DISS. ETH NO. 31284

PROTOCOL DESIGN AND ANALYSIS IN THE SYMBOLIC MODEL

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES
(Dr. sc. ETH Zurich)

presented by

FELIX E. LINKER

born on 12.02.1995

accepted on the recommendation of

Prof. Dr. David Basin, examiner
Prof. Dr. Ralf Küsters, co-examiner
Prof. Dr. Peter Müller, co-examiner

2025

ABSTRACT

Recent years have seen substantial progress in the automated formal analysis of security protocols in the symbolic model. In a formal analysis of a security protocol, one mathematically and formally proves that it provides desired security guarantees. Formal analysis not only establishes confidence in a design's security, but also forces one to be explicit about desired security guarantees and assumptions required to establish them.

Given the many success stories, one may wonder why it has not become a requirement for security protocols to be formally analyzed prior to their deployment. One reason is likely that applying a formal analysis to a new protocol remains challenging despite the advances of recent years. First, it is non-trivial to identify a protocol's desired security guarantees, let alone formalize them such that they can be proven. Second, automated security protocol verification tools like Tamarin and ProVerif often do not scale well to large and complex protocols. Modern security protocols often use complex loop structures, which introduce sources of non-termination that must be carefully accounted for. In this thesis, we present contributions related to both of these problems in two parts. We show how to achieve provable, system-wide security guarantees and advance the state-of-the-art of protocol verification in the symbolic model.

In the first part, we focus on identifying and formalizing system-wide security guarantees, concretely long-term key authentication, and we present two systems that provably provide such guarantees. As many systems aiming to provide authentic long-term keys relied on trust assumptions that were shown to often not hold in practice, new alternative approaches were suggested. First, social authentication was suggested as an intuitive and usable paradigm for long-term key authentication between end users. Using social authentication, users authenticate their peers using digital identities managed by identity providers. Second, it was suggested to augment systems that provide long-term key authentication by relying on certifying authorities with accountability mechanisms. Should an authority ever maliciously attest a key's authenticity, accountability mechanisms make such misbehavior detectable and punishable. Thus, the hope is that such misbehavior will not happen in the first place. We present SOAP, a SOcial Authentication Protocol, which provides social authentication, and ADEM, an Authentic Digital EMblem, which provides authentication using an accountability mechanism, and we formally prove both systems' security.

In the second part, we turn our attention to complex protocols with looping control structures. One of the main challenges for modern, automated protocol verifiers is that modern security protocols often use complex loop structures, which introduce sources of non-termination and complicate automated proof search. We present two ways of using induction to prove properties of looping protocols using the Tamarin prover. We first show how one can use trace induction, a well-established proof methodology, to analyze looping protocols in their full complexity with modern, state-of-the-art symbolic provers by proving the iMessage PQ3 protocol secure. Second, we develop and implement a novel cyclic induction mechanism for the security protocol domain and show how it can be used to prove looping protocols secure while requiring considerably less effort than when using trace induction.

ZUSAMMENFASSUNG

Die formale Analyse von Sicherheitsprotokollen hat sich in den letzten Jahren drastisch weiterentwickelt. Wenn man ein Sicherheitsprotokoll formal analysiert, beweist man mathematisch rigoros, dass es gewünschte Sicherheitsanforderungen erfüllt. Eine formale Analyse etabliert nicht nur, dass ein Protokoll tatsächlich sicher ist, sondern zwingt einen zudem, gewünschte Sicherheitsanforderungen und Beweisannahmen zu formalisieren.

Man kann sich fragen, warum es trotz vieler Erfolgsgeschichten nicht zum Standard geworden ist, Sicherheitsprotokolle formal zu analysieren, bevor sie ausgerollt werden. Vermutlich ist ein Grund, dass die formale Analyse eines neuen Protokolls nach wie vor herausfordernd ist - trotz aller Fortschritte der letzten Jahre. Zum einen ist es nicht trivial, die gewünschten Sicherheitsanforderungen eines Protokolls zu identifizieren und zu formalisieren. Zum anderen stossen Programme zur automatisierten, formalen Analyse von Protokollen, wie zum Beispiel Tamarin und ProVerif, an ihre Grenzen, sobald sie auf grosse und komplexe Protokolle angewendet werden. Moderne Sicherheitsprotokolle verwenden jedoch oft komplexe Loop-Strukturen, die mit Vorsicht gehandhabt werden müssen, da sie leicht dazu führen, dass Computer-gestützte Beweise nicht terminieren. In dieser Dissertation präsentieren wir wissenschaftliche Beiträge zu beiden Problemstellungen in zwei Teilen.

Im ersten Teil konzentrieren wir uns auf das Identifizieren und Formalisieren von Sicherheitseigenschaften. Genauer konzentrieren wir uns auf die Authentifizierung von Public Keys und präsentieren zwei Systeme, die dies ermöglichen. Viele Systeme, deren Ziel es war, klassisch Public Keys zu authentifizieren, stellten sich im Nachhinein als unsicher heraus, weil sie auf Annahmen beruhten, die in der Praxis nicht erfüllt wurden. In Konsequenz wurde vorgeschlagen, das Ziel der klassischen Public Key Authentifizierung fallen zu lassen, und stattdessen andere Sicherheitseigenschaften zu garantieren. Der erste Vorschlag heisst “Social Authentication” und Social Authentication verspricht intuitive und benutzerfreundliche Authentifizierung von Public Keys. Wenn ein Nutzer den anderen “sozial authentifiziert”, wird überprüft, dass der Peer eine oder mehrere digitale Identitäten kontrolliert, die für gewöhnlich von einem Provider wie einem Social Network verwaltet werden. Der zweite Vorschlag sieht vor, Systeme, in denen Public Keys von Zertifizierungsstellen attestiert werden, um einen Rechenschaftsmechanismus zu ergänzen. Sollte eine solche Zertifizierungsstelle fälschlicherweise Public Keys attestieren, kann das durch einen solchen Mechanismus festgestellt und bestraft werden.

Die Hoffnung ist, dass sich Zertifizierungsstellen somit von Anfang an korrekt verhalten. Wir präsentieren zwei Systeme und analysieren beide formal: SOAP, ein "SOcial Authentication Protocol", implementiert Social Authentication und ADEM, ein "Authentic Digital EMblem", nutzt einen Rechenschaftsmechanismus.

Im zweiten Teil wenden wir uns der formalen Analyse von komplexen Protokollen mit loopenden Kontrollstrukturen zu. Für moderne Programme zur automatisierten, formalen Analyse von Protokollen ist es eine der grössten Herausforderungen, mit solchen Kontrollstrukturen umzugehen. Sie verkomplizieren die automatisierte Beweissuche und führen schnell dazu, dass diese Suche nicht terminiert. Wir verwenden zwei mathematische Induktionsverfahren, um Computergestützte Beweise über solche loopenden Kontrollstrukturen zu führen. Zunächst zeigen wir, wie "Trace Induktion" verwendet werden kann, um zu Beweisen, dass das iMessage PQ3 seine hohen, selbstgesteckten Sicherheitsanforderungen erfüllt. Wir verwenden dazu Tamarin, ein Programm zur Analyse von Sicherheitsprotokollen. Danach entwickeln wir ein neues, zyklisches Induktionsverfahren zur automatisierten Beweisführung für Sicherheitsprotokolle. Wir implementieren dieses zyklische Induktionsverfahren für Tamarin und zeigen, dass Beweise, die auf zyklischer Induktion beruhen, mit deutlich weniger Aufwand konstruiert werden können als Beweise, die auf Trace Induktion beruhen.

ACKNOWLEDGMENTS

I am fortunate to say that the last five years were the best of my life, and there are many people who made that possible. It all starts with my parents and sister, who always supported me in what I wanted to do. I always felt free to explore life, and never felt pressured to live up to any standards. I know that I can appear at your doorstep any moment, but even if I do not, I know you always have my back.

There is one thing that I am particularly grateful for when looking back at my life, and that is the number of people who supported me extraordinarily. Thank you to my mathematics teacher Mr Heinz (unfortunately, I never learned his first name) who "ordered" me out of boring Biology classes to let me read books on philosophy. Thank you to my philosophy teacher and now friend, Mandy Haupt, who let me teach my first philosophy classes and sparked my love for philosophy. Thank you, Gerhard Brewka and Ringo Baumann. You let me take my first steps in research and took me to my first conference. Without you, David would have never had the opportunity to tease me about Dung logics every two weeks or so. Thank you to Andreas Maletti, who I consider to be my first mentor without us having ever talked about it. Thank you for your support in moments of doubt. Finally, thanks to Alastair Reid, without whom I would have probably never started my Doctorate. To this day, I wonder what would have happened if you had not offered me an internship on formal methods after knowing me for no longer than five minutes.

I would also like to thank all those who made Leipzig and Zürich my new homes. To my friends from Leipzig (whether you still live there or not), thanks for creating a city that feels more like home the more often I return. Thank you to my friends in Zürich who created a new home for me after the pandemic caused a rough start. Thank you to my triathlon girls (also known as Mädels). You were the first to truly make me feel welcomed at the Triathlon Club Zürich and I would not want to miss a single kilometer we rode together. Thank you to my former partners, Antonia and Sofia. You always believed in me, especially when I did not, and I am thankful for the time we spent together.

To all my colleagues at ETH Zurich, thank you. You created a welcoming and friendly workplace where ideas are exchanged and friendships were found. You show that striving for excellence works best together. Thank you to my colleagues at the ICRC, Mauro Vignati, Samit D'Cunha, and Tilman Rodenhäuser, for introducing me to the world of diplomacy. You helped me grow in ways I would have never imagined when starting my Doctorate, and made the tough parts

of diplomacy enjoyable because of your presence. Thank you, Peter Müller and Ralf Küsters, for agreeing to review my thesis in great detail.

And finally, thank you, David. I remember when I started realizing how much trust you put into me. You let me be the face of our research while our competitors were represented by professors with years of experience and reputation. I always felt you cared about me more than yourself. And also, it was great fun working together with you.

CONTENTS

1 INTRODUCTION

1.1 FORMAL ANALYSIS OF SECURITY PROTOCOLS

Recent years have seen substantial progress in the automated formal analysis of security protocols. A *formal analysis* of a security protocol is an attempt to mathematically and formally prove that it provides desired security guarantees, and numerous real-world protocols have been analyzed using state-of-the-art, automated tools such as Tamarin [112, 139] and ProVerif [25, 26]. Ideally, formal analysis accompanies a protocol's development. This is what happened for TLS 1.3 [129], the backbone of today's Internet, which was formally analyzed during its standardization efforts, securing it by design [51, 52].

Unfortunately, formal analysis during the design phase has neither always been nor has it become the norm. The EMV protocol powers the communication of credit cards with credit card terminals. EMV is used in more than 80% of all transactions that use card payment [14], but its development was not accompanied by formal analysis. More than 20 years after the protocol's initial release and deployment, researchers found the protocol to be insecure while trying to prove it secure [14, 15]. Concretely, attackers could bypass the requirement to enter a PIN when paying with stolen credit cards.

A formal analysis of a security protocol includes three steps. First, the protocol, its environment, and assumptions on the adversary's capabilities are modeled in a mathematical framework, for example, as a state transition system. Second, the protocol's desired security guarantees are formalized, for example, as first-order logic formulas. Third and finally, one proves that the protocol model provides the formalized properties in the modeled environment and against the adversary considered. Formal analysis not only establishes confidence in a design's security, but also forces one to be explicit about desired security guarantees and assumptions required to establish them. Oftentimes, precisely defining security guarantees and assumptions is as valuable as the proof itself.

The analysis of security protocols typically takes place in either the *computational setting* or in the *symbolic model* of cryptography, and both have their relative strengths and weaknesses. In the computational setting, protocols are analyzed with respect to computational definitions of security. Agents manipulate bit strings, the adversary's capabilities are modeled by probabilistic polynomial-time Turing machines, and security definitions are thus probabilistic. These models support a detailed analysis of cryptography. However, the proofs

can be quite complex and hence they typically involve their own abstractions or protocol simplifications. Moreover, given that the proofs are traditional pen-and-paper arguments, they are more error-prone than proofs checked by computers. There are exceptions, namely computational proofs constructed with tools like CryptoVerif [24] or ProofFrog [62], but these are usually limited to the study of relatively simple combinations of primitives.

In the symbolic model of cryptography, messages are represented as terms in a term algebra (rather than bit-strings) and one uses possibilistic rather than probabilistic definitions of security. Tamarin, the tool that was used to analyze the TLS 1.3 and EMV protocols, ProVerif, and DY* [19] are examples of tools constructing proofs in this setting. For example, to show that the adversary cannot learn a secret, one would use these tools to prove that, no matter how arbitrarily many protocol runs are interleaved, including runs where the adversary is active, the adversary cannot possibly learn the intended secret.

There are benefits to having both computational and symbolic proofs as they both have their relative strengths. Computational proofs capture the detailed cryptographic assumptions on the operators used. They can also capture the adversary's advantage in attacking a protocol, by bounding the probability of success for an adversary with given computational resources. In contrast, symbolic proofs support machine-checked proofs much better, using different computer-supported proof techniques, like constraint solving and mathematical induction. Such proofs may be constructed automatically or interactively, and attempts to prove false statements generally yield attacks on the specified properties, such as for the EMV protocol. This support for automation typically allows for larger protocol models and more complex adversary capabilities and operational semantics, considering unboundedly many protocol participants and interleaved parallel sessions, and verifying these against detailed, fine-grained security properties.

Given success stories like the formal analysis of TLS 1.3 or EMV, which were both carried out in the symbolic model, one may wonder why it has not become a requirement for security protocols to be formally analyzed prior to their deployment. One reason is likely that applying a formal analysis to a new protocol remains challenging despite the advances of recent years. First, it is non-trivial to identify a protocol's desired security guarantees, let alone formalize them such that they can be proven. While there are many standard formal notions that capture the security guarantees of individual cryptographic primitives in the computational settings, this is often not the case for security guarantees of entire systems built from many such components. Second, automated security protocol verification tools like Tamarin and ProVerif often do not scale well to large and complex protocols. Modern security protocols often use complex loop

structures, which introduce sources of non-termination that must be carefully accounted for.

In this thesis, we present contributions related to both of these problems. We show how to achieve provable, system-wide security guarantees and advance the state-of-the-art of protocol verification in the symbolic model. In Part i, we focus on the formalization of system-wide security guarantees as trace properties. Concretely, we consider long-term key authentication and present two systems that provably provide these guarantees. In Part ii, we turn our attention to complex protocols with looping control structures. We show how both an existing and a novel proof technique can be used to formally prove properties of protocols with loops.

1.2 FORMAL LONG-TERM KEY AUTHENTICATION GUARANTEES

Security protocols can only provide their desired security guarantees if peers use each other's authentic cryptographic keys when running them. For example, when encrypting a message for a recipient, that message can only remain secret between the sender and intended recipient if the intended recipient's key was used for encryption. Using inauthentic keys immediately voids any security provided by the encryption: A user may encrypt their messages for the adversary directly. To address the non-trivial problem of obtaining each other's authentic cryptographic keys, protocols most often use or specify a *public key infrastructure (PKI)*, which associates pseudonyms with (ideally) authentic public keys. The property that PKIs aim to establish is *long-term key authentication*.

The most prominent example of a PKI is the Web PKI, which associates domain names with public keys and thus powers all HTTPS connections made by browsers. In the Web PKI, certificate authorities (CAs) sign certificates that attest which domain names should be associated with which public keys. Browsers come with pre-installed *root CAs* which the browser trusts, and CAs can sign *CA certificates* that declare other CAs as trusted. A browser then trusts all root CAs and all CAs for which it has a CA certificate signed by another trusted CA. By default, any CA can sign a certificate for any website, and thus compromise of a single CA can have devastating consequences. The inevitable happened in 2011 when it was discovered that the DigiNotar CA was compromised. DigiNotar keys were used to sign fraudulent certificates for google.com, which were used to attack Iranian users. An investigation of this attack found that DigiNotar had issued at least 531 fraudulent certificates [95].

A system less often thought of as PKI is secure messaging, and secure messaging has a history of vulnerabilities too. Modern, secure messengers like iMessage, Signal, or WhatsApp promise their users *end-to-end encryption*: Users, typically identified by phone numbers,

can send messages to one another, and only the two intended "ends" should know about those messages' contents. End-to-end encryption consists of two components. Clients first use a key directory operated by their messaging provider to obtain their peers' long-term public keys. They then use an end-to-end encryption protocol to send and encrypt their messages.

Many modern messengers, perhaps surprisingly, allow their users to communicate without properly authenticating their chat partners. When they opt to do that, users trust the authentication performed by the application provider during registration and that the application's key directory correctly reports other users' public keys to them. For messengers that use phone numbers to identify their users, registration only requires entering an SMS one-time password (OTP) that the messaging provider sent to a phone number claimed by a registering user. This key distribution practice requires trusting both the security of SMSes and the application servers themselves, but the breach of Signal's SMS OTP provider Twilio [157] calls into question whether that trust is warranted. Through social engineering, attackers gained read access to Signal SMS OTPs and re-registered phone numbers, one of which belonged to a prominent journalist [70].

Motivated by the above attacks, recent work proposed to approach long-term key authentication from two new angles. Concretely, it was suggested to augment PKIs with transparency systems and to design PKIs such that they provide social authentication.

A number of *transparency systems* have been proposed as an alternative to key authentication [41, 96, 97, 99, 100, 108, 113, 165]. In a transparency system, providers such as CAs or messaging providers must commit to the pseudonym-to-key bindings they attest in such a way that they later cannot deny having made that attestation. Transparency systems suggest augmenting long-term key authentication with an *accountability* mechanism. Should a provider ever maliciously attest a pseudonym-to-key binding, that attestation is stored in a transparency log. This makes any misbehavior detectable and allows holding providers accountable. Thus, the hope is that providers will not misbehave in the first place.

Social authentication was suggested as a simple and usable long-term key authentication protocol for secure messaging [86, 159]. When performing social authentication, users verify that their actual chat partner controls accounts at different *identity providers (IdPs)*, for example, their social media providers, which they know are controlled by their intended chat partner.

In Part i of this thesis, we present two systems that use social authentication and accountability respectively to provably provide long-term key authentication guarantees. We start by presenting *SOAP, a SOcial Authentication Protocol*, a secure and practical protocol that implements social authentication, in Chapter 4. Although past works

have presented designs, social authentication has never been studied as an authentication protocol. No prior work defined what security guarantees social authentication should provide, let alone considered whether a given design correctly provides those guarantees. We precisely define and prove that SOAP provides the security objectives of social authentication.

Afterwards in Chapter 5, we present *ADEM, an Authentic Digital EMblem*, which tackles a novel security problem that arises in times of armed conflict. International humanitarian law (IHL) mandates that military units must not target medical facilities, such as hospitals. The emblems of the Red Cross, Red Crescent, and Red Crystal are used to mark *physical* infrastructure (e.g., by a Red Cross painted on a hospital's rooftop), thereby enabling military units to identify those assets as protected under IHL. ADEM extends such markings to *digital* infrastructure such as servers and networks, which raises unique security challenges. ADEM solves these challenges by relying on an accountability mechanism. In that, it follows the ideas behind transparency systems. The marking of assets with ADEM was designed so that one can hold parties accountable who illegitimately mark unprotected infrastructure. Next to ADEM itself, we present a formal analysis, which includes a formal definition and machine-checked proofs of ADEM's system-wide accountability guarantees that we use to provide authenticity.

1.3 AUTOMATED ANALYSIS OF LOOPING PROTOCOLS

One of the main challenges for modern, automated protocol verifiers is that modern security protocols often use complex loop structures, which introduce sources of non-termination and complicate automated proof search. In particular, modern end-to-end encryption protocols used in secure messaging have become quite intricate and use complex, nested loop structures.

Historically, end-to-end encryption protocols provided security against adversaries that can compromise participants' long-term key material and that control the network over which messages are exchanged. By now, secure messengers aim to additionally guarantee secure communication against adversaries who can compromise partial session state during protocol execution or who may obtain quantum computing capabilities in the future. In particular, concerns have focused on "harvest now, decrypt later" quantum adversaries, who exploit the decreasing cost of mass storage to intercept and store ciphertexts. Such adversaries anticipate future developments in quantum computing and aim to decrypt ciphertexts as soon as sufficiently powerful quantum computers become available.

Considering partial session state compromise led to the study of the security guarantees *forward secrecy* and *post-compromise security* [27,

49]. Forward secrecy provides resistance against partial session state compromise in the future and guarantees that messages exchanged prior to the compromise remain secure. Post-compromise security provides resistance against partial session state compromise in the past. A protocol providing post-compromise security enables its clients to recover security after a compromise and to securely exchange messages in the future again.

One of the first protocols to provide both forward secrecy and post-compromise security was the Signal protocol, which extended the design of the Off-the-Record Messaging protocol [30]. The Signal protocol combines the X3DH handshake protocol [110] for key establishment with the double ratchet algorithm [121] for key derivation and message encryption. The handshake protocol establishes a shared secret using long-term key material. In the double ratchet, clients continuously update that shared secret with fresh, ephemeral key material in an outer ratchet and use that shared secret to derive per-message encryption keys in an inner ratchet. Clients use non-invertible Key Derivation Functions (KDFs) to update the shared secret and derive encryption keys, which coined the term "ratcheting." The key schedule is like a ratchet that can only move in one direction.

Intuitively speaking, the double ratchet provides forward secrecy because clients continuously update message encryption keys using non-invertible KDFs. It provides post-compromise security because clients use fresh, ephemeral key material to update the shared secret. Even if an adversary learned, for example, the shared secret now, they cannot learn previously encrypted messages. To achieve that, the adversary would need to learn prior shared secrets, which they cannot as that would require inverting the key derivation function. And even if an adversary compromised the shared secret now, clients recover from that compromise by updating the shared secret with new, uncompromised ephemeral key material.

To protect against "harvest now, decrypt later" adversaries, more and more end-to-end encryption protocols now integrate quantum-secure Key Encapsulation Mechanisms (KEMs) into the key establishment or double ratchet. Extensions are necessary because end-to-end encryption protocols traditionally rely on Elliptic-Curve Diffie-Hellman (ECDH) cryptography, which in turn relies on computational hardness assumptions that have been shown to not hold against quantum computers. Signal extended its handshake protocol X3DH to the quantum-secure version PQXDH [91], and a proposal to upgrade Signal's double ratchet algorithm to also provide quantum security has been published but not been deployed yet [58]. With the iMessage PQ3 protocol, Apple were the first to integrate KEMs into a double ratchet construction [2]. Their integration of KEMs into the double ratchet complicates its looping behavior even further, as it additionally adds a non-deterministic branch to the outer ratchet.

Doublet-ratchet constructions have become the state-of-the-art design for end-to-end encryption, with Signal, WhatsApp, and iMessage all relying on them. The Signal protocol, including its double ratchet, has been extensively studied in both the computational [8, 16, 23, 39, 48] and symbolic model [19, 22, 54, 88]. Curiously, however, no approach in the symbolic model has yet faithfully captured all steps of a double ratchet protocol. For example, no approach in the symbolic model captured Signal's inner ratchet, making it impossible to accurately prove all forward secrecy and post-compromise security guarantees provided by the protocol.

In Part ii, we present two ways of using induction to prove properties of looping protocols such as the double ratchet using the Tamarin prover. In Chapter 7, we first show how complex, looping protocols can be analyzed in their full complexity with modern, state-of-the-art symbolic provers by proving the iMessage PQ3 protocol secure. For that, we use Tamarin's trace induction mechanism, a well-established proof methodology, supported by Tamarin since its release and supported by ProVerif as well [26]. Our proofs capture all security guarantees provided by iMessage PQ3 without abstracting the protocol flow in any way. In particular, we prove that iMessage PQ3 provides complex and fine-grained forward secrecy and post-compromise security guarantees, and we consider a complex adversary model in which the adversary may gain access to a quantum-computer in the future.

As a second approach, we adapt *cyclic induction* to the security protocol domain in Chapter 8, and we show how it can be used to prove protocols like those using double ratchets secure while requiring considerably less effort than when using trace induction. Cyclic induction exploits recurring patterns in proofs and can thus elegantly tackle looping protocols. When using this new induction mechanism, Tamarin can prove many lemmas that previously required, often complex, auxiliary lemmas, and can prove new lemmas for protocols that were previously out of reach. We showcase our approach on a detailed model of the Signal protocol.

1.4 OUTLINE

We use the Tamarin prover in each of our chapters. We use it to formally analyze SOAP, ADEM, and iMessage PQ3 (Chapters 4, 5, and 7), and we extend it to support cyclic induction (Chapter 8). To prepare for all that, we introduce the Tamarin prover in Chapter 2.

We consider the problem of long-term key authentication in Part i, where we introduce and formally analyze SOAP and ADEM. In Part ii, we show how to formally analyze looping protocols in the symbolic model using both trace induction and cyclic induction. We draw conclusions in Chapter 9.

1.5 PUBLICATIONS

This doctoral thesis is based on the following publications.

- Felix Linker and David Basin, **"ADEM: An Authentic Digital EMblem,"** in Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS), 2023.
- Felix Linker and David Basin, **"SOAP: A Social Authentication Protocol,"** presented at the 33rd USENIX Security Symposium, 2024.
- Felix Linker, Ralf Sasse, and David Basin, **"A Formal Analysis of Apple's iMessage PQ3 Protocol,"** presented at the 34th USENIX Security Symposium, 2025.
- Felix Linker, Christoph Sprenger, Cas Cremers, and David Basin, **"Looping for Good: Cyclic Proofs for Security Protocols,"** to appear in in Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS), 2025.

In this thesis, we update the formal analysis of ADEM's accountability properties slightly in Chapter 5. Moreover, the paper presenting the formal analysis of iMessage PQ3 provides a general proof methodology for looping protocols, which we omit in our presentation of the work in Chapter 7.

1.6 ARTIFACTS

We provide a number of artifacts related to each of our contributions, which we list below.

SOAP: A SOCIAL AUTHENTICATION PROTOCOL A formal model and proofs are available at [69]. Prototypes are available at [144, 145]. A web-based prototype is accessible at: `https://soap-proto.net/`. A video demo of a Signal-based prototype is available at: `https://youtu.be/Ip_RAF8PRrM`.

ADEM: AN AUTHENTIC DIGITAL EMBLEM A formal model and proofs are available at [68]. A technical specification as referenced in this thesis is available at [5].

A FORMAL ANALYSIS OF APPLE'S IMESSAGE PQ3 PROTOCOL A formal model and proofs, as well as case studies and a pseudocode specification of iMessage PQ3 are available at [102].

CYCLIC INDUCTION FOR SECURITY PROTOCOL VERIFICATION The source code of a modified Tamarin implementation as well as case studies are available at [103].

2 THE TAMARIN PROVER

Tamarin [112, 139] is a model checker for security protocol verification, which incorporates a constraint solving algorithm based on symbolic backwards search. It was originally introduced in two doctoral theses [111, 138]. Later extensions added support for various features, for example, observational equivalence properties [13], the XOR operator [59], and a subterm predicate and natural numbers [53]. In this chapter, we introduce the Tamarin prover following [53, 111].

2.1 PRELIMINARIES

A *multiset* $\{a, b, \dots\}^\sharp$ is a set of items a, b, etc., where each item can occur multiple times. We denote a multiset's cardinality by $|M|$. We extend the usual operations and constants for sets to multisets using the $\sharp$ superscript, e.g., $\emptyset^\sharp$ is the empty multiset. Whenever we use relations, e.g., $\subseteq^\sharp$, or functions, e.g., $\cup^\sharp$, on multisets, they take cardinality into account. For example:

$$\{a, a\}^\sharp \subseteq^\sharp \emptyset^\sharp \cup^\sharp \{a, a\}^\sharp = \{a, a\}^\sharp$$
$$\{a, a\}^\sharp \not\subseteq^\sharp \{a, a\}^\sharp \setminus^\sharp \{a\}^\sharp = \{a\}^\sharp$$

A list $[a, b, \dots]$ is an ordered sequence of items. We sometimes write $\vec{L}$ to denote that a variable L is a list. We again write $|L|$ for a list's cardinality, and denote the set of a list L's *indices* by $idx(L) = \{1, \dots, |L|\}$. Lists can be indexed using subscript notation, for example, $[a, b]_1 = a$. We use $\mathrm{set}(X)$ to denote the set of all elements in a list or multiset X.

An *order-sorted signature* $\Sigma = (S, \leq, F)$ consists of a set of sorts S, a partial order $(\leq) \subseteq S \times S$ on the sorts, and a set of function symbols F. Every sort $s \in S$ has an infinite set of associated variables $\mathcal{V}_s$ and constants $\mathcal{C}_s$, which are all pairwise disjoint. We write $v{:}s$ for $v \in \mathcal{V}_s$ and $\mathcal{V} = \bigcup_{s \in S} \mathcal{V}_s$ for the set of all variables. Functions in F have the form $f : s_1 \times \cdots \times s_n \to s$ (where $s_1, \dots, s_n, s \in S$). We require that every connected component C in $(S, \leq)$ has a supremum, which we denote by $top(s)$ for $s \in C$. Furthermore, we require that if $f : s_1 \times \cdots \times s_n \to s \in F$ then $f : top(s_1) \times \cdots \times top(s_n) \to top(s) \in F$.

