JS* as well: i.e., the view on a given AppSKey is synchronised correctly between a given *ED*s, the *JS* and the *AS*.

As with LoRa 1.0, we analyse additional lemmas, primarily to check the correctness of our modelling, or for auxiliary security properties (e.g., that the key-setup/commissioning we model between Devices and the Join Server, is secure). However, as they are not pertinent to the analysis, we omit details on those here.

Last but not least, we recall that we run our verification for these properties on both the "`Desync-Model`" and "`Sync-Model`" of the specifications, in each of the three threat models $\mathcal{M}_{\texttt{LoRa1.1Spec}}$, $\mathcal{M}_{\texttt{NS-weakCorrupt}}$, $\mathcal{M}_{\texttt{AS-NS-Secure}}$, with two different methods for distributing AppSKey. To this end, we created various bespoke oracles for the respective analyses.

Hierarchy of Properties Analysed

$$\begin{tikzcd}
\text{In the ``}\texttt{Sync Model}\text{'':} & \text{CF\_ED\_JS\_AS\_key} \arrow[r] & \text{CF\_ED\_AS\_packet\_key} \arrow[r] & \text{CF\_ED\_AS\_packet} \arrow[r] & \text{sync\_ED\_AS} \\
\text{In the ``}\texttt{Desync Model}\text{'':} & & \text{CF\_ED\_AS\_packet\_key} \arrow[r] \arrow[u] & \text{CF\_ED\_AS\_packet} \arrow[r] \arrow[u] & \text{sync\_ED\_AS}
\end{tikzcd}$$

FIGURE 8.5: Hierarchy of Confusion-Freeness and Synchronization Goals

The relationships between our various security properties are as follows:

Agreement Properties.
Our agreement properties are directly drawn from the definitions of Lowe [Low97]. Although the protocols we analyse are not two-party, the agreement results work on a pairwise basis. Hence we have that injective agreement is stronger than non-injective agreement, which in turn is stronger than weak agreement.

Confusion-Freeness Properties.
The simplest of the confusion-freeness properties is CF_ED_AS_packet. CF_ED_AS_packet is stronger than sync_ED_AS, as it looks not just at the *AS* decoding the packets, but also the *AS* attributing the device address inside said packets to a device. This process is tied in with the format of the application-layer messages per se and (also) with how the device address (that appears in these messages) is fed[9] to the *AS*. In other words, analysing CF_ED_AS_packet also looks at potential confusion of device identities.

Also, in this vein, CF_ED_AS_packet is weaker than CF_ED_AS_packet_key, as the latter requires agreement on AppSKey as well as the packets sent to be associated with that AppSKey. In fact, note that the confusion-freeness properties (CF_ED_AS_packet_key, CF_ED_JS_AS_key,) that require agreement on both the AppSKeyas well as the data-packet, logically, are not truly needed in the "`Desync-Model`": if the *AS* is able to report failure when decoding the packet, it is less relevant if it receives an incorrect AppSKey. As such, we check the CF_ED_AS_packet_key, CF_ED_JS_AS_keyonly in the "`Sync-Model`".

Some of the confusion-freeness properties require three parties to agree, compared to the simpler case of just two parties. Clearly, the former are stronger requirements than the latter. Not only that, but given the modelling, if the 3 parties at hand agree on an AppSKey (i.e., CF_ED_JS_AS_key holds), then necessarily the *AS* and *ED* would agree on the packet as well (i.e., CF_ED_AS_packet_key holds); this is because the first agreement over 3 parties also binding over identification of the device (*ED*'s EUI and *ED*'s address), in a way that links in with the packets sent and received by the *ED*and the *AS*, respectively.

[9] For details, see e.g., pages 12, 18, 19 of the backend specifications.

Other Relationships.

Although not necessarily true in the universal setting, an attack on key secrecy for the LoRa Join protocol immediately results in a scenario where the adversary can forge data packets (and the associated MACs). Such an attack would thus entail violations of the majority of the other properties checked.

We note that the "`Desync Model`" encodes a weaker system than the "`Sync Model`". So, for the formulae that make sense to be checked in both (e.g., CF_ED_AS_packet and sync_ED_AS), we have an implication from "`Desync`"-based formulations to "`Sync` "-based formulations.

Unlike in the case of "`Desync`" vs. "`Sync`", hierarchical lines cannot be drawn in between properties holding in the variations of the models w.r.t. AppSKey deliveries: that is, the " `AppSKey-from-NS`" and "*`AppSKey-from-JS`*" are incomparable with respect to these properties holding:

$$\text{prop. in ``}\texttt{AppSKey-from-NS}\text{''} \overset{\not\rightarrow}{\not\leftarrow} \text{prop. in ``}\texttt{AppSKey-from-JS}\text{''}$$

Finally, if any property holds in the $\mathcal{M}_{\texttt{NS-weakCorrupt}}$ model, then it will hold in the $\mathcal{M}_{\texttt{LoRa1.1Spec}}$ model. If any property holds in the $\mathcal{M}_{\texttt{LoRa1.1Spec}}$ model, then it will hold in the $\mathcal{M}_{\texttt{AS-NS-Secure}}$ model:

$\mathcal{M}_{\texttt{NS-weakCorrupt}} \Rightarrow \mathcal{M}_{\texttt{LoRa1.1Spec}} \Rightarrow \mathcal{M}_{\texttt{AS-NS-Secure}}$

In Figure 8.5 we present the part of our hierarchy of properties that does not trivially follow from Lowe's agreement lattice, but is rather dictated by our threat and system models.