For $s \in S$, the terms of sort s, $\mathcal{T}_\Sigma^s$, are defined inductively as follows. Let $s_1, \ldots, s_n \in S$ and $t_1 \in \mathcal{T}_\Sigma^{s_1}, \ldots, t_n \in \mathcal{T}_\Sigma^{s_n}$. $\mathcal{T}_\Sigma^s$ is the smallest set such that:

$$\begin{array}{ll} v \in \mathcal{T}_\Sigma^s & \text{if } v \in \mathcal{V}_s \\ c \in \mathcal{T}_\Sigma^s & \text{if } c \in \mathcal{C}_s \\ f(t_1, \ldots, t_n) \in \mathcal{T}_\Sigma^s & \text{if } f : s_1 \times \cdots \times s_n \to s \in F \\ t \in \mathcal{T}_\Sigma^s & \text{if } t \in \mathcal{T}_\Sigma^{s'}, s' \leq s \end{array}$$

The set of all terms of an order-sorted signature is $\mathcal{T}_\Sigma = \bigcup_{s \in S} \mathcal{T}_\Sigma^s$. A term is *ground* if it contains no variables.

A substitution is function $\sigma : \mathcal{V} \to \mathcal{T}_\Sigma$ mapping variables to terms. We require that substitutions respect sorts, i.e., variables of sort s must be mapped to terms of sort $s' \leq s$. A substitution can be defined explicitly as a list of mappings, e.g., $\sigma = [a \mapsto b, c \mapsto d]$ substitutes a with b and c with d. Such a substitution is the identity on all variables not mapped in that list. The extension of a substitution σ by a new list of mappings is denoted by $\sigma[a \mapsto b]$. We extend substitutions homomorphically to terms and other constructs (to be defined later) as usual, and write $t\sigma$ for an application of σ to a term $t \in \mathcal{T}_\Sigma$. For example, for $t = f(a, b)$ and $\sigma = [a \mapsto g(c)]$, we have $t\sigma = f(g(c), b)$. A *valuation* is a substitution to ground terms. Valuations can also be defined as explicit lists. In that case, variables not mapped in that list are irrelevant and can map to arbitrary terms.

2.2 PROTOCOL SPECIFICATIONS

Tamarin works in the *symbolic model*, where protocol messages are represented as elements of a term algebra. An equational theory encodes the semantics of these terms. Along with the equational theory, the protocol and attacker behavior are specified by a set of multiset rewriting rules. A Tamarin protocol model is a pair of an equational theory and a set of multiset rewriting rules.

We will later extend Tamarin to support cyclic proofs in Chapter 8. We do not need the details of Tamarin's equational theories and adversarial reasoning to do so, and thus introduce equational theories only at a high level. For full details, we refer the reader to [53, 111].

2.2.1 *Equational Theories*

Protocol models are defined with respect to an order-sorted signature Σ and an *equational theory* E, a set of equations on terms admitted by the signature. Typically, the signature and equational theory model cryptographic operations and their relationships. For example, the following equation defines the symbolic model of symmetric encryption, formalizing that decrypting an encryption with the same key yields the original message: `sdec(senc(m, k), k) = m`.

In this thesis, we fix the sorts in Σ to *msg*, *fresh*, *pub*, and *nat*. *fresh*, *pub*, and *nat* are subsorts of *msg*. Sort *fresh* models fresh, unguessable values such as cryptographic random numbers. Sort *pub* models publicly known values, e.g., constants such as message tags or public parameters of protocols such as group generators. The sort *nat* was introduced in [53] for modeling natural numbers, and we base our definitions with respect to *nat* on that paper. Public constants are denoted by single-quoted strings, e.g., '*tag*' is a constant. There is only one constant for *nat*, which is %1. We sometimes prefix variables with a sort-specific symbol. Variables in *fresh* are prefixed with ~, variables in *pub* with $, and variables in *nat* with %. When not explicitly mentioned and not clear from context, a variable without prefix or sort annotation is of sort *msg*.

When modeling a theory in Tamarin, users define the protocol model's signature and equational theory E by importing pre-defined equational theories with associated signatures and by defining their own functions and equations. All reasoning about the protocol then happens modulo E. We use the subscript $\circ_E$ to indicate that an operator $\circ$, e.g., $=$ or $\in$, is considered modulo E.

User-defined functions cannot be arbitrary. For example, the equational theory must have the finite variant property, and functions f in Σ are always of the form $f : msg \times \cdots \times msg \to msg$ with the only exception being the addition of the natural numbers equational theory $+ : nat \times nat \to nat$ [53]. The restrictions allow Tamarin to efficiently decide whether the adversary knows certain terms, and Tamarin uses them, for example, to compute adversary deduction rules from a model's signature and equational theory.

2.2.2 *Multiset Rewriting Rules*

In Tamarin, state transitions are modeled using multiset rewriting rules. The global state consists of a multiset of ground facts, and the rewriting rules define how the state is updated. Facts are similar to predicates in first-order logic and are of the form $F(t_1, \ldots, t_n)$. F is a fact symbol from a fact signature Σ_{Fact}, and $t_1, \ldots, t_n$ are terms defined over the model's signature Σ. A fact is ground if all its terms are ground. Facts typically model the local state of participants or the network, e.g., who posses which keys, or what messages have been sent out on the network. We consider the special, reserved fact symbols Fr, In, Out, $!\mathsf{K}^{\uparrow}$, and $!\mathsf{K}^{\downarrow}$ of arity 1 in Σ_{Fact}. These symbols respectively model generating a fresh, unguessable value, receiving and sending a message over an insecure network, and adversary reasoning.

Multiset rewriting rules in Tamarin have the form

$$[l_1, \ldots, l_m] \; -[a_1, \ldots, a_n]\!\!\rightarrow \; [r_1, \ldots, r_o].$$

l_i, a_j, r_k are facts and may use free variables ($1 \leq i \leq m$, $1 \leq j \leq n$, $1 \leq k \leq o$). $l_1, \ldots, l_m$ are the rule's *premises*, $a_1, \ldots, a_n$ the rule's *actions* (also called *action facts*), and $r_1, \ldots, r_o$ the rule's *conclusions*. $[a_1, \ldots, a_n]$ can be omitted when a rule has no actions. Rules can be instantiated by a substitution σ. One rule ri is another rule r's *instance* if there exists a substitution σ such that $r\sigma = ri$. We write $prems(r)$, $acts(r)$, and $concs(r)$ for a rule r's list of premises, actions, and conclusions respectively. A rule (instance) is ground if all its facts are ground.

Facts are either persistent (preceded by a !) or linear. When a rule is applied, the linear facts in its premise are removed from the global state and replaced by the facts in its conclusion. Persistent facts are not consumed when used in a rule's premise. For example, the following rule models sending a nonce, encrypted using a symmetric key, and has no action facts as labels:

$$[\mathsf{Fr}(n), !\mathsf{Ltk}(k)] \rightarrow [\mathsf{Out}(senc(n, k))].$$

Here, the persistent fact !Ltk models storing a long-term key.

A set of multiset rewriting rules P is a *multiset rewriting system* if

(i) no rule uses a fresh constant,

(ii) no rule's premise contains an Out, $!\mathsf{K}^{\uparrow}$, or $!\mathsf{K}^{\downarrow}$ fact, and

(iii) no rule's conclusion contains a Fr, In, $!\mathsf{K}^{\uparrow}$, or $!\mathsf{K}^{\downarrow}$ fact, and

(iv) all *fresh*, *msg*, and *nat* variables used in a rule's conclusion are introduced in a rule's premise.

Example 1. Consider the following model, which we introduce in Tamarin's source file syntax. Tamarin's source file syntax closely resembles the mathematical notation introduced so far, and we will use both notations interchangeably.

```
rule KeyGen:
  [ Fr(~k) ]
  --[ KeyGen($A) ]->
  [ !Ltk($A, ~k) ]

rule KeyReveal:
  [ !Ltk($A, ~k) ]
  --[ Compromised($A) ]->
  [ Out(~k) ]

rule Encrypt:
  [ Fr(~msg), !Ltk($A, ~k) ]
  --[ Send($A, ~msg) ]->
  [ Out(senc(~msg, ~k)) ]
```

This model consists of three rules, `KeyGen`, `KeyReveal`, and `Send`. These rules model a party, identified by the public variable $\$A$, generating

a symmetric key and sending a message symmetrically encrypted under that key. The message and key are modeled by the fresh variables *~msg* and *~k*. Additionally, a party $\$A$'s key may become compromised by revealing their key to the adversary.

Built-In Rules

Tamarin uses a number of built-in rules to model the generation of random values and the adversary deduction. The following rules model generating fresh values:

$$\textsc{Fresh} := [] \rightarrow [\mathsf{Fr}(x{:}\mathit{fresh})].$$

Tamarin models adversary deduction by precomputing adversary message deduction rules when a model is loaded in the prover. The precise rules available to the adversary depend on the model's signature Σ and equational theory E. All adversary deduction rules use three, unary facts: K, modeling adversary knowledge, $!\mathsf{K}^{\downarrow}$ (also $!\mathsf{KD}$), modeling adversary deconstruction, and $!\mathsf{K}^{\uparrow}$ (also $!\mathsf{KU}$), modeling adversary construction. Furthermore, all rules respect certain normal form condititions with respect to the equational theory E. We refer to the set of all adversary deduction rules as ND (normal-form message deduction rules).

We do not introduce how adversary deduction rules are computed and which normal form conditions they respect. However, to provide an intuition for how Tamarin's adversary deduction rules work, we showcase some next. The adversary can learn messages sent over the insecure network via the Irecv rule:

$$[\mathsf{Out}(x)] \rightarrow [!\mathsf{K}^{\downarrow}(x)].$$

The adversary then can apply deconstruction rules using $!\mathsf{K}^{\downarrow}$ facts to decompose the term x into its constituents following the equations available in the model's equational theory E. For example, the adversary can access a tuple's first element using the following rule:

$$[!\mathsf{K}^{\downarrow}(\langle x, y\rangle)] \rightarrow [!\mathsf{K}^{\downarrow}(x)].$$

Using the Coerce rule, the adversary can transition from $!\mathsf{K}^{\downarrow}$ to $!\mathsf{K}^{\uparrow}$:

$$[!\mathsf{K}^{\downarrow}(x)] \; -[!\mathsf{K}^{\uparrow}(x)]\rightarrow \; [!\mathsf{K}^{\uparrow}(x)].$$

Using $!\mathsf{K}^{\uparrow}$ facts, the adversary can construct more terms using the functions available in the model's signature Σ. For example, given two elements, the adversary can construct a tuple:

$$[!\mathsf{K}^{\uparrow}(x), !\mathsf{K}^{\uparrow}(y)] \; -[!\mathsf{K}^{\uparrow}(\langle x, y\rangle)]\rightarrow \; [!\mathsf{K}^{\uparrow}(\langle x, y\rangle)].$$

Finally, the adversary can send a message over the insecure network to a protocol participant using the Isend rule:

$$[!\mathsf{K}^{\uparrow}(x)] \; -[\mathsf{K}(x)]\rightarrow \; [\mathsf{In}(x)].$$

Observe that this rule uses the action fact K.

Example 2. Consider again the model from Example 1. If the adversary learned a participant's long-term encryption from the rule `KeyReveal`, they could learn all messages sent by that participant encrypted under that key. In Tamarin, the action fact K in the ISEND rule formalizes that the adversary learned a term. To learn the plaintext, the intuition is that the adversary constructs the term $sdec(senc(\sim msg, \sim k), \sim k)$, which reduces to the plaintext $\sim msg$. Formally, Tamarin will apply the following sequence of multiset rewriting rules. Presume, the rules `KeyReveal` and `Encrypt` were applied so that the respective Out and !Ltk facts are in the state.

$$
\begin{aligned}
& \{\mathsf{Out}(senc(\sim msg, \sim k)), \mathsf{Out}(\sim k), !\mathsf{Ltk}(\sim k)\}^\sharp \\
\longrightarrow\ & \{!\mathsf{K}^\downarrow(senc(\sim msg, \sim k)), !\mathsf{K}^\downarrow(\sim k), \dots\}^\sharp && \textsc{Irecv} \\
\longrightarrow\ & \{!\mathsf{K}^\downarrow(\sim msg), \dots\}^\sharp && \text{(precomputed)} \\
\longrightarrow\ & \{!\mathsf{K}^\uparrow(\sim msg), \dots\}^\sharp && \textsc{Coerce}, !\mathsf{K}^\uparrow(\sim msg) \\
\longrightarrow\ & \{\mathsf{In}(\sim msg), \dots\}^\sharp && \textsc{Isend}, \mathsf{K}(\sim msg)
\end{aligned}
$$

In each line, we only reference newly added facts and omit old persistent facts. We annotate the rule name and action facts added to the trace on the right side. Tamarin generates the rule reducing $!\mathsf{K}^\downarrow(senc(\sim msg, \sim k))$ and $!\mathsf{K}^\downarrow(\sim k)$ to $!\mathsf{K}^\downarrow(\sim msg)$ from the *sdec* equation during a precomputation phase.

2.3 SEMANTICS

2.3.1 *Executions and Traces*

Given a multiset rewriting system P, adversary deduction rules ND, an equational theory E, and $R = P \cup ND \cup \{\textsc{Fresh}\}$, the tuple (R, E) is a *Tamarin* or *protocol model*. Tamarin models induce a labeled state transition system. The initial state is the empty multiset $\emptyset^\sharp$. A state transition is the application of a ground rule instance $ri = l -[a]\!\!\rightarrow r$. We use $\mathsf{lfacts}(x)$ and $\mathsf{pfacts}(x)$ to denote the linear and persistent facts respectively in x. ri can be applied if its premises are contained in in the global state, which is then updated accordingly. Formally, the state transition relation $\Rightarrow_{(R,E)}$ is defined as:[1]

$$S \Rightarrow_{(R,E)} (S \setminus^\sharp \mathsf{lfacts}(l)) \cup^\sharp r \quad \text{if } \mathsf{lfacts}(l) \subseteq^\sharp S, \mathsf{pfacts}(r) \subseteq \mathsf{set}(S)$$

An *execution* is a sequence of labeled state transitions. We only consider executions where each instance of FRESH is unique. Each

[1] Note that r is technically a list, but we understand the state update to add r as multiset to S.

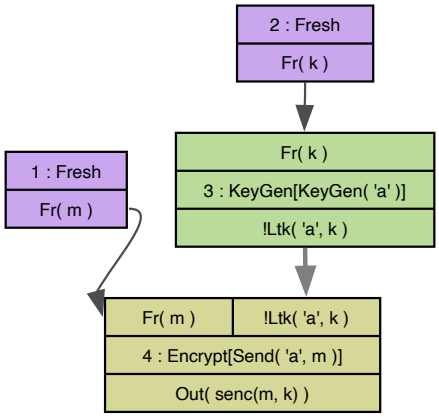


Figure 2.1: Dependency graph

execution has a corresponding *trace*, which is the list of sets of action facts labeling each transition. The semantics of a Tamarin model is its set of traces.

2.3.2 *Dependency Graphs*

Dependency graphs capture the sequence of rule applications and dependencies between rule premises and conclusions. They are closely related to the constraint systems used in Tamarin's protocol analysis, and they enable effective proof search. Figure 2.1 shows a dependency graph for the theory shown in Example 1. Here, n and k are fresh constants, and 'a' is a public constant. Every node corresponds to a ground rule instance with up to three parts. The upper part are the rule's premises, the lower part the rule's conclusions, and the middle part the rule's *timepoint*, name, and action facts. When a rule has no premises or conclusions, these parts are omitted. The timepoints reflect at which point in time a rule was applied.

Definition 1 (Dependency graph)**.** Let (R, E) be a Tamarin model. A *dependency graph* is a tuple $dg = (I, D)$ where I is a finite list of ground rule instances in R, and $D \subseteq \mathbb{N}^2 \times \mathbb{N}^2$. A rule's index in I is its *timepoint*. We write $(i, u) \rightarrowtail (j, v)$ for $((i, u), (j, v)) \in D$, when D is clear from the context. Here, (i, u) denotes the conclusion with index u of the rule with index i in I. Similarly, (j, v) denotes the premise with index v of the rule with index j. We require that edges correctly connect premises and conclusions, i.e., $i < j$ and the conclusion (i, u) is equal modulo E to the premise (j, v). Furthermore, each premise must have exactly one incoming edge, every linear conclusion has at most one outgoing edge, and instances of FRESH are unique.

Definition 2 (Trace)**.** A *trace* induced by a dependency graph $dg = (I, D)$ is a list $trace(dg)$ of sets of action labels. Concretely, $trace(dg)$ is

of length I and every index i corresponds to the set of action labels at timepoint i, i.e., to $\mathrm{set}(\mathit{acts}(I_i))$.

One can show that the dependency graphs of a model (R, E) induce the same set of traces as the set of traces derived from the model's executions ([111, Theorem 3]). The trace of the dependency graph shown in Figure 2.1 is:

$$[\emptyset, \emptyset, \{\mathsf{KeyGen}(\text{`}a\text{'})\}, \{\mathsf{Send}(\text{`}a\text{'}, m)\}]\,.$$

To reason about adversary deduction in a dependency graph $dg = (I, D)$, we define the *deconstruction-chain relation* $(\dashrightarrow_{dg}) \subseteq \mathbb{N}^2 \times \mathbb{N}^2$. The deconstruction rule tracks which terms the adversary deconstructed to learn some other term. The relation is defined as the smallest relation such that $c \dashrightarrow_{dg} p$ if c is a $!\mathsf{K}^{\downarrow}$-conclusion in dg and either:

(i) $c \rightarrowtail p \in D$, or

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

2.4 PROTOCOL PROPERTIES

In Tamarin, protocol properties are expressed as first-order logic *trace properties*, which are given by closed *trace formulas*. To reason about temporal ordering of events, we introduce a new sort *tmp*, incomparable to *msg*, for temporal values and variables. The constants of *tmp* are the elements of $\mathbb{Q}$.

For a fact f, temporal variables i and j representing timepoints, and terms t and u, atomic trace formulas are:

1. false $\bot$,
2. action formulas $f@i$, and
3. the predicates for
   (a) timepoint equality $i \doteq j$,
   (b) timepoint ordering $i < j$,
   (c) term equality $t = u$, and
   (d) subterm relation $t \sqsubset u$ (also written $t \ll u$; introduced in [53]).

Trace formulas can be combined with the logical connectives $\neg$, $\wedge$ and *fresh* and *msg* variables can be existentially quantified. As usual, we use $\vee$, $\Longrightarrow$ and universal quantification as derived connectives. We write $\mathit{fv}(\varphi)$ for the free variables and $\mathit{fv}_{\mathit{tmp}}(\varphi)$ for the free temporal variables of a formula φ.

We assume that all trace formulas are *guarded*. This means that all formulas using existential or universal quantifiers are of the form $\exists \vec{x}.\, f@i \wedge \varphi$ or $\forall \vec{x}.\, f@i \implies \varphi$, where $\mathrm{set}(\vec{x}) \subseteq fv(f@i)$, i.e., all bound variables are free in the action formula $f@i$. Intuitively speaking, guardedness requires that all instantiations of quantified variables are related to a protocol execution (and its trace) via some action fact $f@i$. We write $\widehat{\varphi}$ for $\neg\varphi$'s negation normal form.

Trace properties to be proven in Tamarin are called *lemmas*. For example, the following lemma expresses a property of the theory introduced in Example 1. It models that a message m sent by a party p either stays secret (there exists no action fact K on the trace), or p revealed its key to the adversary:

$$\forall p, m, j.\, \mathsf{Send}(p, m)@j \implies (\neg\exists x.\, \mathsf{K}(m)@x) \vee (\exists x.\mathsf{Compromised}(p)@x).$$

Given a valuation θ of the free variables of φ, we write $(tr, \theta) \models_E \varphi$ to mean that the trace tr satisfies φ under the valuation θ. Note that we interpret variables of sort s in the set of ground terms of sort s. As all functions in Σ (except for $+$) are of sort *msg*, this means that variables of sort *msg* and *nat* are interpreted as ground terms, and *fresh*, *tmp*, and *pub* as constants. The relation $\models_E$ is defined as follows.

$(tr, \theta) \models_E \bot$	never
$(tr, \theta) \models_E f@i$	if $\theta(i) \in idx(tr) \wedge f\theta \in_E tr_{\theta(i)}$
$(tr, \theta) \models_E i < j$	if $\theta(i) < \theta(j)$
$(tr, \theta) \models_E i \doteq j$	if $\theta(i) = \theta(j)$
$(tr, \theta) \models_E t_1 = t_2$	if $t_1\theta =_E t_2\theta$
$(tr, \theta) \models_E t_1 \sqsubset t_2$	if t_1 is a strict subterm of t_2, for details see [53]
$(tr, \theta) \models_E \neg\varphi$	if not $(tr, \theta) \models_E \varphi$
$(tr, \theta) \models_E \varphi \wedge \psi$	if $(tr, \theta) \models_E \varphi$ and $(tr, \theta) \models_E \psi$
$(tr, \theta) \models_E \exists x{:}s.\varphi$	if for a ground term $u \in \mathcal{T}^s_\Sigma$, $(tr, \theta[x \mapsto u]) \models_E \varphi$

A trace formula φ is *valid* for a protocol model (R, E), written $R \models^{\forall}_E \varphi$, if for all traces tr of (R, E) and every valuation θ, one has $(tr, \theta) \models_E \varphi$. φ is *satisfiable* in (R, E), written $R \models^{\exists}_E \varphi$, if there exists a trace tr of (R, E) and valuation θ such that $(tr, \theta) \models_E \varphi$.

2.5 CONSTRAINT SYSTEMS

To verify that a trace property φ is valid for a model (R, E), Tamarin negates the property and searches for an attack, i.e., a dependency graph satisfying the negated property. If none is found, this proves that the property holds.

2.5.1 *Syntax*

Tamarin uses constraint systems to symbolically describe sets of dependency graphs, i.e., potential attacks. There are five types of constraints:

1. Formula φ: Trace formula φ must hold.
2. Node $i : ri$: Rule ri is applied at timepoint i.
3. Edge $(i, u) \rightarrowtail (j, v)$: The fact at index u of node i's conclusion is used for node j's premise at index v.
4. Premise $f \triangleright_v i$: Node i has fact f as its premise at index v. This constraint is used to generate new node constraints and to connect their conclusions to the respective premises using edge constraints. We call unsolved premise constraints *open premises*.
5. Chain $(i, u) \dashrightarrow (j, v)$: The adversary learned the term at node j's premise v by deconstructing the term at node i's conclusion u, possibly transitively. This constraint reasons about which messages an adversary may have learned specific terms from.

We extend the predicate $=$ to also apply to facts and rule instances. For example, for two facts $f(t_1, \ldots, t_n)$ and $f'(u_1, \ldots, u_m)$ the constraint $f(t_1, \ldots, t_n) = f'(u_1, \ldots, u_m)$ is equal to $\bot$ if $f \neq f'$, and $\{t_1 = u_1, \ldots, t_n = u_n\}$ otherwise (observe that $n = m$ in this case).

A *constraint system* Γ is a set of constraints over a given protocol model (R, E). Let $\mathbb{C}$ be the set of all constraint systems defined over the given protocol. We extend the functions $fv(\cdot)$ and $fv_{tmp}(\cdot)$ from formulas to constraint systems as expected.

2.5.2 *Semantics*

A *model* for a constraint system Γ is a pair (dg, θ) consisting of a dependency graph $dg = (I, D)$ and a valuation θ of Γ's free variables. We write $(dg, \theta) \Vdash_E \Gamma$ if such a model satisfies all constraints in Γ. We define the relation $\Vdash_E$ as the smallest relation such that:

$(dg, \theta) \Vdash_E \varphi$	if $(trace(dg), \theta) \models_E \varphi$
$(dg, \theta) \Vdash_E i : ri$	if $\theta(i) \in idx(I) \wedge ri\theta =_E I_{\theta(i)}$
$(dg, \theta) \Vdash_E (i, u) \rightarrowtail (j, v)$	if $(\theta(i), u) \rightarrowtail (\theta(j), v) \in D$
$(dg, \theta) \Vdash_E f \triangleright_v i$	if $\theta(i) \in idx(I) \wedge f\theta =_E prems(I_{\theta(i)})_v$
$(dg, \theta) \Vdash_E (i, u) \dashrightarrow (j, v)$	if $(\theta(i), u) \dashrightarrow_{dg} (\theta(j), v)$

A dependency graph dg is a *solution* for Γ if there exists a valuation θ such that $(dg, \theta) \Vdash_E \Gamma$. We write $sols(\Gamma)$ for the set of all solutions of Γ.

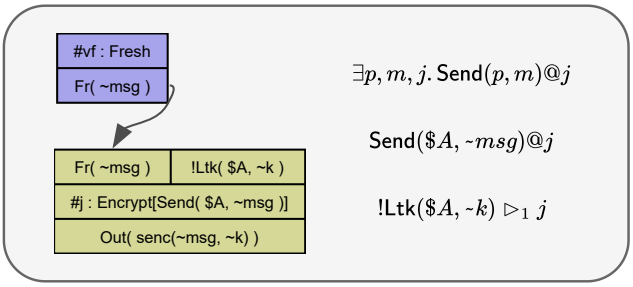


Figure 2.2: Example constraint system

Example 3. Tamarin can visualize constraint systems, and Figure 2.2 shows an example constraint system with two node, one edge, one premise, and two formula constraints. These constraints are satisfied, for instance, by the dependency graph in Figure 2.1 under the valuation $\theta = [vf \mapsto 0, j \mapsto 3, \sim msg \mapsto m, \sim k \mapsto k, \$A \mapsto \text{`}a\text{'}]$.

Observe that we defined two logical relations for trace formulas. We write $(tr, \theta) \models_E \varphi$ if a trace tr satisfies φ, and $(dg, \theta) \mid\!\models_E \varphi$ if a dependency graph dg satisfies φ. There is a close connection between $\models_E$ and $\mid\!\models_E$, namely, for any protocol model (R, E) and trace formula φ, there exists a trace tr and valuation θ such that $(tr, \theta) \models_E \varphi$ if and only if there exists a dependency graph dg and valuation θ such that $(dg, \theta) \mid\!\models_E \{\varphi\}$, i.e., $sols(\{\varphi\})$ is non-empty. This reduces the formula satisfaction problem to a constraint solving problem.

2.5.3 *Timepoints and Temporal Order*

Each constraint system Γ induces a *temporal order* $\prec_\Gamma$ on its temporal variables, which is defined as the smallest transitive relation such that $i \prec_\Gamma j$ if:

(i) $i < j \in \Gamma$, or

(ii) $\exists u, v.\, (i, u) \rightarrowtail (j, v) \in \Gamma$, or

(iii) $\exists u, v.\, (i, u) \dashrightarrow (j, v) \in \Gamma$.

We write $\preceq_\Gamma$ for the reflexive closure of $\prec_\Gamma$. Note that timepoints in dependency graphs are linearly ordered (as they are in $\mathbb{N}$), whereas a constraint system's temporal order only partially orders temporal variables, reflecting that the execution order of certain rules may be irrelevant (e.g., two parallel protocol runs can be arbitrarily interleaved). They are related as follows.

Lemma 1 ([111, Lemma 7])**.** Suppose $(dg, \theta) \mid\!\models_E \Gamma$. Then:

(i) if $i \prec_\Gamma j$ then $\theta(i) < \theta(j)$ and

(ii) if $i \preceq_\Gamma j$ then $\theta(i) \leq \theta(j)$.

2.6 CONSTRAINT REDUCTION AND DERIVATION TREES

We are now prepared to explain the idea behind Tamarin's proof methodology. To prove that $R \models_E \varphi$ holds for a protocol model (R, E), Tamarin uses a tree-search, constraint-solving algorithm. Every node in the search tree corresponds to a constraint system. Initially, the tree only contains a root constraint system $\{\widehat{\varphi}\}$, which transforms proving the validity of φ into proving the unsatisfiability of $\neg\varphi$, i.e., into an attack search problem.

2.6.1 *Constraint Reduction Rules*

Constraint reduction rules refine the search tree's leaf constraint systems and are of the form

$$\Gamma \rightsquigarrow \{\Gamma_1, \ldots, \Gamma_n\}\,. \tag{2.1}$$

Such a rule encodes that the constraint system Γ can be solved by solving any of the constraint systems $\Gamma_1, \ldots, \Gamma_n$. We will often write concrete reduction rules as $\Gamma \rightsquigarrow \Gamma_1 \parallel \ldots \parallel \Gamma_n$, where the cases are easier to distinguish. Given constraint systems Γ and Δ, we write (Γ, Δ) for $\Gamma \cup \Delta$ and use constraints as singleton constraint systems when clear from context, for example, φ stands for $\{\varphi\}$.

To apply a constraint reduction rule to a constraint system, the rule's free variables must be instantiated with a substitution so that the constraint reduction rule matches the constraint system. We therefore close the relation $\rightsquigarrow$ under substitutions, that is, $\Gamma\sigma \rightsquigarrow \{\Gamma_1\sigma, \ldots, \Gamma_n\sigma\}$ for all substitutions σ and rules (2.1).

There are different types of constraint reduction rules. For instance, there are rules that

1. work on formula connectives, similar to the deduction rules of first-order logic,
2. introduce new node constraints from $f@i$ constraints,
3. backwards-complete premise constraints $f \rhd_v i$, by introducing new nodes and edges connected to open premises,
4. enforce the single use of linear facts, or
5. derive contradictions, written as $\Gamma \rightsquigarrow \bot$.

We list Tamarin's constraint reduction rules in Appendix A.

Example 4. Consider the two constraint reduction rules $\mathcal{S}_@$ and $\mathcal{S}_{\neg,@}$. The former adds a node for an action formula $f@i$ and performs a case split over all rule instances that could introduce the respective action fact. The latter derives a contradiction when there is an action

fact but also an action formula that requires this fact to not be in the constraint system. This typically happens when a formula requires certain events, such as the reveal of a secret key, to not occur.

$\mathcal{S}_{@} : \Gamma \rightsquigarrow \|_{ri \in R} \|_{f' \in acts(ri)} (i : ri, f = f', \Gamma)$
if $(f@i) \in \Gamma$ and $f \neq !K^{\uparrow}(t)$ and $f@i \notin_E as(\Gamma)$.

$\mathcal{S}_{\neg,@} : \Gamma \rightsquigarrow \bot$
if $\neg(f@i) \in \Gamma$ and $(f@i) \in_E as(\Gamma)$.

2.6.2 *Derivation Trees and Proofs*

When a constraint reduction rule (2.1) is applied to a leaf Γ of the search tree, the constraint systems Γ_1 to Γ_n become that node's children. We formally define Tamarin's search tree as a *derivation tree*.

Definition 3 (Derivation Trees)**.** A *derivation tree* $\mathcal{D} = (\mathcal{N}, \mathcal{E}, \gamma)$ for a formula φ is a tree $(\mathcal{N}, \mathcal{E})$ with nodes $\mathcal{N}$ and edges $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ together with a function $\gamma : \mathcal{N} \to \mathbb{C}$ labeling each node with a constraint system such that

- $\gamma(v_0) = \{\widehat{\varphi}\}$ for the root $v_0 \in \mathcal{N}$ of $\mathcal{D}$, and
- for each non-leaf node v, there is a constraint reduction rule with $\gamma(v) \rightsquigarrow \{\gamma(v') \mid (v, v') \in \mathcal{E}\}$.

A leaf v is called an *axiom leaf* if $\gamma(v) = \bot$ (i.e., $\gamma(v)$ is contradictory) and an *open leaf* otherwise.

Tamarin's constraint reduction algorithm terminates when it finds a *proof*, i.e., all leaves are axioms, or an *attack*, i.e., there exists a leaf constraint system that is *solved*. Solved constraint systems Γ are sufficiently constrained to enable the extraction of a model $(dg, \theta) \Vdash_E \Gamma$ [111, Theorem 5]. In particular, such a constraint system contains no node constraints with an open premise, which allows the extraction of a dependency graph. For theories that use the subterm operator $\sqsubset$ or reducible function symbols such as XOR, it can also terminate with *unfinishable*, i.e., the result is unknown.

A constraint reduction rule of the form (2.1) is *sound* if no solutions are lost, i.e., $sols(\Gamma) \subseteq \bigcup_i sols(\Gamma_i)$, and *complete* if no solutions are added, i.e., $sols(\Gamma) \supseteq \bigcup_i sols(\Gamma_i)$.[2] Since Tamarin's constraint reduction rules are both sound and complete, a proof of φ implies $R \models_E \varphi$ whereas a reduction sequence leading to a solved form implies $(dg, \theta) \Vdash_E \{\widehat{\varphi}\}$ for some dependency graph dg and valuation θ.

[2] We use the proof-theoretic definitions of these terms here, whereas in [111] the former relation is called completeness and the latter is called correctness.

Well-Formedness Constraints

Tamarin's constraint systems must satisfy the well-formedness conditions **WF1-5** presented in [111, p.99 and p.120]. Given a protocol model (R, E), these conditions can be summarized as follows:

WF1. All node constraints in Γ are instances of a multiset rewrite rule from R,

WF2. All premise constraints $f \rhd_v i \in \Gamma$ correctly refer to a premise of a node constraint in Γ,

WF3. All edge constraints $(c \rightarrowtail p) \in \Gamma$ connect the conclusion c of some node instance in Γ to the premise p of another node instance in Γ,

WF4. All subformulas $\forall \vec{x}.\, \omega$ of any trace formula $\varphi \in \Gamma$ are of the form $\omega = f@i \implies \psi$ such that $\mathrm{set}(\vec{x}) \subseteq fv(f@i)$.

WF5. All chain constraints $(c \dashrightarrow p) \in \Gamma$ connect the $!\mathsf{K}^{\downarrow}$ conclusion c of some node instance in Γ to the $!\mathsf{K}^{\downarrow}$ premise p of another node instance in Γ.

By inspection of the constraint reduction rules $\mathcal{S}_\forall$ and $\mathcal{S}_\exists$ for quantifiers, it is easy to see that **WF4** can be strengthened to

WF4′. All formulas $\varphi \in \Gamma$ are guarded trace formulas.

Part I

PROVABLE LONG-TERM KEY AUTHENTICATION

3 LONG-TERM KEY AUTHENTICATION

There is no secure communication without long-term key authentication. Many systems aiming to provide authentic long-term keys, however, relied on trust assumptions that were shown to often not hold in practice. As a consequence, alternative approaches were suggested. In this part of the thesis we will present, use, and formalize such approaches. To start, we present a formal notion of authentication and show how it relates to long-term key authentication (Section 3.1). We then sketch the two alternative approaches: social authentication (Section 3.2) and accountability (Section 3.3).

In later chapters, we present two novel systems that follow each of these approaches and provide provable social authentication and accountability guarantees. We present SOAP, a SOcial Authentication Protocol, which provides social authentication, in Chapter 4 and ADEM, an Authentic Digital EMblem, which provides authentication using an accountability mechanism, in Chapter 5.

3.1 TRUE LONG-TERM KEY AUTHENTICATION

We formalize long-term key authentication by instantiating *agreement* properties as defined by Lowe. Lowe famously introduced a "A Hierarchy of Authentication Specifications" [107], in which he defines five, increasingly strong authentication properties that aim to capture various notions of authentication. Of these five properties, we introduce *non-injective agreement* and *injective agreement*. Both properties guarantee that when a protocol P was run between two parties A and B, both parties agree on their view of that run. In particular, they agree to have run P with each other and they agree on a set of data items ds that were exchanged while running P. The properties differ in that injective agreement requires a one-to-one (injective) mapping from sessions of A to sessions of B, which formalizes that P provides replay protection. Following Lowe, we slightly recast both properties as follows.

Security Property (Non-injective agreement; adapted from [107])**.** A protocol guarantees an initiator A *non-injective agreement* if whenever A receives a message with data items ds, apparently from responder B, then B was previously running the protocol as the responder, and the two agents agree on the values of ds.

Security Property (Injective agreement; adapted from [107])**.** A protocol guarantees an initiator A *injective agreement* if whenever A receives a message with data items ds, apparently from responder B, then B

was previously running the protocol as the responder, and the two agents agree on the values of *ds*, and each run of A corresponds to a unique run of B.

In contrast to the definitions above, [107] also requires that A and B agree on the intended recipient (A), which we drop so that we can also consider only sender-authenticated protocols. Most protocols that provide agreement as defined above and run between two parties provide agreement on both participants A and B and in both directions, i.e., in the stronger sense. However, long-term key authentication typically does not require receiver-authentication as identity to public key bindings are public.[1]

We formalize long-term key authentication as an agreement property. A protocol P provides long-term key authentication for a responder B to an initiator A if (i) the data items *ds* encode B's public key, and (ii) for all runs between two honest parties A and B, P guarantees A agreement. To illustrate this, we sketch how a long-term key authentication protocol provided by all modern secure messengers provides injective agreement.

All modern messengers, including Facebook Messenger [42], iMessage [158], Signal [119], Telegram [155], Threema [164], Viber [160], and WhatsApp [4], offer their users to compare *safety numbers* with their chat partners, which is called an *authentication ceremony*. Safety numbers are fingerprints of the two conversation participants' public keys. To compare safety numbers in Signal's Android app, for example, users can use the safety number verification menu, which displays the safety number as a QR code and numerically. Users can compare safety numbers either by scanning each other's QR codes in-person or by relying on a different, trusted out-of-band channel. If two chat partners' safety numbers match, they are using the same public keys, and as they rely on a trusted channel such as in-person comparison, they know that they use each other's authentic public keys. The Signal app provides a button to mark contacts as verified whenever safety number comparison was successful.

One can consider the in-person comparison of safety numbers as a protocol with the following steps:

1. A (the initiator) opens the safety number verification menu on their device and asks B (the responder) to do the same.

2. By opening the menu, B "sends" A their view on each other's long-term keys, visually encoded in the QR code. Here, the data items *ds* contain A's and B's safety number and, hence, B's public key.

[1] Identity to public key bindings are typically public for the purposes of authentication. For privacy reasons, one may want to share one's public key selectively, but receivers would not need to authenticate such mechanisms.

3. A "receives" B's message by scanning their QR code. If the comparison performed by the app is successful, A marks B as verified in their application.

It is easy to see that the above protocol provides injective agreement and thus long-term key authentication. The safety number comparison provides agreement, and the fact that it happens physically assures that the agreement is injective. One cannot replay physical actions. Moreover, the in-person comparison of safety numbers provides particularly strong long-term key authentication as it relies on minimal trust assumptions. Its limitations, however, should be apparent. Meeting in person is a high bar, and it does not apply to many other use cases.

3.2 KEY AUTHENTICATION USING SOCIAL AUTHENTICATION

The first alternative approach to true long-term key authentication is to use *social authentication*. The idea of social authentication is for a verifier to authenticate a prover's long-term key using one or more digital identities at IdPs. Instead of verifying that one truly communicates with the intended peer, one verifies that the peer controls a number of digital identities. This idea was pioneered by Keybase [86] in context of secure messaging. Keybase allows its users to bind their social media accounts, e.g., at Twitter, to their Keybase account using so-called *proofs* [87]. Users do this by posting a signed message on a social media platform. Other users can verify that a user linked their accounts by checking that the signature was generated using the key associated with the Keybase account and posted by the claimed social media account.

Later, [159] coined the term "social authentication," proposed its application to Signal, and conducted a user study using a mock-up prototype. The authors found that users regard social authentication as understandable, easy to use, working asynchronously and remotely (in contrast to, e.g., in-person verification), and that it enhances their security when using multiple providers. However, users gave social authentication a lower trust score than in-person verification, partially stemming from their limited understanding of the mechanism. For example, users feared that a compromise of their social media accounts could lead to a compromise of their Signal account, and they distrusted social media providers in general. Users also mentioned the risk of social engineering with fake accounts.

Social authentication is appealing as: (i) many users have pre-existing relationships on social media, and (ii) by linking their social media presence to a different account, they can transfer the relationship from one medium to another. Note that these two notions are independent: even if you have no relationship with a given social media account, you could still be convinced that you are talking to the account holder. Moreover, social authentication allows for authentication in useful

ways, that are impossible using conventional solutions, which we explicate in two further use cases.

SOCIAL AUTHENTICATION AS A SECOND FACTOR One may question the value of social authentication whenever users cannot authenticate their chat partner's identities because they have no relationship to these identities. For example, someone who has never received an e-mail from a given Outlook address would be unable to verify that e-mail address as truly belonging to the person in question without further interaction.

In such cases, social authentication is still valuable as it can serve as a second factor, raising the bar for compromise. If one of your contacts authenticated themselves as in control of two accounts, and you are prompted that this contact's public key changed, you can check whether the "new" contact still controls both these accounts. This information can help to distinguish a public key maliciously associated to your contact's profile from a legitimate, fresh public key after key-rollover.

NATIVE DIGITAL AUTHENTICATION For some online interactions, users do not base the identification of their chat partners in the physical world, but rather in the digital world. For example, in the physical world, I might like to authenticate my chat partner as "my colleague Alice, who I eat lunch with every day." In contrast, in the digital world, I might like to authenticate my chat partner as "the open-source maintainer Alice123 on GitHub, who I have never met in real life but writes beautiful JavaScript." In the latter case, social authentication promises to seamlessly bootstrap a secure communication channel from such a pre-existing relationship.

So far, however, social authentication has never been studied formally. It has never been properly defined, e.g., as a trace property, and as a consequence, it has never been studied whether a design promising social authentication actually provides it. In Chapter 4, we fill this gap. We formally define social authentication and present SOAP, a SOcial Authentication Protocol, which provably social authentication.

3.3 KEY AUTHENTICATION USING ACCOUNTABILITY

The second alternative approach to true long-term key authentication is to use an *accountability* mechanism. Protocols providing accountability do not (generally) prevent attacks but allow one to identify and punish participants that do not follow the protocol. Utilizing accountability mechanisms to provide long-term key authentication was proposed in *transparency systems*. In transparency systems, *log operators* maintain publicly verifiable *logs*, which are authenticated data structures that commit to the identity-to-key bindings attested

to by *providers*, and identity-to-key bindings only become valid when they are included in one or more logs.

In context of the Web PKI, *Certificate Transparency (CT)* [96, 97, 149] holds CAs accountable. In context of secure messaging, *key transparency* systems hold messaging providers accountable (see, for example, [41, 99, 100, 108, 113, 165]), and both iMessage [1] and WhatsApp [101] provide key transparency services. Different transparency systems specify different security and operational details. CT, for example, specifies that certificates must be included in at least two logs from different log operators, whereas key transparency logs are typically operated by the messaging provider itself. Some designs require third-party auditors that monitor log providers to maintain certain invariant, such as providing an append-only authenticated data structure. They all require, though, that parties controlling a given pseudonym monitor the log for the public keys bound to their identity, and take action should a provider ever lie about their public key. Only this monitoring enables holding providers accountable should they ever lie about identity-to-key bindings.

In Chapter 5, we present how accountability mechanisms, such as provided by the systems above, can solve a unique and novel problem that arises in times of armed conflict. ADEM, an Authentic Digital EMblem, is a system for *digital emblems* that mark digital infrastructure as protected under international humanitarian law (IHL), just like the physical emblems of the Red Cross, Red Crystal, and Red Crescent mark physical infrastructure as protected. During armed conflict, all these markings signal that the respectively marked assets enjoy special protection under IHL and thus should not be attacked.

Digital emblems have unique security requirements. First, digital emblems must be authentic. Digital assets must not be able to illegitimately signal protection, for example, by moving legitimate digital emblems to different contexts. Second, digital emblems must provide the novel security requirement *covert inspection*. Covert inspection requires that those inspecting digital emblems must not reveal themselves as such. As digital emblems are signs to stop attacks, those looking for them likely intend to attack unmarked assets, and would never want to signal this fact.

As the protection signalled by digital emblems is codified in IHL, digital emblems must fit into the existing framework of IHL. Critically, IHL requires that digital emblems cannot have one or more designated "root authorities," which could attest to the authenticity of digital emblems. Put as a requirement, the mechanism providing digital emblems with authenticity must be decentralized. We thus specify an accountability mechanism to provide digital emblems with authentication, both inspired by and using existing transparency systems. In Chapter 5, we present ADEM, and we formally express both

ADEM's authentication and accountability guarantees and formally prove that ADEM provides them.

4

SOAP: A SOCIAL AUTHENTICATION PROTOCOL

4.1 INTRODUCTION

Most messengers allow their users to communicate without authenticating their chat partners. When users opt to do that, they rely on the application's key server correctly distributing public keys and on the authentication performed by the application provider during registration, which typically only requires to complete an SMS OTP challenge. Thus, using SMS-based attacks like SIM swaps [84], an adversary can impersonate users, and by compromising a key server an adversary can eavesdrop on users as a Meddler-in-the-Middle (MITM). Users must anyway trust their messaging provider to correctly implement the promised end-to-end encryption, but the default security policies force them to also trust that their providers' key servers are never compromised by insiders, attackers, or force by government authorities.

In this chapter, we present *SOAP, a SOcial Authentication Protocol*, that allows users to seamlessly and socially authenticate each other, and propose its application to secure messenger. We make the following contributions:

- We precisely define the security objectives of social authentication and argue that it should provide a novel security property that we call *sender correspondence*. This is a strong security property in that messaging sessions can only be compromised if all digital identities and the application's key servers are compromised. This raises the bar for the adversary and distributes trust among many providers. In contrast to most messengers' default security, neither the cellular provider nor the key servers can individually intercept messaging sessions.

  Sender correspondence fills the gap left by previous works [28, 86, 159] on social authentication, which never formalized social authentication as a security property, let alone studied whether a proposed design actually provides it.

- We formally relate sender correspondence to existing notions of authentication, and show how sender correspondence applies to systems beyond secure messaging.

- We present SOAP, a secure and practical protocol implementing social authentication. SOAP automates the authentication ceremony and provides a straightforward and immediate means for adoption. Figure 4.1 provides an overview of our design.

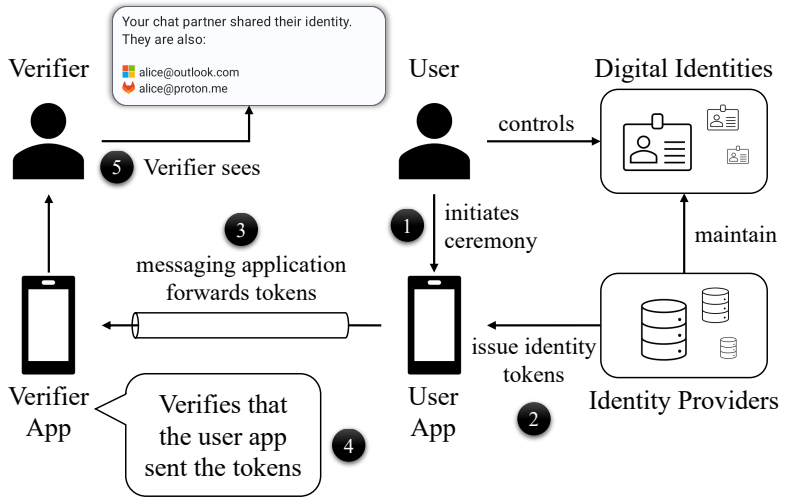


Figure 4.1: SOAP implements a social authentication ceremony. A user initiates the ceremony in their messaging application, which requests an identity token for each of the user's identities and forwards the tokens. The verifier's application verifies the token's sender. The verifier uses the identities to authenticate the user.

- Using Tamarin, we formally prove that SOAP satisfies our novel security property and that SOAP respects user privacy. Users can decide to whom they disclose which identities, and SOAP leaks no information to IdPs beyond that one is using the messaging app in question, e.g., Signal. By employing a salt-and-hash scheme, we avoid revealing key material to IdPs and, thus, leaking one's contacts to providers.
- We show that SOAP is straightforward to adopt by implementing it in two fully functional prototypes: a web-based application and an extension of the Signal Android application. The former requires some user interaction whereas the latter functions mostly automatically.

SOAP is the first formally verified authentication ceremony for messaging applications that works remotely. SOAP does not require users to work with cryptographic objects like keys or fingerprints, and subsequent work studying SOAP's usability found that this design can indeed enhance user security [76]. By leveraging an existing and widely used standard, SOAP is easy to implement and can be used immediately with any IdP that already supports OpenID Connect, which we demonstrate with two functional prototypes.

OUTLINE We proceed as follows. We formally define social authentication by instantiating a more general property which we call sender correspondence (Section 4.2). In that section, we also contextualize sender correspondence by showing how it applies beyond social authentication and how it relates to other notions of authentication. We

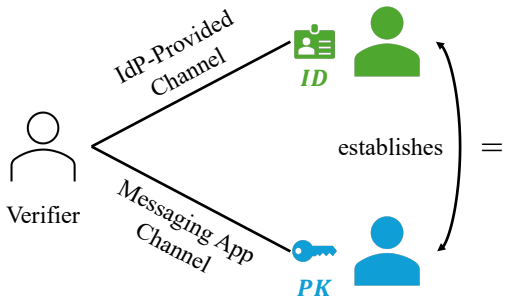


Figure 4.2: Social authentication establishes for a verifier that the digital identity ID and messaging application public key PK are controlled by the same person.

then present SOAP's design goals and design (Section 4.3). Afterwards, we formally prove SOAP's security (Section 4.4) and present prototypes (Section 4.5). We close by comparing SOAP to related work (Section 4.6) and drawing conclusions (Section 4.7).

4.2 SOCIAL AUTHENTICATION FORMALLY

In this section, we formalize the intuition of social authentication as presented in Section 3.2. We begin by defining the security property *sender correspondence* which can be instantiated as social authentication. Later in our related work section (Section 4.6), we show that sender correspondence finds application beyond secure messaging, and we relate it to existing, formal notions of authentication.

Security Property (Sender correspondence)**.** A protocol P guarantees a verifier *sender correspondence* between two pseudonyms A and B if, whenever P successfully terminates, then all messages that appear to have been sent by A and all messages that appear to have been sent by B were sent by the same user.

Social authentication is an instantiation of sender correspondence, where the pseudonym A is the messaging application public key PK and B is an IdP-controlled digital identity ID. We define sender correspondence in more general terms than social authentication because sender correspondence finds application in other protocols (see Section 4.6). We illustrate social authentication's security guarantees in Figure 4.2.

4.3 PROTOCOL DESIGN

In this section, we present SOAP's design in detail, which utilizes the OpenID Connect protocol [135, 136] to facilitate adoption. We start by stating its design goals, threat model, and design idea informally.

We then proceed to introduce the OpenID Connect protocol [135, 136], and finally present SOAP itself.

4.3.1 *Design Goals*

Naturally, SOAP was primarily designed to provide social authentication as defined in Section 4.2. Beyond that, SOAP was also designed to provide privacy. For example, IdPs cannot learn with whom their users communicate. We define SOAP's privacy property in terms of the allowed leakage to an IdP. In particular, IdPs neither learn who the prover authenticates to nor which other IdPs the prover uses. IdPs should only be able to learn that SOAP users (i) use SOAP, (ii) the messaging applications where they use it, and (iii) when they use it.

4.3.2 *Threat Model*

SOAP was designed to provide its security properties against two kinds of adversaries: We establish social authentication against an active network adversary and privacy against a malicious IdP. Whereas the social authentication adversary is an active network adversary in that it can read, intercept, reorder, and replay all messages, the malicious IdP can do the same but only with messages sent to it directly. For example, the malicious IdP cannot observe whether the prover forwards tokens to the verifier. We restrict our analysis of SOAP's privacy property to a malicious IdP as we wish to show that adding IdPs to the messaging application ecosystem does not threaten user privacy.

Our threat model permits the compromise of the messaging application's key server, the leaking of OpenID Connect requests to IdPs, and the compromise of some of the IdPs integrated into the messaging application. Notably, we make no assumptions on user-behavior other than that users do not leak their credentials. Users click any link sent to them by the adversary, whenever an IdP asks them for consent, they provide it, and whenever an IdP asks them to log in, they do so, even if the adversary triggered that query. We only limit our adversaries' capabilities in the following ways:

1. Adversaries are bound by the security properties of the cryptographic primitives used and the TLS and messaging application's end-to-end encryption protocols. For example, adversaries can neither invert cryptographic hash functions nor eavesdrop on a TLS session.
2. User credentials at IdPs are uncompromised.
3. Whenever a user authenticates via a given IdP, that IdP's signing keys and TLS certificates are uncompromised.

4. The messaging application and the user's web browser are uncompromised. In particular, the parameters of browser redirects to the messaging application remain confidential until they expire, and messaging application key material remains uncompromised.

The necessity for Assumptions 1-3 is self-evident. Regarding Assumption 4, it should be clear that we must require the messaging application and the user's browser to be uncompromised. We will discuss why we require the parameters of browser redirects to remain confidential later in Section 4.4.1 as this assumption requires a deeper understanding of SOAP's design. Finally, we require that messaging application key material remains uncompromised because an adversary could otherwise trivially impersonate the honest user controlling that key material. SOAP was not designed to defend against key compromise, but rather against impersonation by associating malicious keys with pseudonyms.

4.3.3 *Design Idea*

SOAP, our Social Authentication Protocol, works as follows. The prover requests an OpenID Connect *identity token* from an IdP and submits a hashed-and-salted conversation's safety number with that request. The identity token includes a signature by the IdP on the safety number and one of the prover's digital identities. At its core, this signature enables SOAP to provide social authentication and hashing-and-salting the safety number provides privacy. The prover forwards the token to the verifier, whose messaging application verifies it cryptographically and displays the prover's identity if all checks pass. In particular, this means that neither the verifier nor the prover must interact with cryptographic objects such as cryptographic keys or fingerprints thereof. In practice, users must run SOAP once per IdP to authenticate themselves to one contact, and only need to rerun SOAP should their long-term key material change.

We propose to run SOAP with multiple IdPs, which substantially improves user security compared to many messengers' default security. To the user, these multiple runs of SOAP (for multiple IdPs) will appear as one, which will become clear when we explain our prototypes in Section 4.5. Recall that, by default, a user's account can be attacked under the following condition: compromise the application's key server or compromise the cellular provider while the registration lock is not enabled (Chapter 3). Now suppose the prover runs SOAP with two IdPs. Both protocol runs are independent and, when completed successfully, the verifier's app will display the prover's identities with both IdPs.

It is now much harder to impersonate or MITM the prover: the adversary must compromise all IdPs used *and* either the prover's cellular

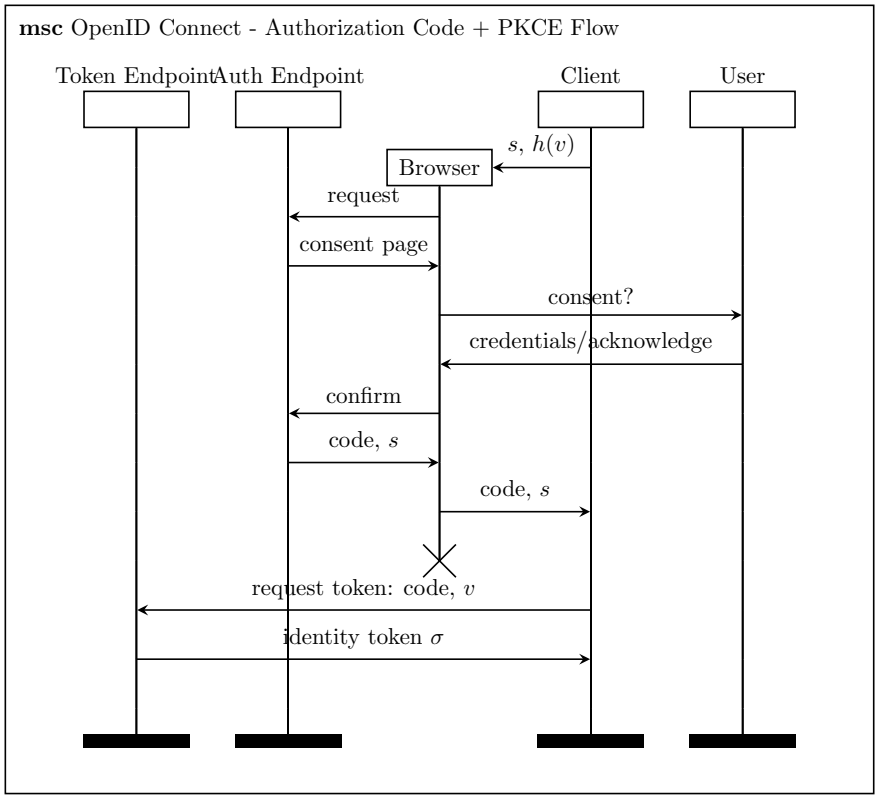


Figure 4.3: OpenID Connect authorization code flow with PKCE. $h(v)$ is the commitment to a random value v as specified by PKCE and s is the state parameter.

provider *or* the application's key server. Critically, the compromise of the application's key server no longer suffices, and in contrast to the registration lock, users can authenticate their chat partners and need not rely on their chat partners activating the registration lock.

4.3.4 *OpenID Connect*

OpenID Connect [135, 136] is an authentication protocol that itself builds on the OAuth 2.0 authorization protocol [75]. OpenID Connect is used to implement many of the well-known *"Login with Google/Microsoft/Apple/..."* buttons. OpenID Connect involves three parties: a *user*, an *IdP* managing the user's identity, and a *relying party* seeking to authenticate the user. The OpenID Connect protocol is executed by a *client* operated by the relying party. At the end of a successful protocol run, the client receives an identity token through a browser redirect from the IdP. The identity token is a cryptographically signed message, proving that the IdP authenticated the respective user, and which the client can use to identify the user. Prior to issuing requests, relying parties must register at the respective IdP. During registration, relying parties whitelist redirect URLs, and IdPs issue *client IDs*.