Analysis Results

We verified the above properties in all our threat models introduced in Subsection 8.2: $\mathcal{M}_{\texttt{AS-NS-Secure}}$, $\mathcal{M}_{\texttt{LoRa1.1Spec}}$, and $\mathcal{M}_{\texttt{NS-weakCorrupt}}$.

We first report on the analysis of *ED-JS* properties. Given the presence of a secure channel between the *NS* and *JS* in all but the $\mathcal{M}_{\texttt{NS-weakCorrupt}}$ model, and the implicit assumption that *ED* is honest in these security claims, these properties hold in the $\mathcal{M}_{\texttt{LoRa1.1Spec}}$ and $\mathcal{M}_{\texttt{AS-NS-Secure}}$ cases. However, these properties do not hold in the $\mathcal{M}_{\texttt{NS-weakCorrupt}}$ threat model, when the adversary is able to manipulate the (mostly plaintext) messages on this channel.

Table 8.2 shows the results of our analysis on the 1.1 Join procedure. A "NM" entry indicates that the property was not modelled. This is because their validity is immediately deducible from the results of other properties, following from our hierarchy of goals. An entry of "NT" indicates that the proof did not terminate given our current oracles. It is possible that with re-working of the associated oracle these would terminate, but this is uncertain due to the underlying undecidability of the verification problem at hand.

We begin with the scenario which most closely follows the LoRaWAN specification.

Threat Model	$\mathcal{M}_{\texttt{AS-NS-Secure}}$				$\mathcal{M}_{\texttt{LoRa1.1Spec}}$				$\mathcal{M}_{\texttt{NS-weakCorrupt}}$			
Synchronisation	Sync		Desync		Sync		Desync		Sync		Desync	
Key Delivery	NS	JS	NS	JS	NS	JS	NS	JS	NS	JS	NS	JS
Security Goal												
key_secrecy	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
wa_ED_JS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
wa_ED_AS	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗
nia_ED_JS	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗
sync_ED_AS	✓	✓	✓	✓	✓	✓	✗	✗	NM	NM	✗	✗
CF_ED_AS_packet_key	✓	✓	✓	✓	NT	NT	✗	✗	✗	✗	✓	✓

TABLE 8.2: Main Verification Results for the LoRa 1.1 Join

This is the $\mathcal{M}_{\texttt{LoRa1.1Spec}}$ threat model, using the `Desync-Model` encoding. This model is weaker than the `Sync-Model`, as the *AS* accepting potentially malformed messages enables some attacks.

In this setting, our analysis shows that sync_ED_AS fails. Further, confusion-freeness CF_ED_AS_packet_key fails, even in the main variant of the protocol where the *AS* obtains the AppSKey from the NS. In this case, our falsifying traces shows that the attacker can cross-wire device addresses and the corresponding AppSKeys as well inject wrong application data-packets.

In the case of the $\mathcal{M}_{\texttt{NS-weakCorrupt}}$ model, the aforementioned properties (sync_ED_AS, CF_ED_AS_packet_key) continue to fail. Moreover, a stronger 3-party variation of the confusion-freeness property, CF_ED_JS_AS_key, fails (also due to the hierarchical implication). More specifically, CF_ED_JS_AS_key fails in the $\mathcal{M}_{\texttt{NS-weakCorrupt}}$ model because the attacker can confuse not only the *AS*, but also the *JS* as to which devices sent which JoinRequests or messages. Concretely, the trace found here shows that an attacker who controls the channel between *NS* and *JS* never forwards anything to the *JS*, and swaps two devices' requests on the *NS-AS* channel.

Implications of the Attacks Found.

The trace corresponding to CF_ED_AS_packet_key failing in the `Desync Model` shows that *AS* can be (adversarially) confused when considering from which devices application-layer messages have come. As a result, *AS* may be ultimately unable to decode a message from some ED_1, because they have received some AppSKey$_2$ to link to ED_1 when this key instead corresponds to some ED_2.

It is not clear how the *AS* would behave in this case, or if it would store the wrong address for ED_1 for a period of time.

With sync_ED_AS failing, and with the specs not being clear on how the *AS* needs to respond, it also means that an *ED* may well believe it is sending LoRa data to an *AS*, when it actually is not. In fact, a malicious party in between the *NS* and *AS* gets hold of this (encrypted) data. This creates not only data loss for the *ED*'s owner, but also data-collection by malicious parties, with no hope that the *AS* would track or signal any malfunction.

Countermeasures.
To stop the failure of sync_ED_AS and CF_ED_AS_packet_key, we first and foremost recommend that the LoRa 1.1 explicitly require that the *AS* check the format of the first data-packet served and end in "error" if the decryption of this eventually fails. Moreover, it should not store any long-term data with respect to this failed message.

Furthermore, we observe that the AKE properties that fail in the $\mathcal{M}_{\texttt{LoRa1.1Spec}}$ model hold in the model $\mathcal{M}_{\texttt{AS-NS-Secure}}$. In practice, this means that strengthening the LoRa specification's requirements to have a secure channel between the *NS* and the *AS* can fix the aforementioned AKE/LoRa 1.1 failing. However, the security of this channel may often fall under the management of the *NS* and *AS*. This means that even if the LoRa specifications were to require that these channels be secure, we cannot be sure that this would be attained in practice.

Our analysis in the $\mathcal{M}_{\texttt{NS-weakCorrupt}}$ setting tells us that any loosening of the trust in *NS* results in several key-agreement properties failing. Notably, because confusion freeness fails, it means that we cannot guarantee that an *AS* will have a correct view as to which device it is speaking to. At the application-level, this is a significant attack.