OpenID Connect supports multiple *flows*, which are protocol variants aiming at specific types of software clients. We use the authorization code flow extended with the Proof Key for Code Exchange (PKCE) standard [137], which is currently recommended as best practice for clients such as mobile applications [105]. The authorization code flow with PKCE implements a commitment reveal scheme, which we depict in Figure 4.3 and works as follows. The client first issues an authorization request by launching the device's browser at a specific URL called the *authorization endpoint*. The request encodes various parameters in the URL: a client ID, redirect URL, code challenge (the commitment, which is the hash of a random string), and optionally a state parameter and a nonce. The state parameter and nonce protect against replay and cross-site request forgery (CSRF) attacks.

After receiving an authorization request, the IdP verifies the redirect URL as whitelisted for the given client ID, authenticates the user, and asks the user for consent. Users usually authenticate by logging in, or through a session cookie already stored in their browser. Depending on the user's history with the IdP, the user may not need to grant consent.

Once a user consents to logging in, the IdP forwards the browser to the redirect URL given in the request. In the redirect URL, the IdP encodes an *authorization code* and the state parameter sent earlier. The client verifies that the state matches the state it previously issued, and exchanges the authorization code for an identity token. The client does this by sending a POST request to the IdP's *token endpoint*, including the authorization code and the *code verifier*. The code verifier opens the code challenge commitment sent earlier to the IdP. This allows the IdP to determine that the identity token is requested from the same client that issued the initial request.

The identity token is encoded as a JSON Web Signature (JWS), which is a signed object that maps keys to values. Among other values, the object includes the issuer, the audience (identifying the client), the subject (the user who was authenticated), the nonce, and a validity period. Usually, identity tokens are short-lived, with lifetimes typically ranging from two minutes to two hours. According to the OpenID Connect specification [135], identity tokens must only be accepted by the intended audience. This prevents a service to which a user logged in from using the identity token with another service.

4.3.5 *Protocol Description*

SOAP consists of three steps for the prover (request, validation, and forwarding) and one step for the verifier (validation), and we base SOAP on the OpenID Connect authorization code flow with PKCE as introduced in the previous section. Figure 4.4 sketches our protocol.

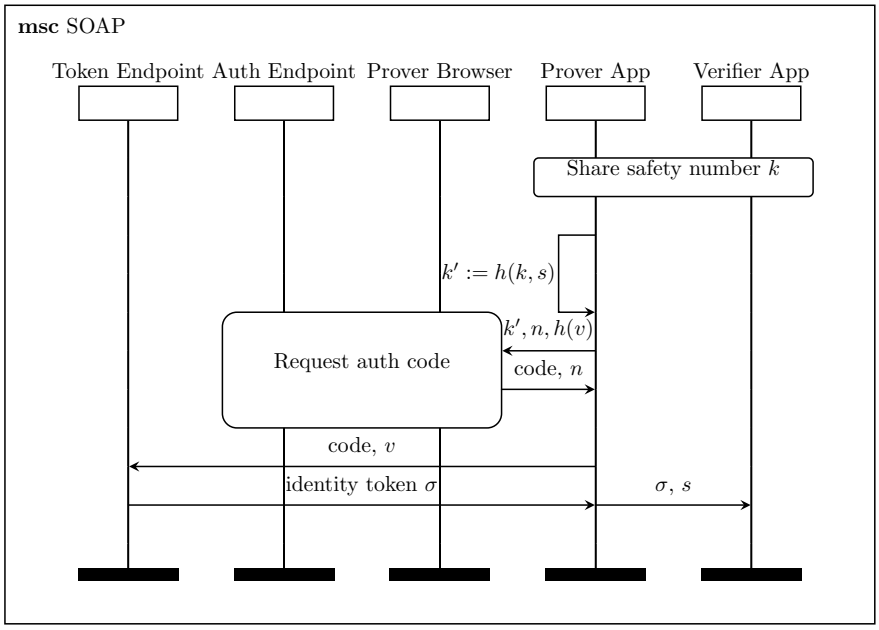


Figure 4.4: SOAP running between the prover, the IdP, and the verifier. Here, h_s is a password hashing algorithm using a salt s and h is SHA-256 as specified by PKCE. The application randomly samples a code verifier v, a salt s, and nonce n. Finally, σ is the OpenID Connect token, which is forwarded to the verifier and includes a signature on $h(k, s)$ and n.

To start a run of SOAP, the prover's messaging application prepares the request. It generates three random values: a code verifier cv, a salt s, and a nonce n. The application then uses a secure password-hashing algorithm to calculate a salted hash $h(k, s)$ of the safety number k. This hash serves to blind the safety number to the IdP. To defend against CSRF attacks, the application stores the salted hash $h(k, s)$, the salt s, the nonce n, the IdP's ID, and the code verifier cv as the most recently issued request. Then, the application launches the authorization code flow with the following arguments:

SCOPE: "openid email"; depending on the IdP, other scopes than "email" may be desirable.

RESPONSE_TYPE: "code"

NONCE: $n \mid\mid h(k, s)$; the application must ensure that it does not include the salt. $\mid\mid$ denotes concatenation. The application must ensure the parsing is unambiguous, e.g., by adding a delimiter character.

STATE: n

CODE_CHALLENGE: S256(cv); S256 marks the SHA-256 hashing algorithm.

CODE_CHALLENGE_METHOD: "S256"

Naturally, the application also includes its IdP-issued application ID, and an appropriate redirect URL. Redirect URLs must use the HTTPS scheme and must be distinct per IdP. Then, the application launches the system's browser with the request URL, which in turn takes the user to the consent and login page.

If the prover consents, the IdP redirects the browser back to the application. The redirect passes the application a state parameter and an authorization code, which can be exchanged for an identity token. Before the application uses the authorization code, it must verify that the state value it just received equals the nonce stored with the most recently issued request, and that the response originates from the expected IdP.

If both checks pass, the application uses the authorization code and stored code verifier to request the identity token from the IdP. It verifies the token's signature and that (i) the issuer matches the redirect URL stored, (ii) the token's audience matches the application ID, (iii) the token's nonce includes the hash stored, and (iv) the token has not expired. Finally, the application clears its storage for the most recently issued request, and stores the nonce in a replay cache. Recording nonces defends against reflection attacks; the description of our web-based prototype in Section 4.5.1 illustrates this threat.

The application forwards the identity token and the salt to the verifier and the verifier verifies the token's signature and that it has not expired. Additionally, the verifier verifies that the safety number encoded in the identity token encodes the prover's and verifier's keys, and that it did not request this token itself by looking up the stored nonces from runs where it was the prover. If these checks pass, the verifier can obtain the sender's identity from the token.

While Figure 4.4 may suggest that SOAP simply "calls OpenID Connect," previous work [65, 67, 105] highlighted many subtleties in implementing an OpenID Connect-based protocol securely. Hence, we next present our formal proof that SOAP indeed provides social authentication.

4.4 SECURITY ANALYSIS

SOAP is designed to implement social authentication and to protect user privacy, as introduced in the previous section and Section 4.2. In the following, we prove its security using Tamarin.

4.4.1 *Social Authentication*

We next provide high-level intuition on why SOAP provides social authentication and then describe our formal proof.

Informal Analysis

We previously defined social authentication as an implication: "If the verifier associates an account A with the public key PK and received a message from each of these pseudonyms, then these messages were sent by the same party."

There are two ways to violate social authentication. Either, there is no send event for one of the messages, or the send events have different senders. In our formal model, we modeled the message-exchange channels as authentic (i.e., given a receive event, there will be a send event) and thus focus on the latter case. Given SOAP's design, the attacker can achieve this by either making the prover send a malicious token through the messaging application, linking an attacker-chosen account to the prover's messaging channel (identity substitution), or making the prover authenticate in an attacker-controlled OpenID Connect flow, linking the prover's account to an attacker-chosen channel (impersonation).

Let us first focus on impersonation attacks, and presume Eve attempts to attack Alice. This means that Eve has a messaging session with Alice and wants to convince Alice that she, Eve, controls one or more of Bob's accounts at IdPs. To achieve this, Eve must send a token that includes a reference to Bob's account and the safety number of Eve's and Alice's public keys. As we assume that Bob's account is uncompromised (Assumption 2, Section 4.3.2), Eve must craft a malicious OpenID Connect request and forward it to Bob. This is straightforward as it requires nothing more than convincing Bob to click on a malicious link encoding such a request. However, Eve cannot obtain a token from this: Bob's application will discard the authorization response if it did not issue the request itself. Moreover, if it did issue the request itself, it would include one of Bob's safety number and not Alice and Eve's.

Similarly, Eve cannot launch an identity substitution attack. Eve would need to run SOAP herself, using a victim's safety number as parameter, log in with her own account, and then make the victim's application accept the resulting browser forward. However, the application would again discard that forward as it did not issue the corresponding request.

This argument spells out the main idea behind SOAP's security. In reality, the security of protocols based on OAuth 2.0 and OpenID Connect is much more subtle. Attackers can in general access authorization code responses through other means than capturing the redirects. For example, [65] first described the IdP-mixup attack, in which applications leak an authorization code by sending it to the wrong IdP. Therefore, we formally evaluate SOAP's security next.

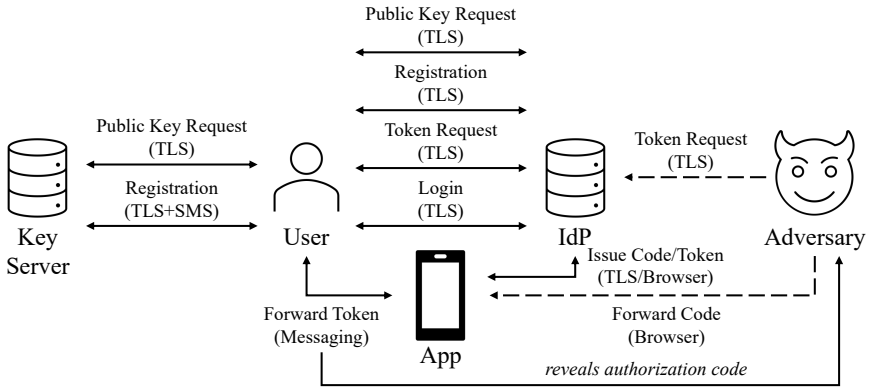


Figure 4.5: Sketch of our formal model of SOAP. Arrows indicate message exchanges, denoted with the respective channels used. Dashed arrows indicate that the adversary can initiate the respective request on the user's behalf. We omit the IdP-controlled bulletin board and the messaging channel.

```
All v sendKey rcvKey m1 idp acc m2 #t #r1 #r2.
    ( Correspond(v, sendKey, idp, acc) @ #t
    & ReceiveMessaging(sendKey, rcvKey, m1) @ #r1
    & ReceiveIdP(idp, acc, m2) @ #r2)
==> ( (Ex s #x1 #x2. SendMessaging(sendKey, rcvKey, m1) @ #x1
      & Sender(s) @ #x1 & #x1 < #r1
      & SendIdP(idp, acc, m2) @ #x2
      & Sender(s) @ #x2 & #x2 < #r2)
    | (Ex p #x. CompromisedAccount(p, idp, acc) @ #x)
    | (Ex #x. CompromisedIdP(idp) @ #x)
    | (Ex #x. CompromisedDomain(idp) @ #x)
    | (Ex app redirectURL #x #y #z.
          IsMessagingApp(app) @ #x
        & IsRedirectURL(idp, app, redirectURL) @ #y
        & CompromisedDomain(redirectURL) @ #z)
    | (Ex p #x. CompromisedMessaging(p, sendKey) @ #x))
```

Figure 4.6: Formalization of social authentication (Section 4.2), also encoding the threat model (Section 4.3.2). Note that the two `Sender` facts are bound to the respective `SendMessaging` and `SendIdP` events, as they occur at the same time points (`x1` and `x2`).

Formal Proofs

Our formal model comprehensively captures SOAP and its heterogeneous environment. Beyond the messaging application and TLS, we modeled all security-critical aspects and auxiliary protocols such as public key requests and distribution. Figure 4.5 depicts a sketch of what we modeled. Our model permits arbitrarily many participants to communicate with each other in arbitrarily many parallel protocol sessions. The adversary can corrupt any party in a fine-grained manner, e.g., a user's account could be compromised independently of their messaging long-term keys, only constrained by our security assumptions from Section 4.3.2. For example, IdP corruption is possible for every IdP except the one with which the prover intends to authenticate themselves.

More specifically, our model includes channels for SMS, TLS, browser redirects, and messaging applications with different security properties and distinct keys. We also modeled communication associated with IdP-controlled pseudonyms (usernames) as a bulletin board where users can post messages associated with their pseudonym publicly after the IdP authenticates them using a password. We modeled SMSes as insecure and the messaging application's encryption protocol as secure (confidential and authentic). We modeled TLS as a secure channel without client authentication, i.e., the adversary can always initiate new sessions with servers.

The adversary can compromise any TLS server, which allows it to read client queries and respond to them. TLS queries can have one of two methods, GET and POST, whereby GET requests can be initiated by the adversary on a user's behalf, modeling that the adversary can trick users into clicking any link. In practice, this allows the adversary to launch the OpenID Connect protocol at various points (e.g., initial request and code forwarding) for non-compromised clients. Browser redirects are modeled as GET requests using an existing session to connect to a new server, initiated by the previous server. This allows us to model, e.g., the redirect to a mobile application by modeling that application as a server.

We modeled the messaging provider, messaging application, end users, and IdPs as different parties. Our model includes SOAP itself, messaging application registration (including SMS OTP verification), IdP account registration, as well as messaging key server and IdP public key requests and responses. Moreover, we fully modeled OpenID Connect and we make no assumptions on this protocol's security.

Within this model, we prove that SOAP implements social authentication. Figure 4.6 shows our Tamarin specification formalizing social authentication as a trace property. It has three parts. Lines 2-8 formalize social authentication: If the verifier associates two pseudonyms with each other (`Correspond`), then all messages received from those two pseudonyms (`ReceiveMessaging` and `ReceiveIdP`) originate from

the same sender. We formalize the latter by showing that there exist two send events (`SendMessaging` and `SendIdP`) for which the sender (`Sender`) is the same party `s`.

Lines 9-16 formalize Assumptions 2-4 from our threat model (Section 4.3.2). Assumption 1 is covered implicitly as Tamarin operates in the symbolic model. There is just one subtle difference from social authentication as presented earlier. Namely, since the messaging channel is not just sender-authenticated, but also receiver-authenticated, we can include the recipient's pseudonym `rcvKey` in the receive and send event in lines 3 and 5. This means that our formalization of social authentication is stronger than sender correspondence.

The size and complexity of our model put its security properties out of reach for fully automated verification. For example, modeling that browser redirects remain confidential *until they expire* (Assumption 4) proved to be challenging. In part, we modeled this assumption by revealing authorization codes to the adversary after receiving the respective identity token. This led to infinite looping in Tamarin's proof construction, which we avoided by proving an inductive, auxiliary lemma showing that authorization codes can only be used once. In total, we verified nine auxiliary lemmas and programmed custom proof heuristics to aid Tamarin's proof construction.

Discussion

With our formal analysis of the authentication property completed, we briefly return to Assumption 4 of our threat model, where we require that the parameters of browser redirects to the messaging application remain confidential. Without this assumption, the adversary could easily obtain an identity token that binds an attacker-chosen safety number to a victim's account. To achieve this, they would only need to trick their victim into clicking a SOAP-request link that includes a malicious safety number as parameter. As soon as the victim logs in and consents (which they will do under our liberal threat model), the adversary could learn the authorization code from the redirect URL and request the identity token themselves.

Users can theoretically protect themselves from this attack by only granting consent to requests they initiated themselves. However, (i) we find it unrealistic to assume users are resistant to social engineering, and (ii) we experienced during our prototype development that some IdPs immediately acknowledge requests without user involvement whenever the user was already logged in and had previously granted consent. In this case, users could be attacked easily, e.g., with malicious URLs obfuscated with a URL shortener.

In practice, though, capturing redirects requires the adversary to have access to a user's browsing history while an attack is launched. This requires the compromise of the user's browser, or the installation of a malicious application handler on the user's device. In case of

a compromised browser, it is nigh impossible to protect the user's account credentials at the same time. To address malicious application handlers, we recommend that the application should use HTTPS redirect URLs as described in Section 4.3. With HTTPS URLs, application developers can utilize the security features provided by modern operating systems to ensure that authorization codes do not leak. For example, both Windows and Android support applications to handle HTTPS URLs, but only as long as these applications have been delegated to do so by the respective URL [61, 74]. This way, application developers can rely on the security features provided by the Web PKI to protect authorization responses. Without HTTPS URLs, an attacker would still need to install a malicious application handler and in turn a malicious application on their victim's device to capture redirects to custom schemes.

4.4.2 *Privacy*

SOAP protects users' privacy against the IdPs in that it only reveals that a prover is using SOAP, which messaging application is being used, and at what times SOAP is used. We formally proved SOAP's privacy as an observational equivalence property using Tamarin. Implementing our threat model (Section 4.3.2), we proved privacy in a simplified model (compared to the model presented in the previous section) that only includes communication between users and the IdP. We modeled the malicious IdP as the adversary and consequently replaced TLS with an insecure channel. Our observational equivalence property shows that IdPs cannot distinguish protocol runs where a user submits the correct salted-and-hashed prover/verifier safety number from runs where the user submits a different safety number. In the symbolic model, our privacy property is straightforward as we will lay out next.

SOAP only includes two requests to the IdP's servers, and these requests include the following parameters: the messaging application ID with the IdP, a redirect URL, a nonce, the code challenge (the hashed code verifier), the code verifier, the authorization code, and the salted-and-hashed safety number. The messaging application's ID and redirect URL reveal the messaging application and that the prover is using SOAP. The nonce, the code challenge, and the code verifier are randomly generated values that change with every request, and hence, leak nothing about the prover. The authorization code is issued by the IdP and therefore allows the IdP to connect the initial authorization request with the token request. Finally, the salted-and-hashed safety number leaks nothing about the prover under the assumption that the IdP cannot break cryptographic primitives such as salt-and-hashing, and because the salt is only shared with the verifier.

In theory, a verifier could share the salt with an IdP to reveal that the prover intended to communicate with the verifier. However, a verifier could leak this information even without SOAP. The IdP could also attempt to correlate public key requests with issued tokens. Fetching public keys, however, is not part of SOAP itself as applications will likely cache public keys. Additionally, the OpenID Connect specification [136] requires that IdPs distribute their public keys publicly and application-independently via HTTPS. For both of these reasons, we deem such a correlation to be practically infeasible. To summarize: SOAP prevents the surveillance by an IdP of its user's contact graphs in messaging applications.

4.5 PROTOTYPES

We implemented SOAP in two prototypes: as a stand-alone web application[1] and as a fork of the Signal open-source Android application. Both prototypes support GitLab and Microsoft as IdPs. The stand-alone version does not require messaging application adoption but requires more user interaction. In its current design, it can be used to associate arbitrary statements to a user's account.

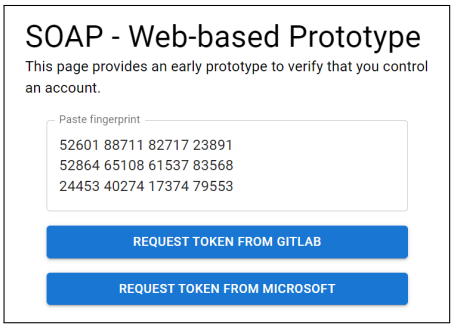
Our prototypes demonstrate that social authentication can be realized practically, with provable security guarantees, and without OpenID Connect-IdP adoption. Our web-based prototype shows that social authentication can be implemented even without messaging application adoption. Library support for OpenID Connect is abundant and, hence, adoption is straightforward. One of the authors could implement an initial prototype within a day. With just a few clicks and in a couple of seconds, users can verifiably share their identity.

4.5.1 *Web-based Prototype*

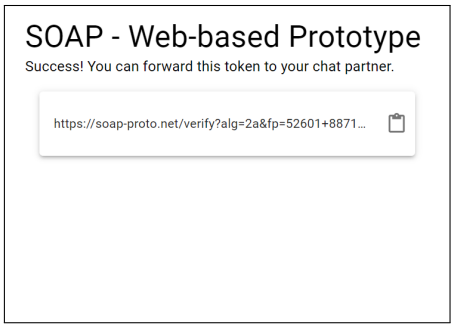
Figure 4.7 depicts the interaction with our web-based prototype. When started, the application renders a text input and a list of IdPs to select from (Figure 4.7a). After the prover selects an IdP, the application assumes that the input contains the safety number to authenticate, and initiates SOAP. After they complete the OpenID Connect flow (see Section 4.3.4), the prover is forwarded to our application, which requests the identity token (Figure 4.7b). The web application verifies the identity token, and, if all checks pass, displays a success message. The prover can then copy a link to send to their chat partners.

This link encodes the identity token and everything needed to verify it. When the verifier clicks the link, the application verifies the token, and if all checks pass, displays the prover's identity, the IdP, and the safety number. The verifier can copy this safety number to

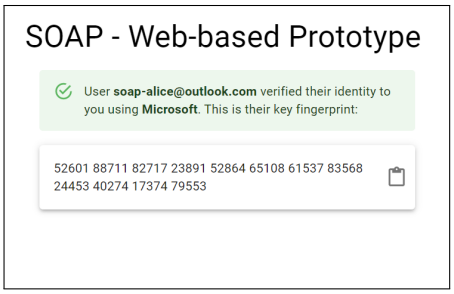
[1] The prototype is hosted at `https://soap-proto.net`.



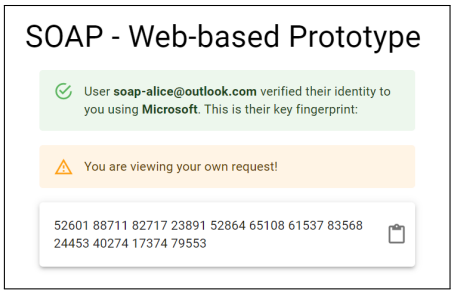
(a) Prover pastes safety number into the application and selects one of two IdPs.



(b) Prover receives identity token and is presented a URL to forward to their chat partner.



(c) Verifier sees the prover's identity and safety number when clicking the link.



(d) Prover sees a warning when clicking a link generated within the same browser.

Figure 4.7: Screenshots depicting the web application prototype. The prover must initiate this flow for each IdP.

their clipboard. The Signal Android application offers its users to compare the safety number with the clipboard, giving the verifier an easy way to check the safety number (see Section 3.1).

Note that the application will render a success message to anyone clicking a link containing an identity token. Only the user can determine whether the safety numbers match. This allows for social engineering attacks whenever users click links they earlier forwarded themselves. To mitigate this threat, the web application warns users that they issued this request themselves whenever they click a link that was issued within the same browser. Figure 4.7d shows the view after clicking on a URL that encodes an identity token.

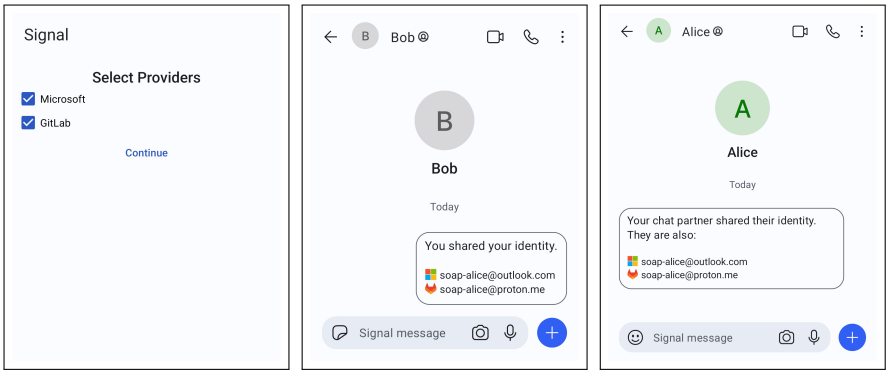
4.5.2 *Signal Prototype*

Our Signal prototype, depicted in Figure 4.8, provides a more streamlined experience compared to the web-based prototype.[2] In particular, the Signal prototype requires significantly less interaction from the prover and no interaction from the verifier. Users need not actively examine and insert safety numbers, and flows need not be initiated for each IdP.

When a user wishes to authenticate themselves to their chat partner (becoming a prover), they must first select the new "Authenticate" option within the attachment menu. Next, the prover selects all the IdPs they wish to authenticate themselves with (Figure 4.8a). When they press continue, they will run SOAP for each of the IdPs. The application verifies all tokens and displays a distinctly styled message to both the prover and verifier that shows the shared identities (Figure 4.8b, 4.8c).

Our Signal prototype demonstrates that SOAP is a practical design that requires little user interaction. While it is adequate for a proof-of-concept, we suggest further enhancements before it is deployed in production. First, previously performed social authentications should be recorded as such in the application. The application could display a contact's identities within the chat header, rather than mark them as "Verified." This would additionally highlight that our proposal is intended to augment in-person safety number comparison, not supersede it. Second, the messaging application could compare the identities provided through SOAP with identities linked to the contact's address book entry on the smartphone to automatically combat impersonation attacks. Messaging applications are usually granted access to address books anyways, providing a straightforward means to automate social authentication further.

[2] A video demo of the Signal prototype can be viewed at https://youtu.be/Ip_RAF8PRrM.



(a) Prover selects the IdPs to authenticate with.

(b) Prover sees that they shared their identities.

(c) Verifier sees the prover's identities.

Figure 4.8: Screenshots depicting the Signal prototype. The interaction with our prototype is depicted from left to right.

4.5.3 *Development*

The web-based prototype was written as a single-page JavaScript application in React [125], i.e., all code runs in the user's browser. The prototype consists of roughly 500 lines of code and the first version was developed within a day by one person. In contrast to the web-based prototype, the Signal prototype's development was more involved and required around three person weeks. Understanding the poorly documented Signal Android codebase demanded most of the time. We changed 21 files and added around 1000 lines of Java and Kotlin code for the application logic, and also changed 17 files with around 200 additional lines of code for configuration changes, like layout and localization.

During prototype development, we noticed that not all IdPs support OpenID Connect ideally for our use case. Microsoft, for example, does not support HTTPS redirect URLs for Android applications. We tried working around this restriction by registering our Android application as a Single-Page Application, which permitted us to configure an HTTPS redirect URL. However, requesting the identity token failed as cross-origin request headers were missing.

GitLab supports HTTPS redirects and does not distinguish the types of applications upon registration. However, GitLab does not ask for consent again after the user consented once to log in. This causes HTTPS redirects to Android applications to fail. A Chrome policy requires user interaction in order to redirect users to Android applications through HTTPS URLs [9]. This left us with using custom schemes, e.g., `auth://`, in redirect URLs to support GitLab as an IdP.

Finally, Google neither allows one to configure a redirect URL nor lists a redirect URL when registering Android applications as an OpenID Connect client. In this way, Google hides their OpenID Connect API. We suspect this practice is intended to force developers

to implement "Sign-In with Google" using Google's SDK [150]. This SDK need not be configured with a redirect URL to request identity tokens and does not allow one to specify a nonce.

These findings suggest that while SOAP does not require adapting the OpenID Connect specification, explicating our use case in the OpenID Connect specification could still benefit users and developers. Currently, users will only consent to log in, and developers have to use the nonce field outside its specified intent. If OpenID Connect were to recognize user-submitted claims as a request parameter, users could grant consent to show that they control the given account and developers could use APIs supporting user-submitted claims.

4.5.4 *Performance*

The performance of both prototypes is mainly constrained by the page load times of OpenID Connect authorization endpoints, consent screens, and HTTP redirects. We timed the prototypes with a user that was logged in and had authorized our application already. In the web-based prototype, running SOAP for a single IdP was nearly instantaneous (<1 second), and in the Signal prototype, running SOAP for two IdPs consistently took less than 10 seconds.

As we described in Section 4.3, SOAP requires local, persistent storage to protect against replay and CSRF attacks. SOAP stores the latest issued request, which only requires a small, constant amount of space, and the nonces generated by the application. The number of nonces stored is limited by the number of SOAP sessions initiated by the user, and the nonces can be discarded after a token expires, which in our experience happens within two hours.

4.6 RELATED WORK

4.6.1 *Long-Term Key Authentication*

Already without social authentication, messaging users can defend themselves against impersonation or MITM attacks. As we explained in Section 3.1, all modern messengers support authentication ceremonies in which users compare safety numbers with their chat partners. Although these authentication ceremonies provide strong security guarantees, their actual benefit is questionable.

Various studies showed that users are unlikely to perform the authentication ceremonies in the first place, and even if they were to do so, they are unlikely to perform them correctly. When receiving instructions, these numbers rise to 75%-80%. Remarkably, around 50% of [77]'s participants indicated that they would not perform the authentication ceremony again in the future, even after its importance was explained to them. Another study investigated the usability of

SMS-based authentication ceremonies supported by Signal's share button in the safety screen's top right corner [142]. The authors found that in 40%-60% of the cases, a difference in safety numbers went unnoticed. Notably, the study considered a best-case scenario by recruiting educated, technology-savvy users who were explicitly instructed to compare safety numbers.

But even if users were to authenticate one another, it is unlikely that they would perform the authentication ceremony again when under attack. When an adversary impersonates a user, messaging applications will typically notify the impersonated user's contacts that their conversations' safety number changed, encouraging to proceed only after authenticating the session again. However, to successfully thwart an impersonation attack, users must (i) notice and understand this warning, and (ii) compare safety numbers in person, which requires physical proximity, or (iii) compare safety numbers using an ad-hoc out of band channel, which must be first agreed upon – while possibly talking to an attacker.

As we discussed in Section 3.3, key transparency systems were proposed to address the shortcomings of safety number comparisons, and both iMessage [1] and WhatsApp [101] provide key transparency services. In contrast to social authentication, transparency systems have the upside that they require no user-interaction as long as the providers act honestly, but this comes at the cost of a significant engineering effort to implement authenticated data structures correctly and can require recruiting external auditors. Moreover, transparency systems cannot prevent compromise but only make it detectable, and they provide no notion of authentication, which is desirable in its own right.

Keybase [86] were the first to propose a design promising to provide social authentication, but Keybase requires manually and publicly posting key material, which is error-prone due to its reliance on manual posting and discloses account associations to everyone. In contrast to Keybase's approach, SOAP provides more automation and stronger privacy guarantees. After Zoom acquired Keybase, Zoom published an end-to-end encryption whitepaper [28], which continued this line of work. The whitepaper proposes "Identity Provider Attestations," which authenticate Zoom encryption keys using OpenID Connect and DNS. Organizations can delegate an IdP via DNS as eligible to authenticate that organization's users. The Zoom client can then (i) upload a commitment to a user's public key at the IdP on the user's behalf using OAuth and a custom API, and (ii) request an identity token that includes that commitment. Both these steps require adoption by the IdP, and Zoom's design considers the case in which an account delegates authentication to a single, trusted IdP. SOAP can be seen as an extension of Zoom's design that works without IdP adoption and for multiple IdPs, which makes it strictly more difficult to compromise a messaging account.

4.6.2 *Formal Analyses of OAuth Protocols*

Our formal analysis of SOAP (Section 4.4.1) was influenced by the recommendations of *OAuth 2.0 Security Best Current Practice* standard [105] and the formal analyses of the OAuth 2.0 and OpenID Connect protocols conducted in [65, 67]. The latter works, [65, 67], conducted pen-and-paper proofs in the Web Infrastructure Model [66], which captures more details of the browser environment than our model, e.g., HTTP status codes and their semantics. In contrast, we utilize Tamarin, generating machine-checked proofs and consider a strictly stronger adversary than both [65, 67] and the original specifications [75, 106, 135]. Neither of these considered the leakage of authorization requests.

Only [64], the formal analysis of the OpenID Financial-grade API, considers a stronger attacker model not requiring Assumption 4 (browser redirect parameters remain confidential). However, [64] analyzes a different profile of OAuth 2.0 that does not match our setting as it assumes that applications can protect secrets, which enables the IdP to authenticate clients. Dropping Assumption 4 for SOAP would allow the adversary to capture redirects and thus effectively allow them to run the protocol themselves altogether (see Section 4.4.1).

[73] proposed the Privacy-Preserving OpenID Connect (POIDC) protocol, which enhances OpenID Connect's privacy guarantees, and also analyzed the security of their proposal in Tamarin. While [73] models the process of users granting consent more explicitly (logging in and providing consent are two steps, which we model as one), they make stronger assumptions on user behavior. Namely, they require that users only log in and consent to OpenID Connect flows when they themselves launched the protocol. We do not make this assumption and it is unrealistically strong. Some IdPs neither require a login (given an existing session) nor require consent (given that consent has been granted in the past; see Section 4.3.4). Moreover, [73] does not model the authorization code flow with PKCE, which our design relies upon. Nevertheless, our design's privacy guarantees could be enhanced if designs such as POIDC were adopted.

4.6.3 *Sender Correspondence*

4.6.4 *Sender Correspondence in the Wild*

Sender correspondence finds application beyond social authentication, which we illustrate with the Automatic Certificate Management Environment (ACME) protocol [3, 12], powering the free CA *Let's Encrypt*. ACME automates the certificate request and issuance procedure. Using ACME, CAs verify certificate requests using a challenge-response mechanism in three steps. First, a CA receives a certificate request for a given domain name, signed by a private key. Second, the CA sends

a challenge to the respective public keyholder and asks them to return it using DNS or HTTP. Third, the CA verifies that they receive the signed challenge through the DNS or HTTP channel, at which point it issues a certificate for the respective public key.

The ACME protocol can be seen as establishing sender correspondence. Namely, the requesting public key is the pseudonym A and the domain name is the pseudonym B. This has been overlooked by related work so far. Related literature [20, 80] verifying the ACME protocol only considered key establishment properties. Namely, they require that for any attack on ACME, the adversary must know the certificate's corresponding secret key. However, they do not consider identity misbinding attacks, where the adversary could provide the domain name that gets associated with an honest key. [12] analyzed ACME's domain validation algorithm and considered an authentication-style property. But as the authors only analyzed this one part of ACME, they did not consider ACME's overarching security goals.

In general, sender correspondence can be applied to any two channels that allow for information exchange associated to pseudonyms. An IdP-managed online version-control system like GitLab is a channel where users exchange information (commits, comments, etc.) associated with a pseudonym (usernames). Similarly, DNS is a channel where information (encoded in DNS records) is associated to pseudonyms (domain names).

4.6.5 *Relationship to Other Authentication Properties*

We can formally relate sender correspondence to two other formal notions of authentication: non-injective agreement as introduced in Section 3.1, and sender invariance. In this section, we show how sender correspondence connects to both.

Non-Injective Agreement

To relate sender correspondence to non-injective agreement, we first formalize both properties. Consider the following, standard formalization of non-injective agreement in Tamarin:

```
All R S m #tr. Receive(R, S, m) @ #tr
  ==> (Ex #ts. Send(S, m) @ #ts & #ts < #tr)
```

Now, compare this formalization to one of sender correspondence:

```
All V R1 R2 PX PY mx my #ta #trx #try.
      ( Correspond(V, PX, PY) @ #ta
      & ReceiveChX(R1, PX, mx) @ #trx
      & ReceiveChY(R2, PY, my) @ #try)
  ==> (Ex S #tsx #tsy.
          SendChX(PX, mx) @ #tsx
        & Sender(S) @ #tsx
```

```
8         & #tsx < #trx
9         & SendChY(PY, my) @ #tsy
10        & Sender(S) @ #tsy
11        & #tsy < #try)
```

`ReceiveChX(R, S, m)` and `ReceiveChY(R, S, m)` model that `R` received message `m` from `S` on channel `X` or channel `Y` respectively. `SendChX(S, m)` and `SendChY(S, m)` model that `S` sent message `m` on channel `X` or `Y` respectively. Note that `S` could be a pseudonym. `Correspond(V, PX, PY)` formalizes that the verifier `V` identifies pseudonyms `PX` and `PY` with the same party, i.e., that a verifier `V` terminated a protocol proving social authentication. `Sender(S) @ #t` formalizes that the message sent at time point `t` was sent by agent `S`.

Intuitively, sender correspondence relates to non-injective agreement for two reasons: (i) Sender correspondence only works when the two associated pseudonyms can be used for authentic communication. Otherwise, it would make little sense to "tie" them together. (ii) The formalizations of both properties are very similar: lines 3, 6, 8 and lines 4, 9, 11 exactly match our formalization of non-injective agreement.

We can express both points formally. Namely, sender correspondence establishes non-injective agreement on channel `X` and `Y` for the pseudonyms `PX` and `PY`. Specifically, we show that whenever there is a successful run of a protocol providing sender correspondence between `PX` and `PY`, that trace also satisfies non-injective agreement for both of these pseudonyms (formalized using the respective `ChX` and `ChY` events). If such a trace were a counterexample to non-injective agreement for channel `X`, there must be an event `ReceiveChX(R, PX, m)` for which there is no corresponding `SendChX` event. In that case, since `R1` and `mx` in the formalization of sender correspondence are universally quantified, that trace would be a counterexample to sender correspondence as well. This contradicts our assumption that the protocol provides sender correspondence. The case for channel `Y` follows symmetrically.

Note that this does not mean that sender correspondence implies non-injective agreement. This is because sender correspondence requires that a `Correspond` event has occurred on the trace. It could be that a protocol P providing sender correspondence is designed in such a way that it prevents violations of non-injective agreement when it was run, but that either channel `X` or `Y` does not provide non-injective agreement on their own. Only if we assume that running protocol P is in some notion "independent" from communicating on channels `X` and `Y` can we say that proving sender correspondence also proves the non-injectivitiy of channels `X` and `Y`.

The relationship between sender correspondence and non-injective agreement highlights that sender correspondence is a desirable authentication property. When successfully running a protocol providing sender correspondence, we know that both pseudonyms can be used

for authentic communication *and* that they are controlled by the same sender.

Sender Invariance

In [60], the authors distinguish two notions of authentication that we conflate: *(non-injective) agreement* and *sender invariance*. Whereas non-injective agreement (in [60]) is defined for all kinds of agents, sender invariance is defined only for pseudonyms, capturing the security guarantees behind authenticated channels that use unauthenticated public keys. One does not know who one is connected with, but it must always be the same agent, provided their corresponding private key does not leak. The authors show that non-injective agreement implies sender invariance. Thus, in the formalism of [60], sender correspondence establishes both non-injective agreement and sender invariance.

4.7 CONCLUSION

Social authentication is an exciting authentication paradigm promising usable [159], remote, and automated authentication in messaging applications. In this chapter, we precisely and formally defined social authentication, we presented SOAP, a secure and practical protocol implementing social authentication, we formally proved that SOAP implements social authentication even in the presence of a strong adversary, and we demonstrated SOAP's practicality in two prototypes.

Beyond secure messaging, SOAP can be applied to any application to authenticate key material and, more generally, applied to any kind of pseudonyms. SOAP is automated to a large degree and can immediately be adopted by IdPs (indeed, it may not require adoption at all) because it relies on the well-established OpenID Connect protocol. It implements a secure and complete in-application ceremony that requires nothing more of users than their consent. Widespread adoption in messaging applications would be a cost-effective measure to increase their robustness against impersonation attacks and eavesdropping.

FUTURE WORK Our results suggest several next steps. First, we argue that social authentication should be supported by modern messaging applications. Improving our open-source Signal prototype such that it could be deployed in production is a promising first step in that direction. Second, social authentication should be applicable far beyond secure messaging. For example, it could be used to secure other communication such as e-mail, or video conferencing, or it could be used as a second factor. Finally, we suggest amending the OpenID Connect specification to support user-submitted claims such that users can consent unambiguously and developers can use streamlined APIs.

5

ADEM: AN AUTHENTIC DIGITAL EMBLEM

5.1 INTRODUCTION

In this chapter, we tackle a novel security problem that arises in times of armed conflict, and we address this problem using an accountability mechanism. IHL mandates that military units must not target medical facilities, such as hospitals. The emblems of the Red Cross, Red Crescent, and Red Crystal are used to mark *physical* infrastructure (e.g., by a Red Cross painted on a hospital's rooftop), thereby enabling military units to identify those assets as protected under IHL. In 2022, the International Committee of the Red Cross (ICRC) [131] posed the question: How can one extend such markings to *digital* infrastructure such as servers and networks?

Extending protection to digital infrastructure comes with unique security requirements. They are unique both in that existing authentication systems fail to provide them, and that they do not apply to the physical emblems of the Red Cross, Red Crystal, and Red Crescent.

(i) Digital emblems require authentication. Assets must not be able to fake protection, for example, by transferring markings from medical to military infrastructure.

(ii) Those wishing to authenticate assets must be able to verify protective markings in a way that does not call attention to the fact that they are screening potential targets. We call this property *covert inspection*.

IHL foresees that prior to displaying protective emblems one must seek permission from competent authorities. A natural approach to solve the problem of authentication would be to extend this process. Authorities could sign certificates for what we call *emblem issuers (EIs)*, and EIs could use these endorsed keys to sign digital emblems. But how would one identify competent authorities? Due to the digital emblem's roots in IHL, a digital emblem system cannot designate a fixed set of authorities or a "root authority" that attest which authorities are legitimate. Under IHL, countries must be able to fully autonomously decide which parties can display emblems under their jurisdiction. Thus, the protective markings must work in a decentralized way.

We address the problem of authentication by specifying an accountability mechanism, and thus avoid the problem of authenticating competent authorities. Competent authorities can self-declare as such, and we require both authorities and emblem issuers to commit to

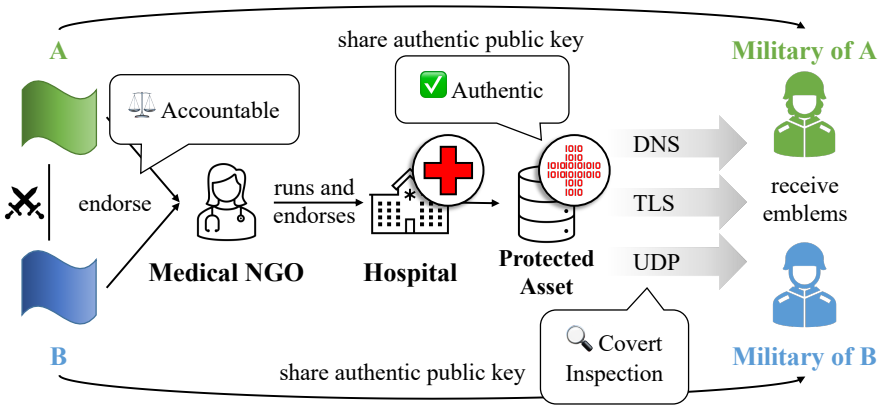


Figure 5.1: Overview of ADEM. The countries A and B endorse an emblem issuer as legitimate, which in turn endorses a hospital as belonging to them. The hospital's protected asset distributes digital emblems using UDP, TLS, and DNS. The countries' militaries can independently verify these emblems using their own countries' public keys.

sufficient evidence that they can be held accountable should they ever mark unprotected infrastructure. This design matches the practice around physical emblems. Everyone can paint a Red Cross on their roof, but they must fear prosecution when doing so unlawfully.

Existing systems solve some but not all of the above challenges. They typically require interaction between two participants and any interaction between a military unit and a potential target would reveal the unit's intention to attack that target, should it be unprotected. An unprotected target could use this knowledge to defend itself against an imminent attack, which in turn would deter military units from verifying whether their targets are protected in the first place. Moreover, typical designs providing authentication and accountability are usually centralized, e.g., in authenticated data structures such as Merkle hash trees (MHTs) [114]. Adding consensus protocols to maintain such data structures in a decentralized way does not help as they aim to establish *consensus*, which cannot be assumed in an international context.

To fill this gap, we present *ADEM, an Authentic Digital EMblem*, a design that solves all these challenges. Figure 5.1 provides an overview of ADEM. Digital emblems in ADEM are cryptographically signed messages and authenticate an asset as protected under IHL. Emblems are distributed actively by these assets using UDP, TLS, or DNS in a way that provides covert inspection. Emblems are accompanied by endorsements, certificate-like objects signed by independent authorities such as countries. ADEM provides accountability by using the CT log infrastructure, where parties commit to their public keys, and ADEM does not require any updates to the Internet's infrastructure.

In total, we present three contributions.

- The ICRC recently published a report that presents the idea of a digital emblem and proposes initial requirements [131].

These requirements, however, are mostly high-level and not actionable. For example, the ICRC requires that a digital emblem must be "obvious and easily visible." We take the report as a basis to precisely define the requirements for a digital emblem, emphasizing its security requirements.

- We present an ADEM, a design that achieves all these requirements and implements a digital emblem analogous to the physical emblems of the Red Cross, Crescent, and Crystal.
- We provide a comprehensive threat model and security analysis showing that ADEM achieves strong security guarantees against an active network adversary. This includes both the formal definition and proof of authentication and accountability using Tamarin and a security rationale for covert inspection. In doing so, we highlight that accountability is a subtle property, hard to root in intuition, and (perhaps not coincidentally) rarely considered in the literature.

ADEM has gained significant traction since its original proposal. Prior to its initial publication, ADEM was evaluated in a series of meetings organized by the ICRC. Domain experts with various backgrounds, such as health care, the military, and cybersecurity were invited to provide feedback on the idea of a digital emblem in general and design proposals, such as ADEM, in particular. From these meetings, the ICRC concluded in their report to "continue research and consultation on a possible 'digital emblem', [which] will require further work on the technical development, validation and verification of possible solutions" [131]. Since then, a resolution encouraging the ICRC to continue the development of a digital emblem was adopted at the 34th International Conference of the Red Cross and Red Crescent [130]. This resolution was voted upon by all states and national Red Cross and Red Crescent movements of the world. Now, the formation of a working group for digital emblems at the Internet Engineering Task Force (IETF), an international standards organization, is in its final stage, and we plan to begin the technical standardization of ADEM in collaboration with the ICRC within the next months [57].

OUTLINE We proceed as follows. We start by defining the requirements of a digital emblem (Section 5.2). Afterwards, we present ADEM (Section 5.3), present formal notions of accountability (Section 5.4), and analyze ADEM's security (Section 5.5). Finally, we present related work (Section 5.6) and draw conclusions (Section 5.7).

5.2 THE PROBLEM

We begin with the requirements for a digital emblem. For this, we introduce relevant legal and historical background, describe the prob-

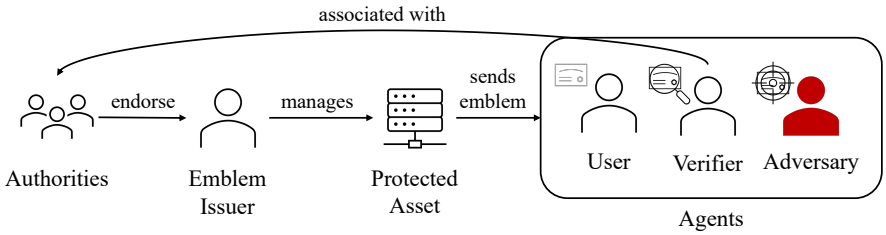


Figure 5.2: Parties involved in deployment and use of digital emblems.

lem domain, and finally provide a digital emblem's requirements, which follow from the ICRC's report on digital emblems [131]. However, whereas the report introduces requirements on a much higher, abstract level and without a detailed consideration of security, we present a comprehensive list of actionable, technical requirements and put security in focus.

5.2.1 *Legal and Historical Background*

The Geneva Conventions [128] and their Additional Protocols (APs) constitute the core of IHL and establish legal rules on the conduct of armed conflict. These rules codify the meaning and usage of protective emblems, namely, the Red Cross, Crescent, and Crystal, permitting EIs to mark their assets, such as vehicles, personnel, or buildings with these signs during armed conflicts. These signs inform other parties about an asset's affiliation with the International Red Cross and Red Crescent Movement (*indicative* use), or about an asset's protected status under IHL (*protective* use). Actors bound by IHL must respect an asset's protected status and not attack it, except in very limited circumstances.

Since 1949, the Geneva Conventions have been amended and extended. AP I [126, 127], for example, contains additional regulations on how EIs could communicate their status using technical means, like radar transponders and radio signals. Recognizing that technology may progress rapidly, AP I allows for the ICRC to convene state experts to review and suggest updates to the technical means by which the EIs may be identified. In order to initiate discussions among states, the ICRC proposed the idea of a digital emblem on an international stage [131] in collaboration with external experts.

5.2.2 *Problem Domain*

We next introduce the different stakeholders and parties of a digital emblem. Figure 5.2 shows the five relevant kinds of actors. First and foremost there are *emblem issuers (EIs)* who operate in areas of conflict and conduct operations that enjoy protection under IHL utilizing

protected digital assets, such as mobile devices (tablets, smartphones), servers (both virtual and dedicated), processes such as web servers, or an EI's intranet. Prior to displaying protective emblems, EIs must seek permission from competent *authorities*. When they are given permission, EIs can apply digital emblems to their protected assets, which *present* them to three types of *agents*.

Regular users of an asset do not pay attention to the emblem, for example, when they visit a website. *Verifiers* respect IHL and thus pay attention to the emblem as they only wish to attack lawful (under IHL) targets. They are typically part of an armed force engaged in a conflict. We can assume that most verifiers (typically military units) will be associated with an authority (typically their country or an ally), and, hence, that such verifiers can obtain the authentic public keys of their affiliated authorities through secure, out-of-band channels. *Adversaries* are willing to violate IHL and seek to abuse digital emblems. For example, they may seek to issue emblems to unprotected assets.

We highlight that we must assume verifiers to respect IHL. An emblem works like a sign and will thus only work for those that respect it. In our case, we define verifiers to be those parties that respect the sign and thus respect IHL, for which there can be many reasons. Countries or independent conflict parties may respect IHL because they are bound by it and, consequently, attacking protected assets can be a war crime. Verifiers not bound by IHL, e.g., cybercriminals, may also want to not target protected assets to avoid unwanted attention or because they respect the humanitarian cause.

5.2.3 *Requirements*

We derive the technical requirements of digital emblems from two, more abstract requirements. First, the ICRC states that a digital emblem should fit into the existing framework of IHL [131]. Compliance with IHL facilitates the diplomatic process of adopting a digital emblem, possibly integrating it into AP I. In particular, this means that a digital emblem should work similarly to the existing physical emblems. Second, a digital emblem must be able to prevent attacks on protected assets in practice. The physical emblems achieve this by informing verifiers about an asset's protected status under IHL.[1] As the emblem is a sign, verifiers must be willing to pay attention to digital emblems for them to have an effect, which critically informs our problem statement.

We start with the digital emblem's *functional requirements*, which we summarize below. We derive these requirements from the abstract requirement that digital emblems must work like the existing physical emblems.

[1] Although IHL also permits the indicative use of an emblem, we subsume this case under the protective use of an emblem in the remainder of this thesis.

FR1 A digital emblem must be *decentralized*. There can neither be a fixed set of parties in charge of distributing emblems nor a fixed set of authorities deciding who is eligible to issue emblems.

FR2 Just as a flag with a Red Cross can be affixed to supplies, personnel, buildings, or vehicles, a digital emblem must be *widely applicable* to a broad range of devices, infrastructures, and organizations.

FR3 Again, just as a flag with a Red Cross can be removed at any time, a digital emblem must be similarly *easily removable*. In some scenarios, digital emblems may become rather a target sign than a protective sign. With a removable emblem, EIs can assess their individual risk and remove digital emblems freely whenever they find it necessary.

FR4 Agents must be able to *verify the presence of digital emblems*. With physical emblems, unprotected assets will simply not show the flag of the Red Cross. Likewise, in the digital domain, these assets will not present a sign stating that the asset has no emblem, but rather no emblem. Verifiers must always be able to decide with reasonable confidence whether an asset is marked with a digital emblem.

We next turn to the digital emblem's *security requirements*. We derive these from the abstract requirement that digital emblems must prevent attacks in practice. This can only be achieved when verifiers are willing to use the digital emblem system. We summarize the digital emblem's security requirements (SR) below.

SR1 Verifiers never want to reveal themselves as inspecting emblems. If verifiers were to reveal this, their targets could use that knowledge to protect themselves against an imminent attack. Verifiers intend to attack lawful (under IHL) targets and would never want to warn their targets before an attack. Thus, a digital emblem must provide what we call *covert inspection*: Agents who wish to verify whether an asset is protected under IHL must be indistinguishable from agents who interact with that asset for other legitimate purposes. In particular, covert inspection implies that emblem presentation must be *active*, i.e., verifiers will be sent emblems and need not query them explicitly.

SR2 Emblems must be *verifiably authentic*. Agents must be able to correctly associate emblems to the respectively marked assets and must be able to have reasonable confidence in the emblem marking actually protected assets.

SR3 Since a digital emblem system must be decentralized (see **FR1**), we argue that a digital emblem can best provide authenticity

when it provides *accountability*.[2] That is, one must be able to identify parties that issued fraudulent digital emblems to assets not protected under IHL. An accountability mechanism additionally matches a digital emblem's legal nature. There cannot be a mechanism that automatically decides whether an asset truly enjoys protection under IHL, and authorities and EIs might err or act maliciously when deciding who to endorse or what to mark with a digital emblem. Should a digital emblem be codified in law, it is crucial that any such misbehavior can be prosecuted.

Interestingly, the requirement for authentication (**SR2**) was not considered when designing the original, physical emblems. Evidently, the physical emblems provide no means to authenticate whether they are legitimate. We argue that this is the case because the physical world provides a means of *immediate accountability*. Whenever someone displays a flag of the Red Cross, they put themselves on the line. If you were to hang a flag of the Red Cross on your balcony, you could expect a visit from the police. If you were to drive around in car with a Red Cross on it, you could be stopped and questioned too. This is not the case in the digital world, where fraud is often cheap, easy, scalable, and not attributable.

Our security requirements can further be justified from the requirements explicitly listed by the ICRC in their report [131], to which we provided major feedback and input. In their report, the ICRC lists that "probing for a 'digital emblem' must not identify a cyber operator as a potential threat actor" and that emblems must be "obvious and easily visible." We address both notions with the positive security requirement covert inspection (**SR1**). Additionally, the ICRC requires that a digital emblem must be "trustworthy" and that "cyber operators [must be] able to verify [an emblem's] validity." We understand this as meaning that a digital emblem must be authentic (**SR2**). Noteworthy, accountability was not considered in the ICRC's report.

5.3 DESIGN

In this section, we start by giving an overview of ADEM and how its design meets the requirements just identified, then we proceed to technical details, and we close with an example of ADEM in practice. ADEM relies, in part, on the Web PKI, which associates domain names with TLS public keys, and the CT ecosystem, which monitors issued certificates. We introduced both the Web PKI and CT in Chapter 1 and Section 3.3 respectively. See [46, 149] for a more detailed overview.

[2] See also Section 5.1 for our reasoning on why a digital emblem should provide accountability.

5.3.1 *Overview of ADEM*

ADEM identifies protected assets via their network address, i.e., an address (IP address or domain name) and port combination, to be as *widely applicable* as possible (**FR2**). Clearly, an address/port combination can label network-connected processes such as servers. Moreover, one can protect entire networks using address prefixes or protect devices by labelling every port of an address.

Digital emblems in ADEM are cryptographically signed and mark a set of assets as protected. Assets send emblems to anyone interacting with them. This distribution mechanism supports *covert inspection* (**SR1**), that emblems are *removable* (**FR3**), and that their *presence can be verified* (**FR4**). An emblem can be verified as *authentic* (**SR2**) by verifying its signature, but naturally this requires authentic public keys.

ADEM specifies three types of public/private key pairs: *asset keys* used to sign digital emblems, *intermediate keys* used to manage key hierarchies, and *root keys* identifying organizations (authorities and EIs). As a means to obtain authentic public keys, ADEM specifies *endorsements*, which are certificates that attest a given public key as belonging to a given organization, and that this organization is believed by the endorsement's signer to be eligible to issue digital emblems. Asset keys are endorsed by intermediate keys, and intermediate keys are endorsed by root keys. The root keys of one party can additionally endorse the root keys of another party.

Verifiers can freely choose which endorsing parties they trust, which makes ADEM *decentralized* (**FR1**). We expect that most verifiers (typically military units) will only accept emblems accompanied by an endorsement from an authority they trust, like their own country or an ally. As we pointed out in Section 5.2.2, we assume that such verifiers can obtain the authentic public keys of their affiliated authorities through secure channels. Such a verification practice establishes a closed loop of trust: Military units can see that their own country endorsed the protected asset in question.

To provide *accountability* (**SR3**), ADEM requires organizations to commit to their root keys. The non-repudiation of digital signatures alone is necessary but not sufficient to provide accountability: a keyholder cannot repudiate signatures but could still repudiate key ownership. ADEM implements this commitment mechanism by utilizing CT logs. Organizations must publish their root keys within certificates that bind them to a central, HTTPS-enabled website identifying the organization, e.g., `https://emblem-issuer.org`, which we call their *organization identifier (OI)*.

This commitment mechanism comes with additional benefits:

1. Parties can monitor impersonation attempts by monitoring the CT logs.

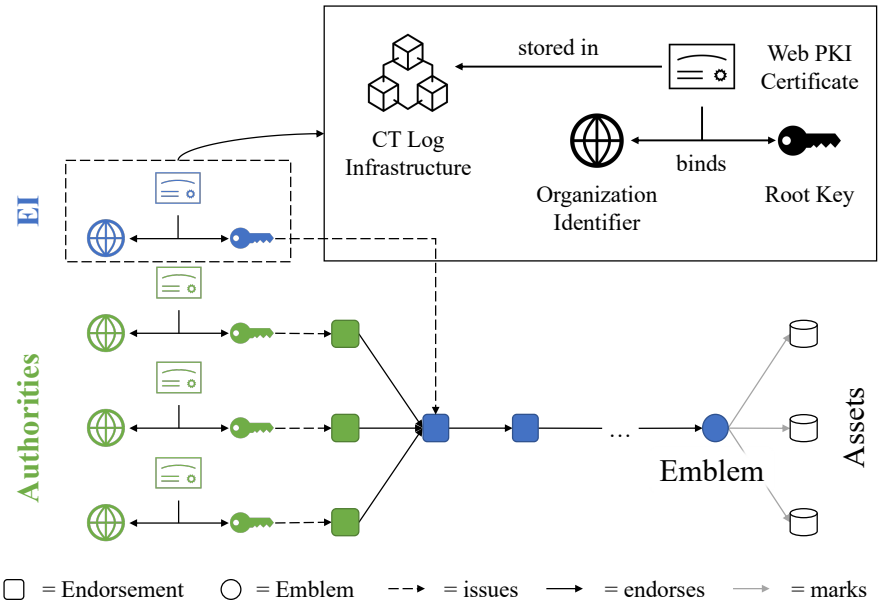


Figure 5.3: A typical ADEM setup. Multiple authorities (green) endorse an EI's root key. The EI (blue) maintains an internal hierarchy of public keys by signing multiple internal endorsements that ultimately endorse the emblem signing key as belonging to the EI. Parties commit to their root keys by submitting specifically encoded Web PKI certificates to CT logs, which are signed by standard CAs.