Due to the aforementioned issues with security ownership and risk, a more costly countermeasure that may prevent the failure of sync_ED_AS and CF_ED_AS_packet_key is the possible introduction of a clear key-confirmation step between the *AS* and *ED*. Or, even further, one can consider making the *AS* part of the Join and not having the AppSKey delivered "blindly" to it alongside the first data-packet

Overall, it maybe be advisable that the Join Procedure be redesigned such that the *AS* cannot be confused irrespective of the security of these channels and trust in *NS*. However, it is important that any modifications should remain as close as possible to the current Join design.

8.5 A Proposal for a Novel Join Procedure

We now propose an amended version of the LoRa 1.1 Join, which we call *LoRA 3-AKA*$^{+}$. This new design is rooted in the observations from the case studies.

In order to prevent the *AS* from becoming confused as to which device it is communicating with, the *NS* provisioner must ensure the security of the incoming and outgoing channels. However, it would be interesting if we could attain the same level of security by instead placing less trust in the messages forwarded by the Network Server. Indeed, it would be ideal if we could propose a design that is secure even if these NS-originating channels are untrusted or compromised. To this end, we leverage that –unlike in the LoRa 1.1 Join– the *AS* could be an active part of the Join. Precisely, in LoRA 3-AKA^{+}, we augment the LoRa 1.1 Join such that the *AS* is minimally active. Arguably, this is as backwards-compatible as possible with the current LoRa 1.1 Join.

So, at the cost of a small increase in latency, we attain the LoRA 3-AKA^{+} Join, which is a secure multi-party AKE , even if the channels out of the *NS* are compromised.

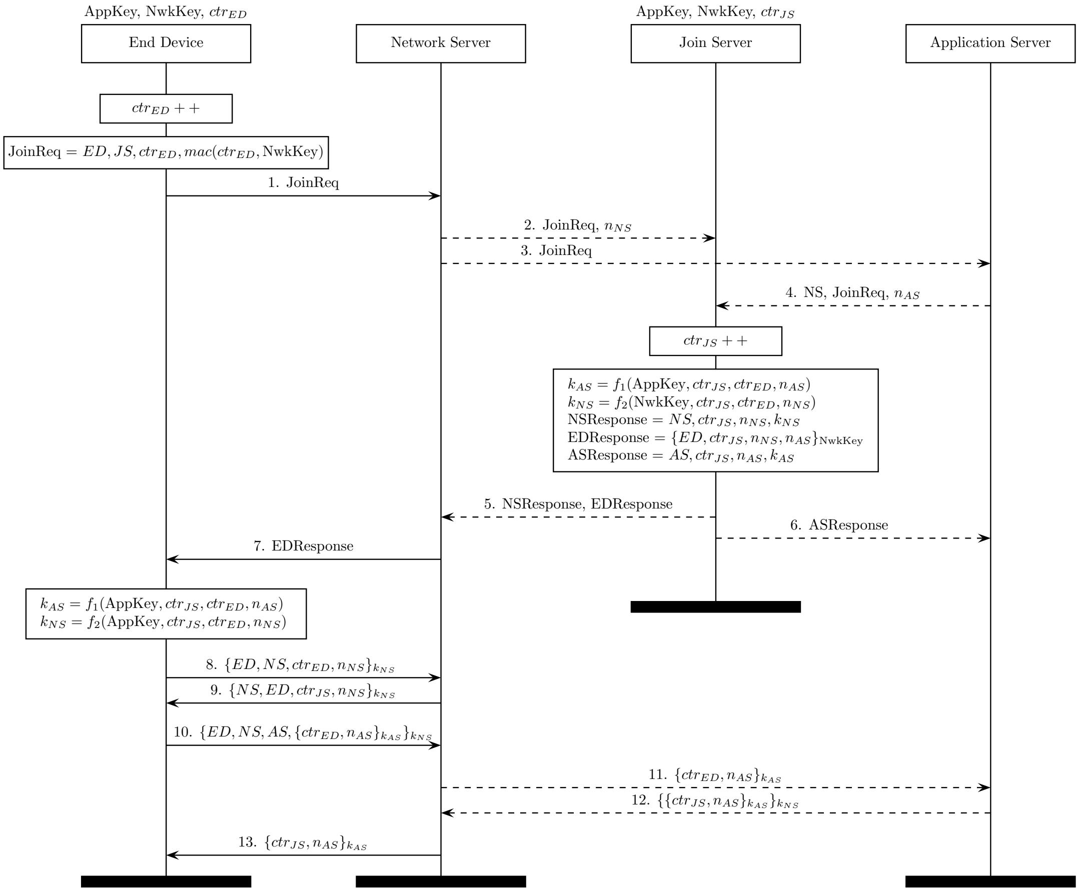

AppKey, NwkKey, ctr_{ED}
AppKey, NwkKey, ctr_{JS}
End Device
Network Server
Join Server
Application Server
$ctr_{ED}++$
JoinReq $= ED, JS, ctr_{ED}, mac(ctr_{ED}, \text{NwkKey})$
1. JoinReq
2. JoinReq, n_{NS}
3. JoinReq
4. NS, JoinReq, n_{AS}
$ctr_{JS}++$
$k_{AS} = f_1(\text{AppKey}, ctr_{JS}, ctr_{ED}, n_{AS})$
$k_{NS} = f_2(\text{NwkKey}, ctr_{JS}, ctr_{ED}, n_{NS})$
NSResponse $= NS, ctr_{JS}, n_{NS}, k_{NS}$
EDResponse $= \{ED, ctr_{JS}, n_{NS}, n_{AS}\}_{\text{NwkKey}}$
ASResponse $= AS, ctr_{JS}, n_{AS}, k_{AS}$
5. NSResponse, EDResponse
6. ASResponse
7. EDResponse
$k_{AS} = f_1(\text{AppKey}, ctr_{JS}, ctr_{ED}, n_{AS})$
$k_{NS} = f_2(\text{AppKey}, ctr_{JS}, ctr_{ED}, n_{NS})$
8. $\{ED, NS, ctr_{ED}, n_{NS}\}_{k_{NS}}$
9. $\{NS, ED, ctr_{JS}, n_{NS}\}_{k_{NS}}$
10. $\{ED, NS, AS, \{ctr_{ED}, n_{AS}\}_{k_{AS}}\}_{k_{NS}}$
11. $\{ctr_{ED}, n_{AS}\}_{k_{AS}}$
12. $\{\{ctr_{JS}, n_{AS}\}_{k_{AS}}\}_{k_{NS}}$
13. $\{ctr_{JS}, n_{AS}\}_{k_{AS}}$