2. Parties can revoke their root keys by revoking the binding certificate.

3. Independent verifiers such as cybercriminals, who may not be able to obtain authentic root keys out-of-band, can authenticate root public keys as belonging to a domain name.

Figure 5.3 shows a typical ADEM setup. Authorities issue endorsements to an EI, and the EI manages an internal public key hierarchy. The authorities and the EI are identified by a domain and a root key. Finally, the EI issues an emblem, which marks multiple assets as protected.

In the rest of this section, we explain the syntax and semantics of emblems and endorsements and how organizations publish their root keys. We also explain how protected assets actively distribute digital emblems. Finally, we explain how verifiers obtain and verify digital emblems.

5.3.2 *Emblems*

We call both emblems and endorsements *tokens*. Emblems resemble the statement: *"Asset A belongs to EI P and is protected under IHL."*

Field		Purpose
`ass`		List of protected assets (AIs)
`iss`		Emblem issuer (OI)
`iat`		Time of issuance
`nbf/exp`		Validity window
`emb`	`prp`	Purposes
	`dst`	Distribution method

Table 5.1: Fields of an emblem. `emb` maps to a JSON object with two keys that constrain the contexts in which the emblem may be used.

```
asset-identifier = address [ ":" port ]
address = [ "*." ] domain-name
        | "[" IPv6 "]"

org-identifier = "https://" domain-name
```

Figure 5.4: Syntax of AIs, identifying network-connected processes, and OIs, identifying organizations. `port` is an integer. Domain names and IP addresses are encoded as usual.

Emblems encode this statement as JWSs [83], a popular standard for signed attribute-value stores, using the attributes listed in Table 5.1.[3]

The attribute `iss` encodes an organization (*who* issues a token) and `ass` encodes assets (*what* is protected). Organizations are identified by OIs, and assets by *asset identifiers (AIs)*. Both closely resemble URIs [18].

Figure 5.4 depicts the syntax of OIs and AIs. OIs are encoded as domain names prefixed by `https://`. AIs are an address (IP address or domain name) and port combination and point to network-connected processes. For example, `https://example.com` is an OI and `example.com:8080` or `[::FFFF:93.184.216.34]:22` are AIs. As AIs permit IP address prefixes and wildcards in domain names, one AI defines a set of protected assets, more concretely, protected processes. A process reachable under the IP address IP and port P is included in the set of an AI AI if both:

1. The domain name encoded within AI resolves to IP or to a prefix of IP, or the IP address encoded within AI is IP or a prefix of IP.

2. AI's port is equal to P, or AI does not include a port.

An emblem can include multiple AIs and marks all assets that these AIs point to as protected under IHL.

[3] In this section, we omit some technical details. We only include critical attributes, omitting those with purely technical purposes, such as version numbers. A complete list of artifacts is provided in ADEM's technical specification (see Section 1.6).

Field		Purpose
`iss`		Endorsement issuer (OI)
`sub`		Endorsed party (OI)
`key`		Endorsed public key
`nbf/exp`		Validity window
`log`	`ver`	Version of CT log
	`id`	ID of CT log
	`hash`	Certificate leaf hash
`emb`	`prp`	Purpose constraints
	`dst`	Distribution constraint
	`ass`	Asset constraints (allowlist)
	`wnd`	Max emblem lifetime

Table 5.2: Endorsement fields. `log` maps to an array of JSON objects that specify in which logs a root key-binding certificate was included. `emb` maps to a JSON object whose attributes constrain emblems signed under this endorsement. The constraints apply to the respectively named emblem attributes.

5.3.3 *Endorsements*

Endorsements are also encoded as JWSs and resemble the statement: *"Party P_1 attests that public key K belongs to party P_2 and that P_2 may signal protection under IHL."* They provide a means to obtain authentic public keys for emblem verification. Additionally, endorsements may come with constraints. For example, a country may endorse an EI to mark its website `humanitarian.org` as protected, but nothing else. Constraints allow EIs and authorities to mitigate the consequences of private key compromises and to prevent abuse within EIs. Technically, anyone can endorse anyone else, even oneself, e.g., to manage key hierarchies within one's own organization. In practice, we expect that most party-to-party endorsements will be signed by countries or supranational alliances like the Arab League or European Union.

Endorsements include the attributes listed in Table 5.2. An endorsement's constraints can specify upper bounds on an emblem's lifetime, over which channels emblems may be distributed, and what kind of assets may be labeled (listing permitted AIs). `log` lists CT logs that provide the party's root key commitment for the purpose of accountability, as we explain in Section 5.5.2.

5.3.4 *Root Keys*

Naturally, chains of endorsements (as specified in the previous section) will terminate in some key that is not further endorsed. We call these keys *root keys*. Recall that most verifiers will be able to authenticate the root keys of at least some authorities out-of-band (see Section 5.2.2).

Such verifiers can authenticate a claim of protection by verifying that it was (transitively) endorsed by a public key they trust. In designing ADEM to provide accountability, however, we do not rely on out-of-band authenticated public keys. ADEM should (and will) provide accountability also in cases where public keys cannot be authenticated by requiring parties to bind their root keys to their OI, which implements a root key commitment mechanism.

Our commitment mechanism enables verifiers to associate emblems and endorsements to domain name holders and to hold these domain name holders accountable for misbehavior. Further, this commitment mechanism enables verifiers to authenticate root keys of authorities they are not affiliated with, it allows parties to revoke root keys, and it enables organizations to detect impersonation attempts, i.e., attempts to maliciously associate public keys with their OI.

Our commitment mechanism works as follows. Organizations commit to a root key using a mechanism inspired by the concept of "self-authenticating traditional (SAT) domains" [154]. First, they calculate their root key's hash $h(pk)$ and register the subdomain $h(pk)$`.adem-configuration.`OI, for example:

```
1 z247co3xrah...danumgcfx7a.adem-configuration.ei.org.
```

Second, they request a TLS certificate [29] that is valid for both their OI and this new subdomain. Finally, they submit this certificate to the CT logs. A root key is specified as correctly associated to an OI if there is a certificate that (i) is not revoked, (ii) properly logged in the CT infrastructure, and (iii) binds the key to the OI as described above. Note that proper CT log inclusion requires that the certificate's signature is valid, and that there is a validation path to a root certificate accepted by the log in question.

ADEM does not specify how verifiers check a certificate's revocation status. We recommend, though, to use offline verification mechanisms that are employed and maintained by modern browsers such as Mozilla Firefox [120] or Google Chrome [45]. Using these mechanisms, verifiers would regularly download a set of revoked certificates that has been assembled by a third-party (Mozilla or Google, in the examples above). Using offline certificate revocation checks, verifiers only reveal that they use the mechanism at the time they update it. We will cover revocation checks in more detail in our security analysis (Section 5.5.3).

5.3.5 *Distribution*

ADEM specifies three interfaces for token distribution. Either, tokens are actively pushed to anyone interacting with a protected asset via TLS [129] or UDP [123], or they are stored in DNS records [115, 116].

In each case, ADEM distributes emblems with the endorsements necessary to cryptographically verify an emblem as genuine.

There is no limit to the number of interfaces ADEM could support, and extending ADEM to support additional interfaces is straightforward. However, the more interfaces ADEM supports, the greater the burden that ADEM puts on verifiers, who would need to implement detection on every interface to ensure that they are not attacking an EI.

TLS We envision TLS to be the most popular interface to transmit tokens. Many application-level protocols, such as HTTPS, run over TLS. Moreover, using TLS provides attractive security properties as tokens cannot be dropped by an on-path attacker.

Tokens distributed over TLS are appended as a custom extension to the `NewSessionTicket` message. This message is intended to establish pre-shared keys for session resumption, but is dropped by clients that do not support its extensions. Thus, our TLS distribution mechanism is backwards-compatible with existing TLS clients.

UDP To support the labeling of arbitrary network traffic, we also use UDP to distribute tokens. UDP does not guarantee reliability, so an adversary may drop UDP traffic that includes tokens. However, an emblem sent over UDP still provides authenticity and can prevent attacks when received. Whenever a protected asset receives a packet from a new network address, it sends tokens to a specified port at that address.

DNS Finally, protected assets can be labeled using DNS TXT records, which enable associating arbitrary information with a domain name [133]. Naturally, this is only possible if the respective asset has an associated domain name. A benefit of labeling an asset via DNS is that access to the asset is not required for distribution or verification and no additional software must be deployed.

Note that DNS distribution requires explicit queries from verifiers, suggesting a possible conflict with the *covert inspection* requirement (**SR1**). We will cover this tension in our security analysis later (Section 5.5.3).

5.3.6 *Determining Protection Status*

The distribution mechanisms just presented specify how a verifier could receive digital emblems given a domain name or given an asset with a port that runs TLS. Still, verifiers require a procedure to decide whether an *arbitrary* digital asset is protected under IHL. In particular, they must be able to determine this for digital assets *not* protected under IHL, while not alerting those assets about a planned attack.

To help verifiers decide whether an asset enjoys protection, assets must distribute emblems the following way. If an asset distributes emblems via UDP, the asset must monitor all incoming requests, e.g., via firewall rules. Whenever an asset receives a request from a new IP and port combination, it must send an emblem via UDP to that address. Assets need not parse incoming packets, however, and may respond in any way they like, e.g., with a TCP RST (reset), Internet control message protocol (ICMP) error messages, or not at all. If an asset distributes emblems via TLS, it must do so with every TLS connection. Such assets must also run TLS servers on ports 443 (HTTPS) and 853 (DNS over TLS). These servers can be minimal and solely distribute emblems.

To decide whether a given asset is protected, verifiers should run a procedure that depends on the address they wish to verify. Given a domain name, they should check the domain's TXT records for emblems and resolve the domain name to IP addresses to check those for digital emblems too.

Given an IP address, verifiers can send an arbitrary packet to that address and wait for an emblem in response via UDP. If they do not receive an emblem, they should also try to establish a TLS connection to port 443 or 853 and wait for an emblem to be sent as part of a `NewSessionTicket`. Should a verifier learn an asset's domain name only while checking for an emblem, e.g., during the TLS handshake, they should additionally check these domain names for emblems stored in DNS TXT records.

5.3.7 *Emblem Verification*

After probing an asset as just described, if a verifier has received emblems and endorsements, they will next verify them. The verification procedure takes as input: a digital emblem, a set of endorsements associated with the emblem (possibly empty), and a set of trusted public keys (possibly empty). For example, if the verifier were a military unit, they might use their country's root public key as a trusted public key. The verification procedure returns a set of *security levels*.

First, the procedure verifies every received token's signature and checks that it has not expired. The procedure discards any token that fails this check. The procedure also checks that the emblem is valid with regard to every endorsement that passed previous checks, e.g., that the emblem's designated protected assets comply with every respective endorsement's constraints, etc.

Second, the procedure determines and returns all the emblem's security levels, which indicate whether an emblem was endorsed and by whom. The security levels are as follows.

INVALID: If the emblem was discarded as invalid during preprocessing or no other security level applies.

UNSIGNED: The emblem does not bear a signature.

SIGNED: The emblem does not designate an issuer.

ORGANIZATIONAL: All tokens designate the same issuer, all endorsements transitively endorse the emblem's verification key, and the topmost endorsement's public key is correctly configured as the EI's root key (Section 5.3.4).

ENDORSED: Same as *organizational*, but additionally, there are endorsements with a different issuer than that of the emblem. All such endorsements' verification keys are correctly configured as the respective issuer's root key and endorse the emblem issuer's root key.

SIGNED/ORGANIZATIONAL/ENDORSED-TRUSTED: Same as the security levels *signed*/*organizational*/*endorsed*, but a trusted public key was used in the verification of the respective level.

The different security levels come with different security guarantees. Security levels other than endorsed were included upon request of the ICRC and increase ADEM's flexibility, e.g., enabling easy deployment in cases of emergency. EIs could announce their root or their asset public keys through other channels, allowing verifiers to authentically associate emblems to those parties even when they are only signed or organizational. In practice, though, we expect that most verifiers will only accept emblems with the security level *endorsed-trusted*.

If a verifier only wants to check if an emblem was endorsed by at least one trusted public key, they could skip the verification of correctly configured root keys, etc. Verification would then require no network calls beyond the initial probing. This leaves them vulnerable to certain kinds of attacks, e.g., they would not notice that an EI revoked its root key, but this highlights how simple verification can be. Even the full verification procedure for an endorsed emblem only requires additional network calls to check that the certificates binding a root key to an OI are correctly submitted to the CT logs. One must also check that these certificates have not been revoked. However, as we recommend using offline verification for these checks, their network-overhead is constant.

5.3.8 *Example*

We close this section with a brief example that showcases how ADEM might be deployed in practice. Consider an imaginary hospital network as depicted in Figure 5.5. The network is connected to the

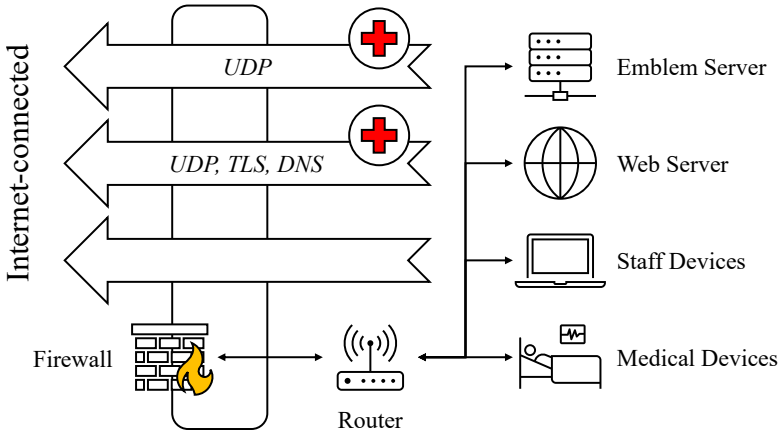


Figure 5.5: Imaginary hospital network

Internet via a router that also hosts a firewall, restricting outside access to certain assets. The network comprises four types of assets: network-connected medical devices, staff devices such as laptops and tablets, a web server advertising the hospital's services, and an emblem server that distributes emblems.

The medical devices do not have the technical means to distribute emblems themselves. The emblem server protectively marks these devices by regularly broadcasting a digital emblem via UDP to the local network. Such emblems serve as a defense in-depth against attackers who penetrated the hospital's network. Such an attacker may not have had a chance to inspect emblems from the outside, e.g., when they deployed malware via a malicious email attachment.

The staff's devices can monitor external traffic themselves, using firewall rules to log the traffic. Whenever they observe a packet from a new network address, they forward the address to the emblem server, which sends an emblem to that address via UDP.

Finally, the web server also marks itself as protected through all the interfaces available (TLS, UDP, and DNS). It requests emblems for itself at the emblem server each time its last emblem expired, but it need not request a new emblem for every recipient, as emblems are not recipient constrained. The server then sends those emblems to clients attempting to connect.

Presume that the hospital operates in a region of conflict between two countries: A and B. Utilizing endorsement constraints, both countries could endorse the EI in question to issue emblems within the IP prefix that is managed by the router. Whenever the countries' militaries would receive an emblem, they could see that their own country endorsed the issuing EI, giving them good reason to trust that the emblem is legitimate. But also independent verifiers could take the fact that two rivaling countries endorsed the EI as an argument to

trust emblems issued by the EI. The chance that two warring countries would collude to set up a fake "EI" is negligible.

This example highlights that ADEM is flexible in that it applies to a wide variety of protected assets and network setups, and that it enables a separation of concerns: not all protected assets need to be equipped with key material, only the emblem server requires that. Moreover, ADEM's design does not require the emblem server to authenticate emblem requests.[4] The emblem server could issue emblems for the IP-range of the hospital's network, and thus they would only apply to actually protected assets.

5.4 ACCOUNTABILITY FORMALLY

In this section, we prepare for our security analysis of ADEM and formally define accountability. In comparison to properties like secrecy or authentication, a standard definition of accountability does not exist, and a formal notion of accountability is harder to root in intuition. We introduce two notions of accountability: One by Küsters, Truderung, and Vogt [93] and one by Morio and Künnemann [118]. For reasons that will become clear later, we call Küsters, Truderung, and Vogt's notion *judge-accountability* and Morio and Künnemann's notion *formula-accountability*. We then compare both notions and explain why we follow judge-accountability in this thesis. Judge-accountability, however, cannot be directly applied using the Tamarin prover because Tamarin considers an adversary that is always active. The framework of judge-accountability has a more flexible adversary model and, in particular, can model that messages sent to the judge cannot be dropped by the adversary. We finally discuss how we adapt judge-accountability such that it can be applied using the Tamarin prover.

5.4.1 *Accountability according to Küsters, Truderung, and Vogt*

Küsters, Truderung, and Vogt [93] define their notion of accountability, judge-accountability, in a setting where participants communicate over secure channels, but can individually be compromised, and in which there is a dedicated protocol participant, the judge, that can blame individual protocol participants. They define accountability in terms of *accountability properties*. An accountability property is a set of *accountability constraints* of the form

$$C_i = \alpha_i \implies V_i^1 \mid \cdots \mid V_i^n,$$

where α_i is a condition under which the constraint applies, and V_i^j are *verdicts*, which blame parties that are misbehaving. Typically, α_i

[4] For the purposes of spam prevention, it would be desirable, though, that the server authenticates requests or throttles emblem generation.

encodes that a desired security goal of the underlying protocol was violated. A verdict V is a formula built from $\vee$, $\wedge$, and atoms $\mathsf{dis}(a)$, which indicate that a misbehaved. A protocol's judge is *fair* when they only blame misbehaving parties. A judge *ensures* an accountability constraint C_i if for every trace t either α_i does not apply to t or the judge blames parties in t that match some verdict V_i^j. A protocol provides an accountability property when its judge (i) is fair and (ii) ensures all accountability constraints.

5.4.2 *Accountability according to Morio and Künnemann*

Morio and Künnemann define an alternative notion of accountability, which we call formula-accountability. They integrate their framework into the Tamarin prover. The authors define accountability as a meta-property, i.e., in relation to a security property φ. Their definition of accountability uses three concepts:

COUNTERFACTUAL RELATION *Counterfactual relations* relate what happened with what could have happened. This relation formalizes which misbehavior *caused* a violation of a security property φ. The authors limit themselves to considering the *weakest counterfactual relation* r_w where $(t, t') \in r_w$ whenever a party being corrupted in t' implies they are also corrupted in t.[5]

A POSTERIORI VERDICT FUNCTION The *a posteriori verdict function* $apv_{r,\varphi}$ identifies misbehaving parties that caused (with respect to a counterfactual relation r) a violation of security property φ. Critically, $apv_{r,\varphi}$ has access to all traces and the counterfactual relation r. This means that it is defined from an omniscient point of view.

VERDICT FUNCTION A *verdict function* v identifies misbehaving parties on a single trace, i.e., without knowledge of the counterfactual relation and other traces.

A protocol provides accountability with respect to a security goal φ if $apv_{r,\varphi}$ and v identify the same parties as misbehaving. To formalize that parties identified by the a posteriori verdict $apv_{r,\varphi}$ indeed caused a violation, formula-accountability requires that $apv_{r,\varphi}$ only identifies parties as misbehaving on a trace t violating φ that also misbehave in every trace t' violating φ that "could have happened" (i.e., $(t, t') \in r$).

One goal of formula-accountability is to distinguish benign corruption from malicious misbehavior. Morio and Künnemann point out that corruption, e.g., by revealing long-term key material to the adversary, is not sufficient to cause misbehavior with respect to a security property violation. For example, the adversary could follow the

[5] See [92] for the consideration of different counterfactual relations.

protocol or deviate in benign ways, which may be impossible to detect in the real world, and thus too strong of a goal for accountability.

Morio and Künnemann provide five *verification conditions*, formalized with respect to a verdict function v, that can be verified with state-of-the-art protocol analyzers such as Tamarin. When all these conditions hold, the protocol provides formula-accountability. Concretely, these conditions check that v (i) only blames parties when φ is violated, (ii) only blames corrupted parties, (iii) is minimal in that v only blames parties that must be blamed, and is (iv) sufficient and (v) complete in that v blames parties if and only if they could have caused the violation of φ.

5.4.3 *Comparison of Accountability Frameworks*

First and foremost, we argue that judge-accountability is easier to understand and thus a more desirable target for verification. It requires non-trivial effort to convince oneself that a framework, defined by reference to a "counterfactual relation," an "a posteriori verdict function," and five verification conditions, as defined in formula-accountability, accurately captures the intuition of accountability. In contrast, judge-accountability defines the general shape that accountability properties have, and that they must be verified with respect to a judge. Thus, their framework allows for more clearly communicating which security guarantees a protocol achieves and under what assumptions.

The technical differences between judge- and formula-accountability can be reduced to three observations:

- Formula-accountability strives for completeness, i.e., all parties that caused a violation of φ are blamed; judge-accountability does not.[6]
- Judge-accountability considers a dedicated protocol participant that is the judge; formula-accountability does not.
- Formula-accountability considers an always-active network adversary; judge-accountability uses a more flexible adversary model.

Formula-accountability's focus on completeness requires a notion of causality. Only by reference to causality can one define when parties must be blamed by a verdict function. Küsters, Truderung, and Vogt instead argue that completeness if often too strong of a requirement. Many protocols do not provide for a way to identify all parties that caused misbehavior, or only under certain assumptions. They argue: "Altogether, rather than fixing the level of accountability

[6] Note that judge-accountability also has a concept called "completeness," but it is defined with respect to an accountability constraint and not with respect to the violation of a security property φ.

protocols are supposed to provide up front [e.g., completeness], it is more reasonable to have a language in which this can be described precisely, allowing to compare protocols and tell apart weak protocols from strong ones." [93]

By having a judge blame misbehaving parties, judge-accountability can disregard whether participants *actually* caused a violation of a desired security guarantee φ, which simplifies their framework. They can do so because the judge, being a protocol participant, must receive evidence to form its verdict. The judge can only accept evidence that is sufficient because the judge must be fair (correctly blaming only misbehaving parties). It is hard to picture a protocol in which a judge could identify "pointlessly" misbehaving parties while receiving sufficient evidence of misbehavior.

Not considering a judge also raises the question whether a verified verdict function v in the framework of formula-accountability could be computed in the real world. The verdict function has access to the entire trace and thus sees all messages exchanged between all participants. This is not the case for a judge as a protocol participant, who must be sent evidence which could potentially originate from the adversary. It is obviously more desirable to prove that a judge can identify some misbehaving parties than to prove that a function identifies all misbehaving parties.

For all the above reasons, we follow judge-accountability in this thesis. This, however, requires us to address the third difference between the two accountability frameworks: judge-accountability allows one to model that the adversary cannot interfere with communication between honest participants and the judge. In Tamarin, however, the adversary is always active and cannot be prevented from dropping all messages to the judge, which in turn would prevent the judge from stating a verdict.

5.4.4 *Accountability against an Always-Active Network Adversary*

We next discuss how we adapt judge-accountability to Tamarin, which is not straightforward as Tamarin considers an always-active network adversary as explained above. In Tamarin, the adversary's capability to drop messages is modeled implicitly. A trace admitted by a set of multiset-rewrite rules does not require all applied rules' conclusions to be used. This applies, in particular, to messages sent by participants, even over secure channels. A trace where a security property is violated and where a judge states a verdict will typically be an extension of a trace in which only the security property is violated. Thus, the trace that only violated the desired security property will in general also be a valid trace, but this means that the judge does not make a verdict and thus does not ensure an accountability property's constraints. Intuitively speaking, the violation of a security property

does not imply that a judge makes a verdict. For protocols providing accountability, it only implies that a judge *could* make a verdict if they received the necessary evidence.

We address this problem by strengthening the accountability constraints' conditions such that they require an active judge. Our accountability constraints are trace properties of the form:

$$\forall \vec{x}.\psi(\vec{x}) \wedge J(\vec{x}) \implies V(\vec{x}).$$

Here, ψ is an assumption under which the constraint holds, J models which evidence the judge requires to make a verdict, and V models the judge's verdict that identifies misbehaving parties. ψ and J together stand in for α as defined in judge-accountability (see Section 5.4.1). When verifying such accountability constraints using Tamarin, one establishes the "judge's" fairness. If the property holds on all traces, it means that the evidence formalized in $J(\vec{x})$ is strong enough to imply that identified parties indeed misbehaved according to $V(\vec{x})$, provided that $\psi(\vec{x})$ holds.

This approach leaves us with two problems. The first problem is that verifying accountability constraints of the above form does not establish completeness in the sense of judge-accountability. This is to be expected when considering an always-active network adversary as discussed previously. We see the goal of completeness to establish an accountability property's relation to a desired security guarantee. As an alternative for completeness, we will discuss our accountability property's relation to other security properties when presenting our security analysis in Section 5.5.2.

The second problem is that our approach suffers from the same issue as formula-accountability: How can we know that a judge as modeled in J can exist? Given an arbitrary condition J, it could be impossible for a *real* judge to decide whether J is the case (from their viewpoint as a protocol participant). We addressed this issue by modeling the constraints of our accountability properties such that they only referred to facts that could be checked by a real judge. In particular, rules using facts referenced in J only receive input from the untrusted network, use environment assumptions, or we provide alternative version of the same rules that can be triggered by the adversary directly.

5.5 SECURITY ANALYSIS

Equipped with a formal definition of accountability, we proceed to our formal analysis of ADEM. ADEM is designed to provide three security properties: authentication, accountability, and covert inspection (see Section 5.2.3). In this section, we define our threat model (Section 5.5.1), formalize ADEM's authentication and accountability guarantees, and prove that ADEM provides both properties using

Tamarin (Section 5.5.2). Finally, we provide a separate security argument for covert inspection (Section 5.5.3).

5.5.1 *Threat Model*

We consider an active network adversary who can inject, intercept, read, reorder, and replay any message and can corrupt any party's keys. We only constrain our adversary by the following assumptions:

1. EIs only issue endorsements for keys under their control, and their root keys are uncompromised.
2. Internal EI endorsements are constrained to only permit the issuance of emblems for protected infrastructure.
3. EIs only use root keys that are endorsed by at least one authority.
4. At least some authorities who endorsed an EI did so honestly. Some may endorse illegitimate parties as EIs, but they cannot convince all other authorities to endorse such parties.
5. Organizations monitor CT logs for fraudulent public keys associated to their OI.
6. Verifiers have access to a trusted offline revocation mechanism.

While Assumption 1 expresses that we do not consider the compromise of EI root private keys, EIs can recover from root key compromise through revocation in practice. Assumption 2 implies that the compromise of EI non-root private keys has no effect as the respective endorsements' constraints prevent any misuse.

Assumption 3 follows the idea that we laid out in Section 5.2.2: we can assume that most verifiers will be affiliated with one or more authorities and will only accept emblems endorsed by those authorities, i.e., will only accept emblems with the security level *endorsed*. Moreover, Assumption 4 expresses that the authority of a verifier's choosing is indeed to be trusted (all but the trusted authorities may be compromised). Assumption 5 concerns the Web PKI domain: we assume that parties utilize the CT logs to monitor their own domains.

Finally, Assumption 6 reflects that we do not consider revocation mechanisms in our formal model explicitly. As ADEM does not specify the precise means verifiers use to check for revocation, a formal model of all the possible ways to do this is out of scope. Nevertheless, we will still consider the implications of revocation for authentication and accountability (Section 5.5.2) and covert inspection (Section 5.5.3).

To put our security analysis into context, recall that in Section 5.2.2, we introduced adversaries as those agents who do not respect IHL and seek to abuse digital emblems. Note that adversaries who only disregard IHL and are willing to attack unprotected entities cannot be defended against using a digital emblem. As we noted in Section 5.2.3: An emblem can only prevent attacks by those who respect it. The goal

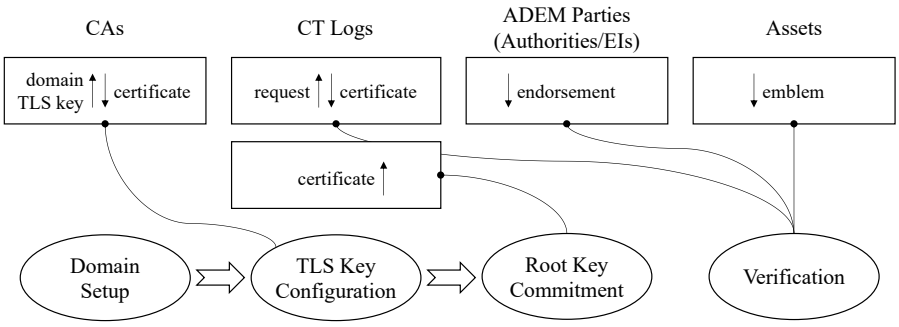


Figure 5.6: Components of our formal model. We model four kinds of parties with five interfaces (e.g., CAs have an interface for certificate signing). The model supports two main protocol flows: EI endorsement and emblem distribution. Lines with dots at the end denote that the (partial) protocol flow uses the connected interface.

of our security analysis is to establish that digital emblems cannot be abused, e.g., to mark unprotected infrastructure.