FIGURE 8.6: `LoRA 3-AKA`$^+$– A Minimally-Augmented LoRa 1.1. Join with More Security, Less Trust. Secure channels are displayed with dashed lines “--→”.

Essential Features of `LoRA 3-AKA`$^+$ Compared to the LoRa 1.1 Join. We work with the following principles in mind:

- The parties and network topology remain the same as in LoRa 1.1.
- The Application Server is “minimally” active during the Join Procedure: it contributes a nonce to the Join Request, adding randomness to the encryption key AppSKey and the MAC key NwkSKey.
- We envisage that the `LoRA 3-AKA`$^+$ specification require (with a ‘MUST’ keyword) that the *NS-AS* and *NS-JS* channels be secure; however, we show that even if these are compromised the (confusion-freeness) security guarantees are not lost.
- A “Key Confirmation” step is added at the end of the protocol, whereby the involved parties check that they have mutually agree on the established session keys. This key confirmation step occurs *before* the record phase begins.

The `LoRA 3-AKA`$^+$ design is as close as possible to LoRa 1.1 Join, whilst giving protection to the Application Server against any possible (key/device) confusion attacks such as those we have highlighted against LoRa 1.1.

The LoRA 3-AKA$^+$ Protocol.
Figure 8.6 gives an overview of LoRA 3-AKA$^+$.

The protocol LoRA 3-AKA$^+$ is split into two main phases. First, the keys are established using a process very similar to LoRa 1.1. Then, a set of "*Key Confirmation*" messages are sent between the End Device and the Network Server and Application Server respectively, in which the newly generated keys are checked for authenticity.

For readability, in this description we do not include references to parties' names or configuration parameters. We also collapse the multiple keys NwkSKey, FNwkSIntKey, SNwkSIntKey into a single value (k_{NS}). However, our Tamarin models do model the addition of parties' names and all LoRaWAN keys.

The LoRA 3-AKA$^+$ protocol proceeds as follows:

1. The End Device *ED* generates a Join Request in the same way as in LoRa 1.1, incrementing its DevNonce counter and sending it along with the request command, including a MAC of the message using the NwkKey.

2–3) The Network Server *NS* forwards the Join Request to the Join Server, adding a freshly generated nonce NwkNonce. The Join Request is also forwarded as-is to the Application Server.

4. The Application Server *AS* acknowledges the Join Request by forwarding another copy of it to the Join Server, adding a freshly generated nonce AppNonce.

5–7) The Join Server processes the Join Request, generating the following values:

$$\begin{aligned}
k_{AS} &= f_1(\mathsf{NwkKey}, \mathsf{JoinNonce}, \mathsf{DevNonce}, \mathsf{AppNonce}) \\
k_{NS} &= f_2(\mathsf{NwkKey}, \mathsf{JoinNonce}, \mathsf{DevNonce}, \mathsf{NwkNonce}) \\
\text{NSResponse} &= \mathsf{JoinNonce}, \mathsf{NwkNonce}, k_{NS} \\
\text{EDResponse} &= \{\mathsf{JoinNonce}, \mathsf{NwkNonce}, \mathsf{AppNonce}\}_{\mathsf{NwkKey}} \\
\text{ASResponse} &= \mathsf{JoinNonce}, \mathsf{AppNonce}, k_{AS},
\end{aligned}$$

NSResponse, ASResponse and EDResponse are then sent to *NS*, *AS* and *ED* (via *NS*), respectively.

8. The End Device re-calculates the keys independently. It then sends the first *key confirmation* request to the Network Server, consisting of the values DevNonceand NwkNonce, encrypted with the calculated k_{NS}.

9. The Network Server responds with a *key response* message, containing JoinNonce and NwkNonce.

10–13) The End Device sends a second key-confirmation message, this time to the Application Server (via the Network Server). These messages are encrypted using k_{AS}, making the message-body unreadable to the Network Server. The key-confirmation message contains values DevNonce and AppNonce, and the key-confirmation response contains values JoinNonce and AppNonce.

Threat Model	$\mathcal{M}_{\texttt{AS-NS-Secure}}$	$\mathcal{M}_{\texttt{LoRa1.1Spec}}$	$\mathcal{M}_{\texttt{NS-weakCorrupt}}$
Security Goal			
key_secrecy	✓	NA	✓
wa_ED_JS	✓	NA	✓
wa_ED_AS	✓	NA	✓
nia_ED_JS	✓	NA	✓
nia_ED_JS	✓	NA	✓
sync_ED_AS	✓	NA	✓
CF_ED_AS_packet_key	✓	NA	✓

TABLE 8.3: Main Verification Results for $\texttt{LoRA 3-AKA}^{+}$

Analysis of the $\texttt{LoRA 3-AKA}^{+}$ Protocol

We modelled the $\texttt{LoRA 3-AKA}^{+}$ protocol in the $\mathcal{M}_{\texttt{AS-NS-Secure}}$ and $\mathcal{M}_{\texttt{NS-weakCorrupt}}$models. These models represent the best-case and worst-case scenarios, depending on whether or not the *NS* is part of the fully trusted environment alongside the *JS* and *AS*.

We modelled the same main properties as for LoRa 1.1 Join.

(1) *weak agreement between ED and AS (*wa_ED_AS*)*

(2) *two-party confusion-freeness* (CF_ED_AS_packet)

(3) *explicit two-party confusion-freeness* (CF_ED_JS_AS_key)

(4) *explicit three-party confusion-freeness* (CF_ED_JS_AS)

We repeat that the aim of encoding CF_ED_AS_packet, as well as CF_ED_JS_AS is merely for proving purposes: i.e., one may prove faster than the other. As before, we created several oracles to simplify the verification process.

Our analysis inside Tamarin proves that wa_ED_AS, CF_ED_AS_packet, CF_ED_JS_AS, CF_ED_JS_AS_key, all hold on $\texttt{LoRA 3-AKA}^{+}$, in both the $\mathcal{M}_{\texttt{AS-NS-Secure}}$ and $\mathcal{M}_{\texttt{NS-weakCorrupt}}$ threat models. In particular, even when the channels in and out of the *NS* were compromised, the fact that the *AS* is alive, as well as the addition of the key confirmation step, prevents any agreement insecurities inside $\texttt{LoRA 3-AKA}^{+}$.

Intuitively, although a corrupt *NS* is able to grant the adversary access to the secure channels and thus read, redirect or inject messages, the addition of explicit checks prevents risk of confusion. Since the confirmation check between the *ED* and *AS* is encrypted using the just-established key (which the adversary cannot reproduce without knowledge of the long term AppKey), agreement and aliveness is ensured.

8.6 Conclusions

Interactions with the LoRa Alliance

As part of the research process, we engaged with the LoRa Alliance. As well as performing responsible disclosure of our findings, we also held a series of interactions with respect to the wider scope of formally verifying the current, as well as future, LoRaWAN specifications.

In relation our findings, the Security Working Group (SWG) and the Technical Committee (TC) of the LoRa Alliance are in alignment with us regarding the following main aspects:

1. clear requirements ought to be stipulated with respect to the security (i.e., confidentiality and integrity) of the NS-AS channel;
2. the *AS* should, to some extent, become an active part of Join, as this would increase the capabilities of the *AS* to verify the authenticity of session keys and application-layer messages.

During this interaction, we confirmed that the LoRa specifications do not explicitly require that the *NS*-*AS* channel be secure.In general, requirements on the *NS*-*AS* channel and the *AS* itself are slightly under-specified in the current version of the specifications due to the fact that the *NS* and the *AS* have been initially thought of as co-located parties. As the commissioning of proprietary Application Servers becomes increasingly common, the LoRa Alliance acknowledges that the issues of the under-specification of the security of the *NS*-*AS* channel as well the amount of trust placed on the *NS* have become more acute. This confirms the validity of our $\mathcal{M}_{\texttt{LoRa1.1Spec}}$ and $\mathcal{M}_{\texttt{NS-weakCorrupt}}$ models.

Indeed, within the LoRa Alliance SWG, work is already ongoing to properly specify requirements on the *NS*-*AS* channel. With regards to the trust placed in the *NS*, it is to be noted that parts of our `LoRA 3-AKA`$^{+}$ design were generally welcome. Concretely, the *AS* contributing with a nonce to the Join procedure is being considered by the LoRa Alliance SWG as a possible solution going forward. However, the key-confirmation step is viewed as potentially too computationally demanding on the end devices; as such, the latter could be simplified and/or made optional, if ever to be adopted.

Conclusions

Here we briefly summarise our results and discuss options for future avenues of research.

Main Recommendations for LoRa 1.0.

Although this is an older recommendation from [AF17], our results *formally* prove that the domain of the nonces in 1.0 ought to be increased. In combination with the results shown by us on LoRa 1.1, we also *formally* prove that the approach of counter-based "nonces" that the LoRa Alliance took to repair the replay resistance failure is not the correct one, as this is also a root of attacks. As such, a better approach would be to increase the domain of the nonces, which would prevent the replay resistance failure, without introducing additional risks.

Main Recommendations for LoRa 1.1.

Firstly, if the LoRa Alliance continue to adhere to their current threat-model, they should eliminate the AppSKey-delivery mode based on SessionID. Otherwise, the *AS* can be confused as to which device *ED* it is communicating with, resulting in severe consequences.

Secondly, we show that the LoRa Alliance should explicitly demand the *NS* and *AS* channel be secured.

Thirdly, we show that relying on one single crux of trust (i.e., the current *NS* in LoRa 1.1) is dangerous. Based on the current LoRa specifications, if only the *NS*-to-*JS* channel becomes compromised, then we show that all AKE security goals can fail.

Finally, we show that AKE-security can be regained even if the trust in the *NS* is lowered, if the LoRa 1.1 Join is modified to make the *AS* "alive" during the Join and include a "key confirmation" step. We believe that this is a good balance of security and trust.

Future Work

Even if this is not in the LoRa specifications, we believe strong device authentication and device unlinkability should be studied. The risk of version-downgrade attacks exists in LoRA. Modelling such attacks may be useful to get a better view of the ecosystem at large.