5.5.2 *Formal Analysis of Authentication & Accountability*

We explain next how we formally modeled ADEM and proved authentication and accountability using Tamarin. Our formal model comprises 327 lines and 15 lemmas, and it required the implementation of a proof heuristic in Python and auxiliary lemmas for the proofs to terminate. Except for one lemma, all lemmas can be proven automatically when using our proof heuristics. Using a laptop with 32 GB of RAM and an Apple M2 Max CPU, Tamarin can verify all lemmas in 87 s.

Formal Model of ADEM

Figure 5.6 provides an overview of our Tamarin model. It includes CAs, CT logs, authorities, and EIs as protocol participants. These parties control keys associated with different protocols: TLS/HTTPS, certificate signing, and emblem/endorsement signing. Since the CT specification requires that logs must be monitored such that they preserve the append-only property, we model CT logs such that certificates submitted to them can never be removed by using persistent facts. This means that our model abstracts from an ordering on log entries. The adversary can, though, include arbitrary entries in compromised logs.

Our protocol model consists of two phases. In the first phase, we non-deterministically assign domain and subdomain names to parties, TLS keys and certificates to these domains, and let ADEM parties commit to root keys by requesting the corresponding Web PKI certificates. In the second phase, a verifier receives an emblem and a non-deterministically determined number of endorsements and

```
1  All id oi asset ak rk #t1 #t2.
2      ( Emblem(id, oi, asset, ak) @ #t1
3      & RootEnd(id, oi, rk) @ #t2)
4  ==> ( (Ex ei #x #y. OI(ei, oi) @ #x
5          & IsAsset(ei, asset, ak) @ #y)
6      | (Ex p #x #y. OI(p, oi) @ #x
7          & CompromisedParty(p) @ #y)
8      | (Ex otherA #x.
9          CompromisedKey(otherA, ak) @ #x)
10     | (not Ex oiA rkA endK #t3.
11         AuthEnd(id, rkA, oiA, oi, endK) @ #t3)
12     | (All oiA rkA endK #x.
13           AuthEnd(id, oiA, rkA, oi, endK) @ #x
14       ==> ( (Ex p #y #z. OI(p, oiA) @ #y
15               & CompromisedParty(p) @ #z)
16           | (not Ex p #y.
17               IsRootPK(p, oiA, rkA) @ #y))))
```

Figure 5.7: Tamarin formalization of the authentication property. `Emblem` represents that a verifier received and verified an emblem. `RootEnd` and `AuthEnd` represent the same for EI and authority endorsements respectively. `IsAsset` expresses that `asset` is a protected asset and `IsRootPK` associates an authentic public key to an OI. Lines 6f.,8f., and 10f. represent Assumption 1, 2, and 3 respectively.

verifies them. Note that we do not enforce the phases' order, i.e., our proofs also cover executions where the phases are interleaved.

We modeled the adversary as able to compromise any party and their keys. Moreover, all certificates, emblems, endorsements, and the like were modeled as being distributed over an insecure network.

Authentication

Within the model just described, we proved that ADEM provides the following authentication property:

Security Property (Authentication)**.** If an endorsed emblem is successfully verified then the emblem claims protection for an actually protected asset that belongs to the organization endorsed.

ADEM provides this property using chains of endorsements that root in a set of (not all compromised) authorities. Interestingly, and perhaps surprisingly, authentication does not rely on the security of an EI's domain that they use as their OI. We only utilize an EI's OI to commit to root keys, not to authenticate them. In practice, this means that ADEM can handle a partial failure of the Web PKI: Even if the adversary managed to obtain a malicious Web PKI certificate for the EI's OI, they could only succeed in forging unauthentic claims of protection if they additionally compromised some authorities' OIs for which the verifier has no other means of obtaining authentic public

keys. Recall, however, that it will likely be trivial for, e.g., military units to obtain at least one authority's authentic public key, namely their own country's public key (Section 5.2.2). Moreover, the fewer external endorsements an emblem is accompanied by, the less believable it is. For example, similarly to the CT log infrastructure, we foresee that emblems will be required to be accompanied by some minimal number of external endorsements before they are accepted.

The formalization of ADEM's authentication property is given in Figure 5.7. The outer implication's left-hand side encodes that a verifier received and successfully verified an emblem with an accompanying root endorsement from a (proclaimed) EI. The right-hand side encodes that either the emblem marks a protected asset of that EI (line 4), or one of our threat model's assumptions was violated (see Figure 5.7 for details), or that for all authorities, either their root key was compromised (lines 14f.) or an unauthentic key was used to verify their endorsements (lines 16f.).

Accountability

We next present our formalization of ADEM's accountability property, which follows the framework presented in Section 5.4.4. Proving that ADEM provides accountability was significantly more challenging than proving that it provides authentication, as it required us to identify precise and concise conditions of the constraints. Our accountability property comprises three accountability constraints.

Security Property (Accountability)**.** The following accountability constraints specify when we can identify parties as misbehaving:

CA ACCOUNTABILITY If a domain owner honestly reports a fraudulent certificate issued for their domain and that certificate is included in a CT log, then the signing CA misbehaved.

AUTHORITY ACCOUNTABILITY If an authority illegitimately endorsed an EI, the authority misbehaved.

EI ACCOUNTABILITY If an illegitimate asset is marked as protected, and an EI endorsed the signing key, the EI misbehaved.

The CA accountability constraint is straightforward, as CT was designed to provide it. The other accountability constraints go beyond what CT provides for the Web PKI. They are conceptually similar: one can blame key holders (authority or EI), when they used their authentic root key to endorse a malicious key (either as a root or asset key).

The three constraints' formalizations are depicted in Figures 5.8-5.10. The fact `Dispute` models a party `p` claiming that a malicious certificate was associated with their domain. Since it is trivial to claim that legitimate certificates are fraudulent, we permit the adversary

```
All p log ca d pk skCA #t2 #t3.
    // Condition
    ( (not Ex #a. CompromisedParty(p) @ #a)
    // Judge input
    & Dispute(p, log, ca, d, pk) @ #t2
    & LogInclusion(log, cert) @ #t3)
    // Verdict
==> (Ex #a. CompromisedParty(ca) @ #a)
```

Figure 5.8: Formalization of CA Accountability. *cert* is a signed certificate from CA `ca`, binding `d` to `pk`.

```
All pAuth oiAuth rkAuth oi rk id #t1 #t2.
    // Condition
    ( IsRootPK(pAuth, oiAuth, rkAuth) @ #t1
    & (not Ex p #x. IsRootPK(p, oi, rk) @ #x)
    // Judge input
    & AuthEnd(id, oiAuth, rkAuth, oi, rk) @ #t2)
    // Verdict
==> (Ex #a. CompromisedParty(pAuth) @ #a)
```

Figure 5.9: Formalization of Authority Accountability.

```
All p oi rk e1 assetKey id #t1 #t2 #t3.
    // Condition
    ( IsRootPK(p, oi, rk) @ #t1
    & (not Ex e2 #x. IsAsset(p, e2, assetKey) @ #x)
    // Judge input
    & RootEnd(id, oi, rk) @ #t2
    & Emblem(id, oi, e1, assetKey) @ #t3)
    // Verdict
==> (Ex #a. CompromisedParty(p) @ #a)
```

Figure 5.10: Formalization of EI Accountability.

```
1  (All id oi rk #t. RootEnd(id, oi, rk) @ #t
2  ==> UsedRootKey(oi, rk) @ #t) &
3
4  (All id oiAuth rkAuth oi rk #t.
5      AuthEnd(id, oiAuth, rkAuth, oi, rk) @ #t
6  ==> UsedRootKey(oiAuth, rkAuth) @ #t) &
7
8  (All oi rk #t.
9      UsedRootKey(oi, rk) @ #t
10 ==> (Ex ca caSk log c1 c2 k #x #y #z.
11       CASk(ca, caSk) @ #x
12     & c1 = <'cert', ca, oi, k>
13     & c2 = <'cert', ca, <oi, sha256(rk)>, k>
14     & InLog(log, <c1, sign(c1, caSk)>) @ #y
15     & InLog(log, <c2, sign(c2, caSk)>) @ #z))
```

Figure 5.11: The above property formalizes that whenever a root key is used to verify authority or EI endorsements (`AuthEnd` and `RootEnd` respectively), this key must be committed to the CT log infrastructure. `c1` and `c2` model the body of the root key binding certificate. We modeled certificates that are valid for multiple domains using two certificates that are valid for the same key (here `k`).

to dispute any certificate and key they like. Thus, our accountability constraints must assume honest disputes as pointed out above. This assumption is formalized as the condition that `p` is not compromised (line 3 in Figure 5.8). The constraints furthermore assume that the judge uses the authentic root keys of the blamed authority or EI respectively (lines 3 and 3 in Figures 5.9 and 5.10 respectively).

COMPLETENESS Completeness as defined by Küsters, Truderung, and Vogt ensures that when an accountability constraint's condition applies, i.e., a desired security guarantee of a protocol is violated, the judge makes a verdict. As we discussed in Section 5.4, completeness cannot be achieved against an always-active network adversary. Nevertheless, we next discuss how our constraints relate to ADEM's authentication property.

As per our threat model Assumption 5, we require that parties monitor CT logs for root keys that are fraudulently associated with their OI. Thus, verifiers can assume that unrevoked root keys are authentic. If they were not, the party controlling the respective OI would have disputed them and revoked the certificates that bind these keys to their OI. Furthermore, whenever someone uses a party's root key to verify an emblem, that root key must be committed to in the CT log infrastructure, which we formalize as an additional property in Figure 5.11. Proving this property is straightforward as it follows directly from the ADEM specification (Section 5.3.4), but it is vital nonetheless. All the above justifies our constraints' assumption that

the judge uses authentic root keys to verify authority and EI key endorsements (lines 3 and 3 in Figures 5.9 and 5.10 respectively).

Suppose an unprotected asset were labeled. Our accountability property addresses two cases why this could have happened that are included as part of our authentication lemma. If the EI were compromised, i.e., maliciously endorsed an asset key (lines 6f. in Figure 5.7) we can hold it accountable as per EI accountability. Observe that the judge input in Figure 5.10 matches the outer implication's left-hand side of Figure 5.7. Otherwise, if all authorities that endorsed the EI were compromised (lines 14f. in Figure 5.7), these authorities could have fraudulently endorsed an EI. In that case, we can hold each of these authorities accountable as per Authority Accountability. CA Accountability does not directly relate to our authentication property. It simply points out conditions under which misbehaving CAs can also be identified.

Note that a judge still must determine whether a labeled asset indeed enjoys protection under IHL or whether an authority maliciously endorsed an EI. We do not formalize all aspects of this process as it is rooted in law. In our model, we non-deterministically explore whether an asset is protected and whether an authority endorsed a legitimate EI but cannot consider all possible, legal reasons why this might be the case.

Discussion

DISHONEST DISPUTES Recall that CA Accountability assumes honest disputes of malicious certificates. In practice, parties could dishonestly dispute certificates committing to root public keys in an attempt to evade being held accountable. For example, Authority Accountability holds a malicious authority accountable when the judge sees a malicious authority endorsement that was signed using the authority's authentic root public key. The accused authority could attempt to contest the constraint's assumption that the authority's authentic root public key was used by disputing it after the fact.

Even in such cases, however, our accountability constraints apply as parties are required to monitor their OI for maliciously associated root public keys. In the example above, it could indeed be the case that the malicious endorsement was signed using an inauthentic root public key. In both cases, however, the party in question can be held accountable. Either because it signed the malicious endorsement or because it did not monitor its OI for maliciously associated root public keys.

MODELING ABSTRACTIONS To support the automation of our formal proofs, our formal model employs abstractions, e.g., regarding the IP and BGP, DNS, and certificate issuance and revocation. In this section, we discuss these abstractions and how they affect our findings, in particular how our results relate to the real-world.

We assume that parties can obtain authentic CA and CT log keys. For simplicity, we do not model intermediate CA certificates. This should not affect our findings, as intermediate certificates would change our model only in so far that whenever we speak about one CA now, we would instead need to speak about a set of CAs. In practice, it would require just a bit more work to determine which CA precisely was to be blamed in case of misconduct.

In Section 5.5.2, we describe that we modeled CT logs assuming that they are properly monitored and thus cannot equivocate. When a CT log equivocates, it misbehaves by providing different clients with different views of the log. [165] recently showcased how utilizing anonymous routing, such as the Tor network, can help to avoid the need for monitoring: If an equivocating CT log cannot see who queries them, they can only guess when providing different clients with different views, defeating the purpose of equivocation. Thus, we suggest that verifiers contact CT logs using anonymous routing protocols should they wish to defend themselves against equivocating CT logs.

Additionally, we do not model CT precertificates, a mechanism that includes certificates in logs prior to their issuance. Precertificates must also be signed by the issuing CA and should thus be covered by our model of standard certificate inclusion. We also do not model internal endorsements of EIs for their own infrastructure and endorsement constraints as, under Assumption 2, internal endorsements must always be constrained in such a way that the endorsed key can only be used for legitimate purposes. Thus, the adversary cannot violate the authentication property by compromising intermediate or asset keys.

Finally, one additional threat vector to ADEM lies in the IP/BGP protocols and DNS. Namely, as emblems identify protected assets via their domain names or IP addresses, an adversary could gain illegitimate protection by hijacking either the respective DNS records and have it point to their IP address, or by BGP hijacking an IP address to have their assets appear as if they were in control of that IP address. However, such attacks would have to be conducted at a massive scale as, by the covert inspection property, adversaries cannot know who wants to attack them. And if they knew who wanted to attack them, it would make little sense to launch a BGP hijack merely to thwart an attack: the adversary could instead deploy more targeted, and cheaper countermeasures, such as dropping all traffic from the adversary.

CERTIFICATE ISSUANCE AND REVOCATION We model CAs as flawlessly authenticating certificate requests. We do model fraudulent certificates as we allow CA key compromise, but our model captures two distinct ways how an adversary might get hold of a fraudulent certificate in the same way. An adversary can either have access to a CA's private key, or they can trick the CA. For example, if they managed to compromise a party's DNS entries, they might be able to

obtain a fraudulent certificate without the CA acting maliciously. As a consequence, the property that the CA is compromised subsumes both cases just mentioned in our model. In practice, though, this has only limited impact as an adversary would still need to compromise *all* non-malicious authorities' OIs to associate a fraudulent key with the OIs or significantly reduce an emblem's believability by dropping the endorsements of uncompromised authorities.

As for revocation, we do not consider the publication of fraudulent certificate revocations. Such revocations constitute a denial of service attack and ADEM does not provide availability guarantees. Beyond availability, we can abstract possible misbehavior into two categories. Either the CA in question does not publish a certificate revocation, or the offline revocation mechanism does not provide the respective revocation. In both cases, the party in question might misbehave intentionally or due to secret key compromise.

In the case of a misbehaving CA, the EI can easily detect this misbehavior and take actions accordingly (they will see that the revocation was not published). To mitigate the cases of misbehaving offline revocation mechanisms, verifiers can consult multiple such mechanisms. In either case, we recommend short-lived root key endorsements to mitigate any kind of key compromise. The short-lived endorsements can be realized easily as endorsements are neither confidential, nor does resigning require new requests. Authorities could provide newly signed endorsements regularly and publicly after they initially approved a request.

5.5.3 *Covert Inspection*

Covert inspection is independent of authentication and accountability. Moreover, adversaries trying to violate this property have completely different motivations: The adversary seeks to distinguish processes that pay attention to an emblem from those that do not, given a process's network transcript and no matter whether it interacts with a protected or an unprotected asset. In this section, we first explain why a formal analysis for covert inspection is infeasible, and after provide a security argument.

One could formalize the idea of covert inspection in a game-based setting where the adversary is given two processes (one checking for the emblem and the other not) and tasked with identifying the one that checks for the emblem. A digital emblem scheme would then provide covert inspection if there cannot be an adversary that recognizes the emblem-checking process with a chance non-negligibly higher than the chance to guess correctly.

Unfortunately, such a definition would be too strong. On the one hand, there are certain classes of programs that can be trivially recognized as non-emblem checking, for example, any process sending no

network packets, or programs not using TLS to interact with an asset only signalling protection via TLS. One would need to rule out certain classes of "obviously non-emblem checking" processes; however, it is unclear how one could define such classes without rendering the security definition tautological. On the other hand, the adversary could gain a non-negligible advantage over random guessing by checking if one of the processes sent "non-standard traffic" to the respective asset. Web-servers, for example, see different "standard traffic" than database servers. One would need to represent these differences in typical traffic in the security game, but this would require a classification of digital assets, and a probability distribution on "typical interacting processes" for each of these classes. This has, to the best of our knowledge, not been previously considered in the literature.

Both of these problems put a formal analysis out of reach. Instead, we analyze what a verifier must do to check an emblem and how these steps might reveal them. The goal of the verifier is to generate traffic that "does not stand out" from standard traffic to the respective asset so that they can hide in a sufficiently large anonymity set of clients performing standard queries. A verifier will take the following steps to decide whether a given asset is protected (see Section 5.3.6 and 5.3.7):

- Check DNS TXT, A, or AAAA records of a (possibly adversarially provided) domain name. TXT records may contain tokens, and A and AAAA records map domain names to IPv4 and IPv6 addresses.
- Check (possibly adversarially chosen) CT logs for the inclusion of root key-binding certificates.
- Send an arbitrary packet to the asset.
- Attempt to perform a TLS handshake with one of the ports 443 (HTTPS) or 853 (DNS over TLS).

The first two steps are necessary to check for emblems in DNS entries and to verify root key setups. To identify a client as emblem-checking through these two steps requires monitoring the client's traffic. This could be done either by an on-path adversary, or because the adversary controlled the DNS servers or CT logs in question. However, in both cases, verifiers are unlikely to be detectable or can take easy steps that reduce the probability of being detected.

Regarding an on-path adversary, traffic to DNS servers and CT logs is usually encrypted, i.e., one could unlikely determine the true contents of queries. Additionally, the verifier could use a virtual private network (VPN) to mask the service they query. Regarding an adversary who controls DNS servers or CT logs, both types of servers usually receive so many queries that an adversary monitoring all these would have a high false-positive rate in identifying verifying clients.

And this constitutes exactly what the verifier needs: a large anonymity set. Additionally, the verifier can choose the DNS servers they want to query, reducing the likelihood that the respective server colludes with the adversary.

The same reasoning applies to the packets that the verifier must send to the asset directly. For UDP distribution, we neither require the verifier to send specific packets nor define the port they must contact, verifiers (i) will likely query these ports in the intelligence phase of their planned attack anyway, and (ii) can hide in other traffic that these ports are likely to receive. These two points apply to TLS distribution as well. Verifiers only need to attempt to *connect* to two ports that are well-known to typically use TLS.

Finally, we recommend that verifiers also perform offline revocation checks and thereby regularly update these mechanisms. However, offline revocation mechanisms by their very nature cannot reveal which certificates the verifier is interested in. Beyond that, mainstream offline mechanisms are maintained by, e.g., Google for Chromium-based browsers [45] and Mozilla Firefox [120], which gives verifiers a large anonymity set of clients to hide in when updating their mechanisms.

5.6 RELATED WORK

In this section, we discuss well-known authentication systems and highlight their differences to ADEM and how covert inspection relates to anonymity.

5.6.1 *Authentication*

AUTHENTICATED DATA STRUCTURES Instead of endorsing EIs, authorities could maintain lists of protected assets by using authenticated data structures, for instance, using transparency systems as introduced in Section 3.3. With such data structures, however, it is hard to maintain the covert inspection property. Assets would likely need to inform clients connecting to them about the logs that they are included in, e.g., by providing clients with a URL. An adversary could easily direct clients attempting to verify an asset's protection to a honey pot instead of a log. The honey pot would not even need to be such a log: the mere connection to a URL would inform the adversary about an imminent attack. At the same time, emblems could not be removed from protected assets, only invalidated.

To avert the honey pot problem, one could instead aim for a global directory of protected assets or public keys that are eligible to claim protection, maintained by consensus protocols, e.g., establishing Byzantine fault tolerance [94]. This would still not solve the issue of removing emblems from assets. Even more critically, consensus

protocols require (by definition) *consensus*, which cannot be assumed in the context of a digital emblem: the parties who must reach a consensus may be at war which each other. To avoid the need for consensus, one could instead include different parties' claims about who enjoys protection, but then one would require an additional mechanism to authenticate these claims, rendering the addition of such logs pointless.

ADEM does rely on authenticated data structures, namely the CT log infrastructure. However, we do not utilize these logs for the authentication of protected assets but rather for accountability.

CERTIFICATE-BASED AUTHENTICATION In certificate-based approaches, most notably the Web PKI [46], authorities attest that certain public keys belong to certain identities. It is clear that adapting the Web PKI to our purposes would introduce many practical hurdles. X.509 certificates [29] have different semantics than emblems and endorsements. Additionally, existing CAs can hardly authenticate EIs or authorities as such, and likewise, EIs and authorities can hardly become CAs in the Web PKI.

However, ADEM shares some similarities with the Web PKI or parts thereof. Internal endorsements resemble proxy certificates in the X.509 ecosystem [163] (a standard for Web PKI key delegation), endorsement constraints resemble X.509 name constraints [29], and altogether, endorsements resemble standard certificates.

In contrast to the Web PKI, we can hold key-holders, i.e., parties using ADEM, accountable and not just CAs and CT logs. Moreover, ADEM's design centers around the idea that verifiers can choose the "root CA" (authority) they want to trust, rather than relying on a small set of "root CAs" that then endorse other authorities.

ADEM's trust model does not implement a Web of Trust [40] either, although ADEM supports certificate chains of arbitrary length and with arbitrarily many "root CAs." Authorities endorse EIs directly, and, as there are no other party-to-party endorsements, "certificate" (endorsement) chains between different issuers always have length one. Verifiers must choose the authorities they directly trust and do *not* "transitively trust" them.

5.6.2 *Covert Inspection, Anonymity, and Undetectability*

In this section, we compare covert inspection to the definitions of anonymity and undetectability, two properties that both seem related to covert inspection. However, neither matches covert inspection.

Anonymity itself is often defined as that a subject of an action is not identifiable [122]. This definition of anonymity has, e.g., been applied in the analysis of systems like the Tor network [63], but it does not suit our needs: It may already suffice for an adversary to know that

someone verifies an emblem rather than to know *who*. Depending on the adversary, though, tools that provide anonymity can still help with covert inspection, e.g., onion routing could help against an on-path adversary trying to identify verifiers.

[122] additionally introduces a definition of *undetectability*: the "[u]ndetectability of an item [...] means that the attacker cannot sufficiently distinguish whether it exists or not." This definition is very similar to covert inspection, but it still does not quite match. ADEM requires verifiers to make *some* queries, e.g., the resolution of a DNS TXT record. An adversary trying to misuse ADEM as a honeypot might notice the queries themselves. However, the adversary could not use them to identify verifiers reliably (see Section 5.5.3) because queries as part of ADEM are "too standard" and would be performed by many non-verifiers as well.

Undetectability was studied in the analysis of steganography [85, 98], which again has similarities with covert inspection. In steganography, two parties try to exchange information over an insecure network such that an adversary monitoring all traffic cannot detect the presence of that communication. In contrast, in covert inspection, verifiers (i) do not send hidden information, and (ii) possibly directly communicate with the adversary.

5.7 CONCLUSION

In this chapter, we presented ADEM, a design that implements a digital emblem, enabling the marking of digital assets as protected under IHL analogously to the physical emblems of the Red Cross, Red Crescent, and Red Crystal. ADEM is a decentralized design that provides authentication, accountability, and covert inspection. Moreover, ADEM can be deployed by EIs autonomously and does not require updating the Internet's infrastructure. We evaluated ADEM through a formal and informal security analysis, and through a series of meetings with domain experts that were conducted in 2021 at the invitation of the ICRC. Both evaluations show that ADEM is secure and that it should fit the needs of EIs and the ICRC in practice.

FUTURE WORK We see four directions for future work. First, ADEM's security analysis could benefit from improved reasoning about the completeness of accountability constraints as we presented them in Sections 5.4.4 and 5.5.2. Probably, this would require adding support for liveness properties to tools like Tamarin such that judges can be modelled. Second, our security rationale for covert inspection could be improved by a formal security argument, which, however, would need to be developed first. Third, there are attacks that do not interact with their targets over the network, such as malware in malicious e-mail attachments or malicious JavaScript. To address this gap "Oblivious

Digital Tokens" were proposed, which make digital emblems presentable on devices to, e.g., malware [104]. ADEM would greatly benefit from integrating this new way of distributing digital emblems. Finally, we plan to standardize and deploy ADEM in collaboration with the ICRC. During the standardization process, which is about to begin at the IETF, we plan to align all technical details with other stakeholders. For example, Microsoft has been particularly supportive of the standardization efforts, but also other major service providers have joined the efforts.

Part II

AUTOMATED ANALYSIS OF LOOPING PROTOCOLS

6 THE CHALLENGE OF LOOPING PROTOCOLS

Proving properties of looping protocols using automated, state-of-the-art protocol verification tools such as Tamarin and ProVerif, is notoriously hard. So hard, in fact, that it was commonly believed that "unbounded (looping) protocols like Signal, and protocols with mutable recursive data structures [...] are [...] out of scope for symbolic provers [such as Tamarin], without introducing artificial restrictions" [19]. Proof techniques such as induction and the use of auxiliary lemmas, which are supported by Tamarin [111] and ProVerif [26], can help reduce the proof search space and alleviate some of the non-termination problems introduced by looping protocols. However, stating inductive conjectures and useful auxiliary lemmas requires user ingenuity and expertise, which hampers proof automation.

The complexity of trace induction may be one of the reasons why it has not yet been applied to the analysis of protocols using double ratchets such as Signal [58, 91, 110, 121] or iMessage PQ3 [2]. Reasoning about such protocols is non-trivial and entails reasoning about unboundedly many parallel instances of the protocol, where the runs (two devices sending messages) are themselves unbounded. Previous works analyzing double ratchet protocols in the symbolic model either did not consider their looping behavior at all or abstracted it drastically, for example, by not modeling the inner ratchet [19, 22, 54, 88].

A notable exception is the formal analysis of the message layer security (MLS) protocol in DY* [161, 162]. While technically not a double ratchet protocol, MLS generalizes the idea of the double ratchet to group messaging and is even more complex. DY* builds on the program verifier F* [153] and uses dependent types. It maintains a global trace variable that is used to express security properties. DY* is highly expressive and allows for intricate proof strategies, which comes at the expense of requiring significant manual interaction as DY* proofs require user-specified invariants on the global trace that are strong enough to imply the desired security property.

We would like to provide more proof automation with fewer auxiliary lemmas. To do so, we present how one can analyze looping protocols with automated protocol model checkers such as Tamarin in their full complexity using two induction mechanisms in this part of the thesis. In Chapter 7, we focus on trace induction and formally analyze iMessage PQ3 [2], a protocol using a double ratchet that provides post-quantum security guarantees by integrating quantum-secure KEMs. Our work shows that symbolic security protocol model checkers, in particular Tamarin, can verify substantial, real-world

protocols with nested loops. Our work also shows, however, that using trace induction in tools such as Tamarin requires user intervention and insight at many points throughout the proof.

Afterwards, in Chapter 8, we present a novel, cyclic induction mechanism that substantially improves proof automation, thus simplifying reasoning about looping protocols. We evaluate cyclic induction by comparing it with trace induction. In particular, we show that message secrecy can be proven for Signal without requiring abstractions or auxiliary lemmas. Our results show that cyclic induction together with only minimal (and generic) changes to Tamarin's existing proof search strategies can prove many lemmas with no or substantially fewer auxiliary lemmas than needed with trace induction.