As the LoRa specifications continue to evolve, it is important that any security models continue to update with them. Similarly, the specificity of models can be even further improved in order to identify new attack vectors. For example, our models consider agent names to be public terms. However, in reality there is an implicit mapping between a *ED*'s DevEUI (which it uses to identify itself) and an associated DevAddr (which is used by the backend network).

Bibliography

[ABD$^+$15] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. Imperfect forward secrecy: How diffie-hellman fails in practice. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015.

[Ada11] Carlisle Adams. *Kerberos Authentication Protocol*, pages 674–675. Springer US, Boston, MA, 2011.

[AF17] Gildas Avoine and Loïc Ferreira. Rescuing LoRaWAN 1.0. In *IACR Cryptology ePrint Archive*, 2017.

[Ash15] Warwick Ashford. PrivDog SSL compromise potentially worse than Superfish. February 2015.

[BBD$^+$18] Karthikeyan Bhargavan, Ioana Boureanu, Antoine Delignat-Lavaud, Pierre-Alain Fouque, and Cristina Onete. A formal treatment of accountable proxying over TLS. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 799–816, 2018.

[BBDL$^+$15] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. A messy state of the union: Taming the composite state machines of tls. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 535–552. IEEE, 2015.

[BBF$^+$17] Karthikeyan Bhargavan, Ioana Boureanu, Pierre-Alain Fouque, Cristina Onete, and Benjamin Richard. Content delivery over tls: a cryptographic analysis of keyless ssl. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 1–16. IEEE, 2017.

[BBK17] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified models and reference implementations for the tls 1.3 standard candidate. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 483–502. IEEE, 2017.

[BK19] Sergiu Bursuc and Steve Kremer. Contingent payments on a public ledger: models and reductions for automated verification. *IACR Cryptology ePrint Archive*, 2019:443, 2019.

[Bla12] Bruno Blanchet. Security protocol verification: Symbolic and computational models. In *Proceedings of the First international conference on Principles of Security and Trust*, pages 3–29. Springer-Verlag, 2012.

[Bla13] Bruno Blanchet. Automatic verification of security protocols in the symbolic model: The verifier proverif. In *Foundations of Security Analysis and Design VII*, pages 54–87. Springer, 2013.

[BMD$^+$17] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software grand exposure:SGX cache attacks are practical. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, 2017.

[BPG18] Ismail Butun, Nuno Pereira, and Mikael Gidlund. Analysis of lorawan v1.1 security: research paper. pages 1–6, 06 2018.

[CAB$^+$15] Chen Chen, Daniele E Asoni, David Barrera, George Danezis, and Adrain Perrig. Hornet: High-speed onion routing at the network layer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1441–1454. ACM, 2015.

[CCC$^+$16] Frank Cangialosi, Taejoong Chung, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Wilson Wilson. Measurement and analysis of private key sharing in the https ecosystem. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 628–640. ACM, 2016.

[CCM16] Taejoong Chung, David Choffnes, and Alan Mislove. Tunneling for transparency: A large-scale analysis of end-to-end violations in the internet. In *Internet Measurement Conference (IMC)*, 2016.

[CF19] Sébastien Canard and Loïc Ferreira. Extended 3-Party ACCE and Application to LoRaWAN 1.1. In Johannes Buchmann, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *Progress in Cryptology – AFRICACRYPT 2019*, pages 21–38, Cham, 2019. Springer International Publishing.

[Cha81] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[CHH$^+$17] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of tls 1.3. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1773–1788, New York, NY, USA, 2017. ACM.

[Com11] Comodo. Comodo report of incident - comodo detected and thwarted an intrusion on 26-mar-2011, 2011.

[CSF$^+$08] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. RFC 5280, May 2008. http://www.ietf.org/rfc/rfc5280.txt.

[dCdCM16] Xavier de Carné de Carnavalet and Mohammad Mannan. Killed by proxy: Analyzing client-end TLS interception software. In *Network and Distributed System Security Symposium*, 2016.

[DG09] George Danezis and Ian Goldberg. Sphinx: A compact and provably secure mix format. In *Security and Privacy, 2009 30th IEEE Symposium on*, pages 269–282. IEEE, 2009.

[Die08] Tim Dierks. The transport layer security (TLS) protocol version 1.2. 2008.

[DMS$^+$17] Zakir Durumeric, Zane Ma, Drew Springall, Richard Barnes, Nick Sullivan, Elie Bursztein, Michael Bailey, J. Alex Halderman, and Vern Paxson. The security impact of https interception. In *Network and Distributed Systems Symposium*, 2017.

[DR08] Tim Dierks and Eric Rescorla. The transport layer security (TLS) protocol version 1.2. Technical report, NIST, 2008.

[DY83] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.

[E. 18] E. Rescorla. The transport layer security (TLS) protocol version 1.3 (draft 28). 2018.

[EBPG19] Mohamed Eldefrawy, Ismail Butun, Nuno Pereira, and Mikael Gidlund. Formal security analysis of lorawan. *Computer Networks*, 148:328 – 339, 2019.

[Fac] Facebook. Introducing our certificate transparency monitoring tool.

[FBK$^+$17] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. Measuring HTTPS adoption on the web. In *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017.*, 2017.

[Gau13] Gaurang. Nokia's MITM on HTTPS traffic from their phone. 2013.

[HCP17] Marcus Hähnel, Weidong Cui, and Marcus Peinado. High-resolution side channels for untrusted operating systems. In *2017 USENIX Annual Technical Conference (USENIXATC 17)*, pages 299–312, 2017.

[HKHH17] Juhyeng Han, Seongmin Kim, Jaehyeong Ha, and Dongsu Han. Sgx-box: Enabling visibility on encrypted traffic using a secure middlebox module. In *Proceedings of the First Asia-Pacific Workshop on Networking*, pages 99–105. ACM, 2017.

[HREJ14] Lin Shung Huang, Alex Rice, Erling Ellingsen, and Collin Jackson. Analyzing forged ssl certificates in the wild. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 83–97. IEEE, 2014.

[Int17] Intel. Intel software guard extensions sdk for linux os (1.9 release). Technical report, 2017.

[ITU00] ITU-T RECOMMENDATION. Information technology–open systems interconnection–the directory: Public-key and attribute certificate frameworks. 2000.

[JU12] Jeff Jarmoc and DSCT Unit. SSL/TLS interception proxies and transitive trust. *Black Hat Europe*, 2012.

[KW03] Chris Karlof and David Wagner. Secure routing in wireless sensor networks: Attacks and countermeasures. *Ad hoc networks*, 1(2-3):293–315, 2003.

[LCS$^+$19] Jie Li, Rongmao Chen, Jinshu Su, Xinyi Huang, and Xiaofeng Wang. Me-tls: Middlebox-enhanced tls for internet-of-things devices. *IEEE Internet of Things Journal*, 2019.

[LLK13] Ben Laurie, Adam Langley, and Emilia Kasper. Certificate transparency. Technical report, 2013.

[LMS$^+$12] S. Loreto, J. Mattsson, R. Skog, H. Spaak, G. Gus, and D. Druta. Explicit trusted proxy in http/2.0. 2012.

[Low97] Gavin Lowe. A hierarchy of authentication specification. In *CSF'97*, pages 31–44, 1997.

[LSL$^+$19] Hyunwoo Lee, Zach Smith, Junghwan Lim, Gyeongjae Choi, Selin Chun, Taejoong Chung, and Ted Taekyoung Kwon. matls: How to make tls middlebox-aware? In *NDSS*, 2019.

[LSP$^+$16] Chang Lan, Justine Sherry, Raluca Ada Popa, Sylvia Ratnasamy, and Zhi Liu. Embark: Securely outsourcing middleboxes to the cloud. In *NSDI*, volume 16, pages 255–273, 2016.

[MAM$^+$99] Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Carlisle Adams. X. 509 internet public key infrastructure online certificate status protocol-ocsp. 1999.

[Mey13] David Meyer. Nokia: Yes, we decrypt your https data, but don't worry about it, 2013.

[MMSS$^+$19] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous multi-hop locks for blockchain scalability and interoperability. In *NDSS*, 2019.

[MSCB13] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The TAMARIN prover for the symbolic analysis of security protocols. In

International Conference on Computer Aided Verification, pages 696–701. Springer, 2013.

[MWNG12] D. McGrew, D. Wing, Y. Nir, and P. Gladstone. TLS proxy server extension. 2012.

[Nar13] V. Narayanan. Explicit proxying in HTTP-problem statement and goals. 2013.

[NFL$^+$14] David Naylor, Alessandro Finamore, Ilias Leontiadis, Yan Grunenberger, Marco Mellia, Maurizio Munafò, Konstantina Papagiannaki, and Peter Steenkiste. The cost of the s in https. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 133–140. ACM, 2014.

[Nir12] Y. Nir. A method for sharing record protocol keys with a middlebox in TLS. 2012.

[NLG$^+$17] David Naylor, Richard Li, Christos Gkantsidis, Thomas Karagiannis, and Peter Steenkiste. And then there were more: Secure communication for more than two parties. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 88–100. ACM, 2017.

[Not14] M. Nottingham. Problems with proxies in http. 2014.

[NSV$^+$15] David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego López, Konstantina Papagiannaki, Pablo Rodriguez Rodriguez, and Peter Steenkiste. Multi-context tls (mctls): Enabling secure in-network functionality in tls. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 199–212. ACM, 2015.

[O'B18] Devon O'Brien. Certificate transparency enforcement in google chrome. 2018.

[ORSZ16] Mark O'Neill, Scott Ruoti, Kent Seamons, and Daniel Zappala. Tls proxies: Friend or foe? In *Proceedings of the 2016 Internet Measurement Conference*, pages 551–557. ACM, 2016.

[PD16] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. *See https://lightning. network/lightning-network-paper. pdf*, 2016.

[PLPR18] Rishabh Poddar, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. Safebricks: Shielding network functions in the cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18), Renton, WA*, 2018.

[PN18] Dmytro Piatkivskyi and Mariusz Nowostawski. Split payments in payment networks. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 67–75. Springer, 2018.

[Pop15] Andrey Popov. Prohibiting RC4 cipher suites, 2015.

[Res00] Eric Rescorla. Http over TLS, 2000.

[Res18] Eric Rescorla. The transport layer security (TLS) protocol version 1.3. Technical report, NIST, 2018.

[RLT17] Elias Rohrer, Jann-Frederik Laß, and Florian Tschorsch. Towards a concurrent and distributed route selection for payment channel networks. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 411–419. Springer, 2017.

[RMSKG17] Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. Settling payments fast and private: Efficient decentralized routing for path-based transactions. *arXiv preprint arXiv:1709.05748*, 2017.

[SB08] Robin Snader and Nikita Borisov. A tune-up for Tor: Improving security and performance in the tor network. In *ndss*, volume 8, page 127, 2008.

[SDM04] Paul Syverson, R Dingledine, and N Mathewson. Tor: The second generation onion router. In *Usenix Security*, 2004.

[Sep15] Timothy J. Seppala. New Lenovo PCs shipped with factory-installed adware. 2015.

[SH12] Jakob Schlyter and Paul Hoffman. The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA, 2012.

[SHS$^+$12] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: network processing as a cloud service. volume 42, pages 13–24. ACM, 2012.

[SLE$^+$] N Sornin, M Luis, T Eirich, T Kramp, and O Hersent. LoRaWAN Specification, Version V1.0.2;. *LoRa Alliance, https://lora-alliance.org/resource-hub/lorawanr-specification-v102*.

[SLPR15] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. volume 45, pages 213–226. ACM, 2015.

[Sor] N Sornin. LoRaWAN 1.1 Specification (April 2018). *LoRa Alliance, https://lora-alliance.org/sites/default/files/2018-04/lorawantm_specification_-v1.1.pdf*, 1.

[SSL] SSLMate. Cert spotter.

[Tea20] Lightning Network Team. Lightning bolt specification, 2020.

[The18] The Tamarin Team. *The Tamarin User Manual*, 2018.

[TII$^+$18] Giorgos Tsirantonakis, Panagiotis Ilia, Sotiris Ioannidis, Elias Athanasopoulos, and Michalis Polychronakis. A large-scale analysis of content modification by open http proxies. In *Network and Distributed System Security Symposium (NDSS)*, 2018.

[TKG$^+$18] Bohdan Trach, Alfred Krohmer, Franz Gregor, Sergei Arnautov, Pramod Bhatotia, and Christof Fetzer. Shieldbox: Secure middleboxes using shielded execution. In *Proceedings of the Symposium on SDN Research*, page 2. ACM, 2018.

[TWE$^+$04] Steven Tuecke, Von Welch, Doug Engert, Laura Pearlman, and Mary Thompson. Internet x. 509 public key infrastructure (PKI) proxy certificate profile. 2004.

[VBMW$^+$18] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 991–1008, 2018.

[VBWK$^+$17] Jo Van Bulck, Nico Weichbrodt, Rüdiger Kapitza, Frank Piessens, and Raoul Strackx. Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1041–1056, 2017.

[Wil15] Owen Williams. Google dropping CNNIC root CA after trust breach, 2015. https://thenextweb.com/insider/2015/04/02/google-to-drop-chinas-cnnic-root-certificate-authority-after-trust-breach/.

[WMY18] Louis Waked, Mohammad Mannan, and Amr Youssef. To intercept or not to intercept: Analyzing TLS interception in network appliances. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 399–412. ACM, 2018.

Chapter 9

Conclusion

In this thesis we demonstrated several strategies for developing security goals specialised for certain protocols. These needs arise out of the demands of the domain - accomodating for physical contraints or in order to encode new adversarial assumptions.

Often a key challenge is in modelling the protocols themselves. A necessary choice is found in the level of granularity of a model. Unfortunately, increasing the specificity of a model can often impact the tenability of constructing security proofs (especially using automated tools), whilst there are countless examples of scenarios where overly coarse analysis leads to inaccurate results.

As such, a very successful strategy is found in building restricted frameworks within which to analyse our protocols. Often, reasoning about the larger setting allows for the construction of more general lemmas that provide a bridge between the intuition of a security goal and an encoding in the specification language of choice.

Unfortunately there are some risks with such an approach. Firstly, restricting our protocols to those built within a framework means that there may be protocols that we cannot describe. Newer protocols often make use of novel cryptographic constructions that may not fit comfortably within our assumptions. Secondly, introducing an extra layer of abstraction gives a new opportunity for modelling errors to fit through the cracks. A very clear example is found in the domain of privacy-based security goals. Such goals make use of some form of equivalence, and it is becoming increasingly apparent that seemingly minor differences in the definitions (often made to simplify the analysis process) can lead to different conclusions being drawn.

Future Work

There is significant room for future work, both in the individual protocol domains considered in each chapter of this work, as well as in the underlying strategies that we have employed.

Although the Dolev-Yao adversary forms an excellent starting point for analysis, we have repeatedly seen that it is not always the optimal choice. This can work in either direction. In Chapter 4 we sought an adversary that was somewhat stronger than traditionally used, by allowing for "partial" corruption. However, in Chapter 8 we saw that for the problem of automating analysis of a particularly complex protocol it was helpful to weaken some parts of the adversary (for example, by introducing typing) to make achieving results tenable.

Ultimately, the best approach here may be in understanding the impact of our assumptions, rather than trying to push for uniformity in our definitions. Work by Basin and Cremers shows that there is an intrinsic relationship between security goals and adversaries – one security goal in one adversary model might be equivalent to a similar goal under a different adversary. On the other hand, work by Chrétien et al. shows that in some situations, simplifying assumptions may not change the precision of our results.

In the longer term, it may be necessary to gain more control over the tools we use. Automated provers generally rely on a set of base assumptions – both to simplify the modelling process and in order to produce soundness results relating to the output of their analysis. Given that we often design our frameworks in order to accommodate for integration with provers, we are sometimes limited by these underlying assumptions. Unfortunately, modifying the tools may often inevitably lead to new problems – as any changes necessarily require new proofs of the correctness of the tool. However, we have seen such developments in the past: each new generation of provers generally brings about more and more control over our specification model. The work in this thesis would not be possible without the advanced tools we have today, and undoubtedly the next generation of provers will allow for models that are even more general still.

Finally, there is always room for growth in the process of designing security goals and understanding the needs of each protocol domain. Several of the attacks discussed in this thesis arise not necessarily out of a protocol's failure to achieve the security goals laid out in its specification, but rather a failure to consider certain security requirements at all. This is often the case as protocol domains expand in complexity – an increase in real-world demands if often not matched with a corresponding shift in the security requirements of this change. We can also often see that as the scope of a protocol expands, it may gain the features of protocols from other domains. As a community, we must be diligent in identifying these connections and understanding which security goals may be shared.