6.1 TRACE INDUCTION IN TAMARIN

Before we proceed with the main chapters, we introduce Tamarin's trace induction mechanism. Trace induction was introduced in [111], which established that a property φ is valid if and only if the following, trace-inductive formula is valid:

$$\mathrm{BC}(\varphi) \wedge (\mathrm{IH}(\varphi) \implies \varphi).$$

The first conjunct, $\mathrm{BC}(\varphi)$, is the base case, and it requires proving φ on the empty trace. The second conjunct, $\mathrm{IH}(\varphi) \implies \varphi$, is the induction step, which requires proving φ on the last element of the trace, where φ is assumed on all previous steps of the trace. $\mathrm{BC}(\varphi)$ and $\mathrm{IH}(\varphi)$ are defined as follows.

$$\mathrm{BC}(\varphi) := \begin{cases} \bot & \text{if } \varphi = f@i \\ \neg\mathrm{BC}(\varphi_1) & \text{if } \varphi = \neg\varphi_1 \\ \mathrm{BC}(\varphi_1) \wedge \mathrm{BC}(\varphi_2) & \text{if } \varphi = \varphi_1 \wedge \varphi_2 \\ \exists x.\mathrm{BC}(\varphi_1) & \text{if } \varphi = \exists x.\varphi_1 \\ \varphi & \text{if } \varphi \text{ is atomic and not } f@i \end{cases}$$

$$\mathrm{IH}(\varphi) := \begin{cases} \neg\mathrm{IH}(\varphi_1) & \text{if } \varphi = \neg\varphi_1 \\ \mathrm{IH}(\varphi_1) \wedge \mathrm{IH}(\varphi_2) & \text{if } \varphi = \varphi_1 \wedge \varphi_2 \\ \exists i.\mathrm{IH}(\varphi_1) \wedge \neg\mathsf{last}(i) & \text{if } \varphi = \exists i{:}tmp.\varphi_1 \\ \exists x.\mathrm{IH}(\varphi_1) & \text{if } \varphi = \exists x{:}s.\varphi_1, s \leq msg \\ \varphi & \text{if } \varphi \text{ is an atomic formula} \end{cases}$$

last, as used in the definition of $\mathrm{IH}(\varphi)$, is a special predicate that is true if and only if it is provided the last time point as argument.

After translating φ into its inductive form, Tamarin attempts to prove it as any other formula. Effectively, it attempts to prove the

base case and induction step separately. The branch proving the base case typically leads to a contradiction immediately. As action facts are replaced with the contradiction symbol $\bot$, syntactic reasoning most often suffices to derive a contradiction. In the branch proving the induction step, the induction hypothesis IH becomes available like an auxiliary lemma.

Example 5 (Loop theory)**.** Consider the following model, which consists of three rules, `Start`, `Loop`, and `Stop`. These rules model a loop that starts with the generation of a fresh nonce, is executed a non-deterministic number of times, and may eventually stop.

```
rule Start:
  [ Fr(x) ]
  --[ Start(x) ]->
  [ A(x) ]

rule Loop:
  [ A(x) ]
  --[ Loop(x) ]->
  [ A(x) ]

rule Stop:
  [ A(x) ]
  --[ Stop(x) ]->
  []
```

A simple, obviously true property of the above example is: Every loop that stops was started earlier. We can formalize this property as follows:

$$\varphi_0 = \forall j, x.\, \mathsf{Stop}(x)@j \implies \exists i.\, \mathsf{Start}(x)@i \wedge i < j.$$

This property, however, cannot be proven using trace induction. In general, one can only prove properties of loops by induction when the loops are expressed in terms of facts that appear repeatedly in the protocol's trace. This is not the case for Stop which only occurs once for every loop, namely when it ends. In the above example, the induction hypothesis will be:

$$\mathrm{IH}(\varphi_0) = \forall j, x.\, \mathsf{Stop}(x)@j \implies \mathsf{last}(j) \vee (\exists i.\mathsf{Start}(x)@i \wedge \neg\mathsf{last}(i) \wedge i < j)$$

$\mathrm{IH}(\varphi_0)$ is effectively vacuous as long as the fact $\mathsf{Stop}(x)$ only occurs at the last time point, which Tamarin must consider as a case when attempting to prove φ_0. Tamarin will start proof attempts for φ_0 by adding $\mathsf{Stop}(x)@j$ to the constraint system, which in turn will introduce a rule instance of Stop. As there are no other rule instances occurring later than Stop, Tamarin must consider the possibility that this rule instance occurs at the last time point. In that case, the first disjunct

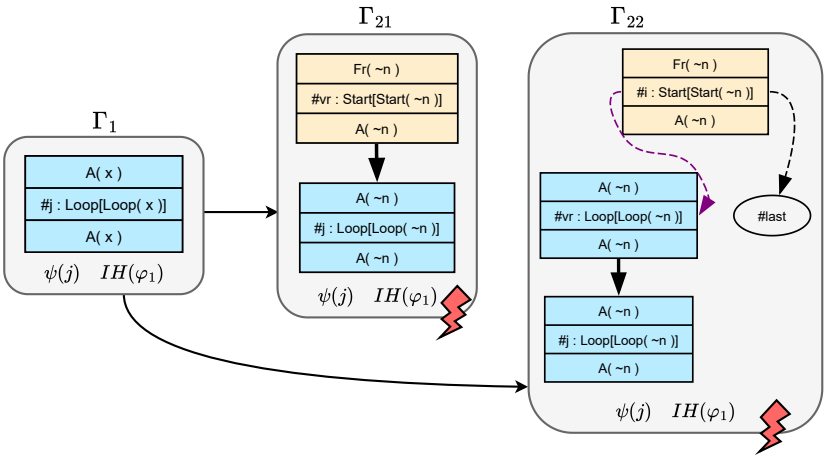


Figure 6.1: Derivation tree for proving φ_1 in the Tamarin theory of Example 5 using trace induction.

on the right-hand side of the implication will apply, whereas one usually requires the second disjunct to apply to make progress on a proof. Only when we can introduce a new Stop fact in the trace that does not occur at the last time point can we use the induction hypothesis.

We can, however, prove a very similar property that expresses the following invariant of loops: For every loop step, the loop must have started earlier. We formalize this invariant as follows:

$$\varphi_1 = \forall j, x.\, \mathsf{Loop}(x)@j \implies \exists i.\, \mathsf{Start}(x)@i \wedge i < j.$$

One can prove φ_1 using trace induction as loop steps are preceded by other loop steps. Also, when using φ_1 as an auxiliary lemma, φ_0 can be proven straightforward and without using trace induction.

When proving φ_1 using trace induction, the branch proving the induction step, i.e., $\mathrm{IH}(\varphi_1) \implies \varphi_1$, results in the derivation tree illustrated in Figure 6.1. The tree's root constraint system is Γ_1, which contains a node constraint and the two formula constraints $\psi(j)$ and $\mathrm{IH}(\varphi_1)$. $\psi(t)$ is what remains from the negated formula φ_1 after instantiating the Loop fact at timepoint t (as in Γ_1).

$$\psi(t) = \forall i.\mathsf{Start}(x)@i \implies \neg(i < t)$$

$IH(\varphi_1)$ is the induction hypothesis:

$$IH(\varphi_1) = \forall j, x.\, \mathsf{Loop}(x)@j \implies \mathsf{last}(j) \vee (\exists i.\mathsf{Start}(x)@i \wedge \neg\mathsf{last}(i) \wedge i < j)$$

To continue the proof search, we solve the open A premise of the node at timepoint j. As the loop step could have been preceded by either the start of the loop or another loop step, this leads to two cases Γ_{21} and Γ_{22}. The presence of the Start node in Γ_{21} immediately contradicts the formula $\psi(j)$. In the case Γ_{22}, the induction hypothesis

now becomes useful as there is a Loop fact at a time point vr that cannot be the last. Tamarin detects this and applies the induction hypothesis automatically, which introduces the node Start at time point i. This again contradicts $\psi(j)$ as $i \prec_{\Gamma_{22}} vr \prec_{\Gamma_{22}} j$ and concludes the proof.

The above examples show that induction can only be applied to formulas that express invariants of loops, but not to formulas that express something that holds after a loop has stopped. For this reason, induction must be applied with care and cannot be blindly applied to prove arbitrary properties of protocols with loops.

7

A FORMAL ANALYSIS OF APPLE'S IMESSAGE PQ3 PROTOCOL

7.1 INTRODUCTION

In this chapter, we show how symbolic security protocol model checkers, in particular Tamarin, can verify substantial, real-world protocols with nested loops in their full complexity by using trace induction as introduced in Section 6.1. We do so by presenting the formal analysis of Apple's advanced, widely deployed iMessage PQ3 Messaging Protocol, or PQ3 for short. PQ3 is used across all of Apple's devices for device-to-device messaging and underlies many other Apple services, e.g., iMessage, FaceTime, HomeKit, and HomePod hand-off. PQ3 is designed to be performant and offers strong guarantees against powerful adversaries, including those who may possess quantum computers in the future.

PQ3 employs a double ratchet algorithm similar to Signal [121]. The protocol takes a hybrid approach to security and combines classical cryptographic primitives, like ECDH, and post-quantum primitives, namely ML-KEM [117], a module-lattice-based KEM. The hybrid construction means that PQ3's security does not solely depend on the security of post-quantum primitives, which are less well understood than their classic counterparts. Moreover, PQ3's integration of hybrid cryptography into the double ratchet provides stronger guarantees than Signal, where a post-quantum KEM is just integrated into the protocol's setup phase, but not into its ratcheting.

We report on our Tamarin model of PQ3, the adversary assumptions, and the protocol's desired properties. We use Tamarin's specification language to specify the messaging protocol and its use of classical and post-quantum cryptography. We also specify all forms of adversary compromise, including the event in which the attacker obtains a quantum computer sufficiently powerful to break all non-post-quantum-secure cryptographic primitives. Essentially, the adversary can compromise any key at any time, either through dedicated key-reveal rules or because they obtained a quantum computer. Using Tamarin's property language, we formalize and prove both secrecy and authenticity theorems. These theorems precisely express the protocol's security guarantees capturing fine-grained notions of key compromise.

Our analysis establishes that PQ3 provides strong security guarantees against an active network adversary that can compromise any secret key, unless explicitly stated otherwise. For example, PQ3 provides forward secrecy, post-compromise security, and post-quantum

security with respect to a "harvest now, decrypt later" adversary. In contrast to Signal, PQ3 provides post-compromise security also against active classical and "harvest now, decrypt later" adversaries and not only against passive, classical adversaries. Moreover, the fine-grained analysis of compromise possibilities and their effects is useful for guiding secure implementations of PQ3. For example, the compromise of a participant's long-term identity key impacts all security guarantees and thus should be stored with extra care, for example, in a device's secure enclave.

CONTRIBUTIONS Our first contribution is the formalization and machine-checked verification of PQ3 to prove all our security claims. Namely, we use Tamarin to prove that PQ3 offers strong security guarantees against a powerful adversary with quantum computing capabilities. These guarantees are fine-grained and comprehensive in that omitting any of the many adversary compromise cases leads to attacks. Our verification thereby provides a formal, machine-checked proof that PQ3 meets the high expectations for a modern device-to-device messaging protocol. This high assurance is important given the prominent role of this protocol, which is used in billions of devices worldwide, and its limited prior analysis.

Our second contribution is to show that symbolic model checkers such as Tamarin can be used to analyze complex looping protocols like iMessage PQ3 or Signal without introducing artificial restrictions. It was previously argued that protocols using double ratchets are out-of-scope for modern, symbolic model checkers such as Tamarin, as they cannot handle the protocol's nested loop and a complex control structure [19]. Our work shows that this is not the case.

OUTLINE In Sections 7.2 and 7.3 we describe PQ3's threat model, requirements, and the protocol itself. In Section 7.4 we present and discuss our Tamarin model of PQ3, the adversary, the protocol's properties, and details on our proofs. We discuss related work in Section 7.5 and draw conclusions in Section 7.6.

7.2 REQUIREMENTS AND THREAT MODEL

7.2.1 *Security Requirements*

SECRECY PQ3 was designed to provide strong secrecy guarantees, namely *message secrecy*, *forward secrecy*, and *post-compromise security*. Message secrecy means that as long as neither participants' session states are revealed, the adversary cannot learn any of their exchanged messages. Forward secrecy and post-compromise security limit the window in which an adversary can learn exchanged messages after they compromise certain parts of the session state.

More concretely, forward secrecy limits the effect of compromise in the past. Should the adversary compromise some state now, they cannot learn previously sent messages. Post-compromise security limits the effect of compromise in the future. Should the adversary compromise some state now, they cannot learn messages sent after the compromise. Noteworthy, forward secrecy and post-compromise security are usually not granted immediately. Typically, the adversary will be able to access some but not all messages sent before or after a compromise, and the precise guarantees depend on the state considered to be compromised. Similarly, protocols typically provide forward secrecy and post-compromise security only with respect to certain adversary assumptions. Signal, for example, provides post-compromise security in its outer ratchet only against passive adversaries.

There exists no standard, formal definition of forward secrecy and post-compromise security. [49] present a formal notion of post-compromise security, but they focus on post-compromise security with respect to long-term key compromise and do not consider other types of key compromise. In our security analysis, we define a secrecy lemma that captures all three notions of secrecy and that addresses the precise implications of partial session state compromise. Describing this fine-grained secrecy lemma requires a detailed understanding of the key material used in PQ3, and is thus deferred to Section 7.4.2.

AUTHENTICATION AND REPLAY PROTECTION A message recipient can identify the message's sender. We formulate this as an *agreement property* as introduced in Section 3.1: the recipient and sender agree on their view of the message. For any message received, the peer must have actually sent the, intending it to go to the receiver, and both peers agree on their view of the message. Moreover, this agreement is *injective*. Namely, a given message is only accepted once by the recipient; hence the protocol provides *replay protection*. Again, we defer to Section 7.4.2 for a precise definition of our authentication guarantees.

7.2.2 *Threat Model*

PQ3 seeks to provide the above security properties even when the protocol is run in the presence of a strong active network adversary who may have access to a powerful quantum computer in the future. As an active network adversary, the adversary can read, reorder, intercept, replay, and send any message to any participant. We assume though that devices use strong randomness and that, short of possessing a quantum computer, the adversary cannot factor large numbers or compute discrete logs. Hence, in the pre-quantum era, cryptographic primitives like ECDH are secure against the adversary.

By default, the adversary can access every participants' key material unless we explicitly forbid this. We will refine our threat model

for each security property and list all the keys that the adversary must not access for the security property to hold. This allows us to focus on which key material the adversary must access to violate a security guarantee and to abstract from whether this compromise is plausible. For example, recently developed cryptographic primitives, designed to provide post-quantum security, may turn out to be flawed. Some keys are stored in a devices' main memory and relatively easy to compromise, whereas others, like identity keys, are stored in Apple's Secure Enclave and are thus much harder to compromise. The fact the adversary can access every key by default allows us to consider all of these cases.

In addition, our threat model accounts for the possibility that the adversary may at some point possess a cryptographically-relevant quantum computer. When this happens, the adversary will be able to break all non-post-quantum-secure primitives, such as ECDH, and can access all such secret key material, independently of what a refined threat model may state.

We constrain the adversary's future quantum computing capabilities by assuming that as soon as the adversary possesses a quantum computer, no honest participant runs the protocol. This models an adversary that anticipates future developments in quantum computing and stores all messages sent by the protocol participants. For this reason, the adversary is a passive quantum attacker and is referred to as a "harvest now, decrypt later" adversary.[1]

For setup and session establishment, the protocol leverages Apple's IDentity Services (IDS) key directory. We assume that this directory is secure in that it only distributes the participants' authentic public keys. The problem of key authentication is orthogonal to PQ3. We discussed it in Part i and Apple recently addressed it for iMessage with their rollout of a key transparency log [1].

7.3 PQ3 MESSAGING PROTOCOL

PQ3 is a device-to-device messaging protocol where either device can asynchronously exchange messages at any time, independent of the connection status of their peer's device. We first describe PQ3 at a high-level of abstraction, followed by a more detailed account. We provide a full pseudocode specification of PQ3 in our artifact (see Section 1.6).

[1] Note that PQ3 only protects past sessions against quantum attackers. To protect active sessions, PQ3's relies on an elliptic curve signature scheme, which can be broken by a quantum computer.

7.3.1 *High-level Account*

In PQ3, communication between two parties, say Alice and Bob, works roughly as follows. Suppose that Alice wants to initiate messaging with Bob.

1. Alice queries Apple's IDS for Bob's *pre-key material* and a *long-term identity public key*.
2. Alice derives an initial *root key*, *chain key*, and *message key*. Alice encrypts her first message for Bob using the message key and sends Bob the ciphertext along with a signature and the key material necessary to derive the initial root key.
3. Upon receiving this new message, Bob lacks the key to decrypt the ciphertext, and so he must derive it. Bob first queries the IDS to verify Alice's long-term identity public key and checks the received signature. He uses the key material received from Alice to derive the initial root, chain, and message key and decrypts the initial message. Alice and Bob have now established a shared session.
4. As long as the session does not change direction (i.e., the current sender keeps sending messages), both parties perform *symmetric ratcheting*. In the symmetric ratchet, participants use the old chain key to derive a new chain and message key.
5. Whenever the session changes direction (i.e., the current receiver wants to reply), both parties perform *public-key ratcheting*. In the public-key ratchet, participants use the old root key and newly sampled asymmetric key material to derive a new root key.

At this high level of abstraction, Steps 2–5 resemble the standard double ratchet. But there are significant differences in the concrete details on how the ratchets are performed, in particular how a post-quantum KEM is integrated into the ratcheting.

7.3.2 *More Detailed Account*

We now expand on the above account. Although this account is more detailed, we still focus on the essential ideas and we omit some low-level details, like message and key derivation tags. Moreover, we describe some additional features of PQ3 at the end of this section.

KEYS Every participant has a *long-term identity key*, a P-256 ECDSA public/private key pair to authenticate messages and other key material. Long-term identity public keys are distributed and authenticated using the IDS. All other keys are used to derive message keys. Figure 7.1 depicts the dependencies between these keys.

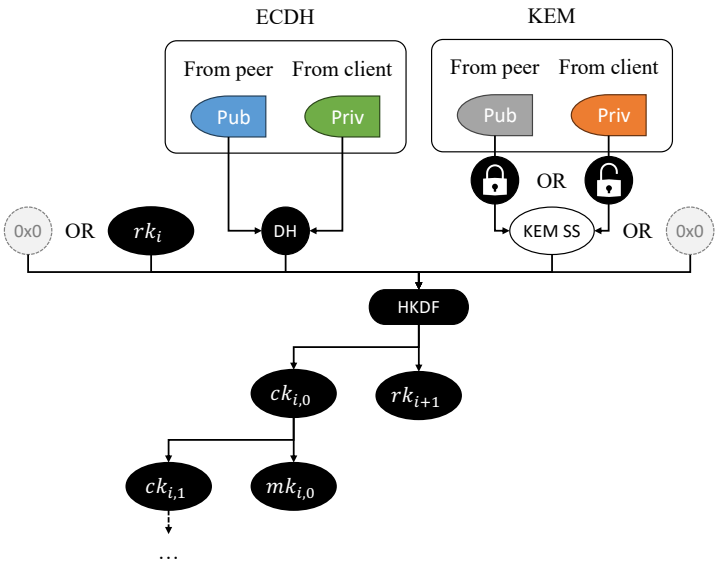


Figure 7.1: Dependency between the keys used by PQ3. Arrows denote that one value is used to derive another. The lock icons denote KEM encapsulation or decapsulation respectively. Sometimes a zero-byte sequence is used instead of a root key or KEM shared secret.

We start by introducing PQ3's three types of symmetric keys. These symmetric keys are derived with respect to a given public-key ratchet step (identified by i in Figure 7.1). *Message keys* (depicted as $mk_{i,0}$) are the message encryption keys and are derived from *chain keys* (depicted as $ck_{i,0/1}$). Chain keys are derived from either previous chain keys or initially from the same entropy sources as the root keys. *Root keys* (depicted as $rk_{i/i+1}$) are used in every public-key ratchet step and, in particular, maintain the entropy from previous public-key ratchets.

Root and initial chain keys are derived from three entropy sources: the session's previous root key (or a zero-byte sequence upon session start; rk_i in Figure 7.1), an ECDH shared secret ("DH" in Figure 7.1), and optionally a KEM shared secret (replaced with a zero-byte sequence when omitted; "KEM SS" in Figure 7.1). To establish these shared secrets, every client uses P-256 ECDH public/private key pairs, which we call *ECDH keys*, and ML-KEM 768 or 1024 public/private key pairs, which we call *KEM keys*. Clients establish the ECDH shared secret by combining an ECDH public key from their peer with their own ECDH private key ("ECDH Pub/Priv" in Figure 7.1). Clients establish the KEM shared secret either by encapsulating it for their peer using their peer's KEM public key or by having their peer encapsulate it for them and decapsulating it with their own KEM private key ("KEM Pub/Priv" in Figure 7.1).

In general, every client uses distinct, fresh ECDH and KEM keys for every session, the public part of which they send in PQ3 messages to their peer. These session-specific keys are called *ephemeral keys*.

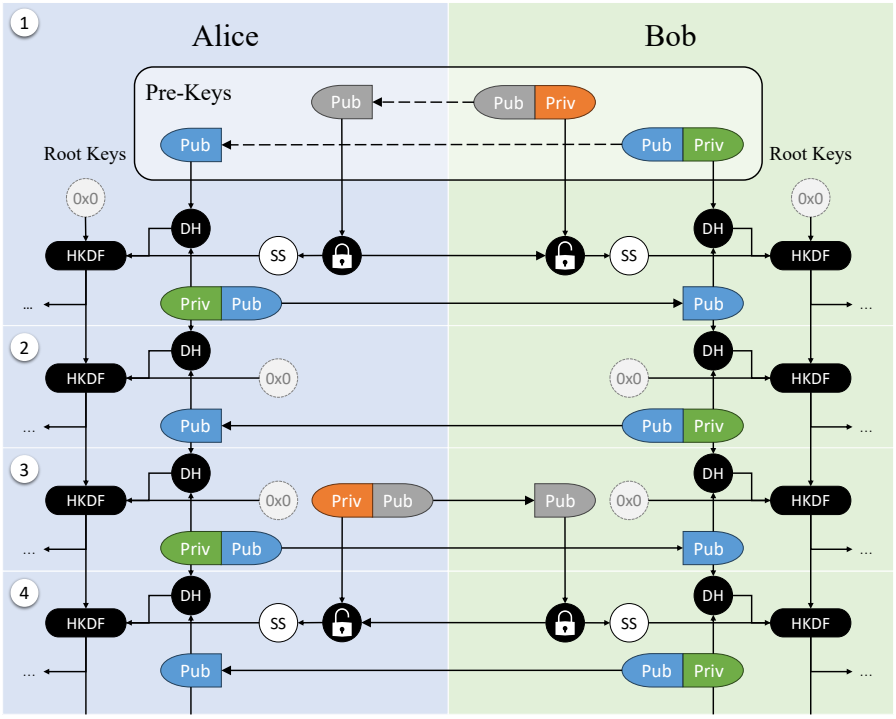


Figure 7.2: PQ3's public-key ratchet. Each block 1-4 illustrates a public-key ratchet step. We omit the symmetric ratchet; chain and message keys are derived from the output of the HKDF (denoted by "..."). In Step 1, Alice initiates a session with Bob and uses pre-key material (white box) to derive a root key. Alice sends a freshly encapsulated shared KEM secret (lock icon), and a freshly sampled ECDH public key to Bob that Bob can use to derive session keys. New KEM shared secrets are only encapsulated and shared when a new KEM public key was sent in the previous public-key ratchet (see block 4). Orange/gray key pairs denote ML-KEM keys, green/blue key pairs denote ECDH keys. This figure was inspired by [121].

Ephemeral keys are short-lived and used only for a specific session. To support asynchronous messaging, clients use ECDH and KEM public *pre-keys* instead of their ephemeral counterparts upon session start (the ECDH and KEM keys depicted in Figure 7.1 could be either ephemeral or pre-keys). Clients upload their pre-keys to the IDS using timestamped pre-key bundles, which are signed with their long-term identity key. Clients can fetch their peers' pre-keys from the IDS to start a new session with any of their peers' clients without requiring that client to be online. Pre-keys can be reused in multiple sessions, but are only used upon session start. PQ3 uses ML-KEM 768 key pairs for ephemeral KEM keys and ML-KEM 1024 key pairs for KEM pre-keys.

SESSION ESTABLISHMENT In the following, we assume, as before, that Alice wishes to establish a new session with Bob. We depict an example run of PQ3 in Figure 7.2, specifically showing the key derivations of both parties. The figure shows four public-key ratchet steps (numbered 1-4). Step 1 illustrates session establishment as explained next. Note that all messages sent between parties include a signature by the respective sender for authentication purposes using their long-term identity key. We omit signatures, long-term identity keys, the steps of the symmetric ratchet, and sent messages from Figure 7.2 to avoid clutter and to focus on the key material used in root key derivation.

Alice's actions are depicted in the left, blue half of Figure 7.2. Alice initiates her session with Bob by performing an IDS query for Bob's identity. Alice thereby learns three keys from the query's result: Bob's long-term identity public key, an ECDH public pre-key, and a KEM public pre-key. Querying and using pre-keys is depicted within the white box in Figure 7.2. Alice then generates a fresh ECDH ephemeral public/private key pair ("Priv/Pub" in Step 1) and encapsulates a fresh KEM shared secret with Bob's public pre-key (lock icon in Step 1). The encapsulation algorithm provides Alice with the cleartext KEM shared secret for her use (shown as "SS" in Step 1), and ciphertext to be given to Bob (the lock to the right of "SS", showing that it used the KEM public pre-key from above). Bob can decapsulate the KEM shared secret with his KEM private pre-key to receive the same KEM shared secret. Alice then combines her ECDH ephemeral private key with Bob's ECDH public pre-key to obtain the initial ECDH shared secret (depicted as "DH").

Alice proceeds to derive the initial root key and the associated initial chain key from the ECDH shared secret, the KEM shared secret, and a zero-byte sequence, which stands in for the previous root key. This is depicted on the far left of Figure 7.2 as "HKDF" in Step 1. She derives a message key from the initial chain key and encrypts her initial message with that message key. She sends Bob the ciphertext, her ECDH ephemeral public key, the KEM encapsulation (with the

latter two shown in Figure 7.2), a hash of Bob's public pre-keys (the *pre-key hash*), and a signature on all these elements and some additional authenticated data. The exact values of the authenticated data field are unspecified, and the field can be used freely by applications.

Bob uses that message to derive the initial root and chain key. Bob's actions are depicted in the right, green half of Figure 7.2. Bob first performs an IDS query to receive Alice's long-term identity public key (not depicted in Figure 7.2), which he uses to verify the message signature. Bob then looks up the private parts of his pre-keys used by Alice, which are identified by the pre-key hash. Bob decapsulates the KEM encapsulation to obtain the KEM shared secret (the open lock symbol in Step 1), and combines Alice's ECDH public ephemeral key with his ECDH private pre-key to establish the ECDH shared secret ("DH" in Step 1). With these two values (and the zero-byte sequence), Bob computes the initial root and chain key (illustrated by "HKDF" in Step 1) and derives a message key from that chain key to decrypt the ciphertext.

SYMMETRIC RATCHET With a shared root key established, Alice can send any number of additional messages to Bob without the participants updating the root key. Nevertheless, each of these messages will be encrypted with a distinct key derived by symmetric ratcheting. Whenever a participant encrypts a message, they use the current chain key to derive a message key, and then ratchet the chain key forward by deriving a new chain key from the previous one. PQ3 establishes per-message forward secrecy as soon as the previous chain and message keys are deleted, i.e., participants should only store the latest root and chain key. The symmetric ratchet, though, is only executed as long as the conversation's direction does not change, i.e., as long as the current sender keeps sending. Whenever the current receiver wishes to respond, they perform a public-key ratchet instead.

PUBLIC-KEY RATCHET Suppose, after receiving some messages from Alice, that Bob wants to reply. This means that the conversation *changes direction*, and whenever this happens clients perform the public-key ratchet. Every public-key ratchet updates the root key and derives a new, initial chain key. The steps taken to derive these new keys are similar to the steps taken during session establishment. Figure 7.2 illustrates (next to session establishment) three further public-key ratchet steps (numbered 2-4).

To perform the public-key ratchet, Bob first generates a fresh ECDH ephemeral public/private key pair. Depending on the conversation's state, Bob may additionally perform either of the following two actions: (i) use the encapsulation algorithm to produce a new KEM shared secret and ciphertext (for decapsulation by Alice), or (ii) generate a new KEM ephemeral public/private key pair. Action (i) is performed when-

ever Bob's peer, Alice, performed Action (ii) in the previous public-key ratchet. To save bandwidth, Action (ii) need not always be performed. Instead, a custom heuristic determines when a client refreshes its KEM keys. The heuristic accounts for the threat environment, performance, and other requirements. As per iOS 17.4, PQ3 clients send a fresh KEM public key roughly every 50 messages or whenever they have not sent a fresh KEM public key within a week [81].

Bob then derives the next root key and the associated initial chain key. He first combines his freshly generated ECDH ephemeral private key with Alice's ECDH ephemeral public key to obtain the new ECDH shared secret. He then uses the previous root key, the new ECDH shared secret, and either the new KEM shared secret or a zero-byte sequence (depending on whether Bob performed Action (i)) to derive the next root key and associated initial chain key. He again derives a message key from that chain key to encrypt his message and sends Alice the following values: the ciphertext, his fresh ECDH ephemeral public key, optionally the new KEM encapsulation (Action (i)), optionally his new KEM public key (Action (ii)), and a signature on all the above.

Figure 7.2 depicts in Step 3 that Alice generates a new ephemeral KEM public/private key pair and sends the corresponding public key to Bob, i.e., Alice executes Action (ii) above. This means that Bob will execute Action (i) in Step 4.

Overall, the cryptographic constructions used are hybrid: all key derivations incorporating a KEM shared secret also involve classical secrets. This design entails (and we establish this formally in our proofs) that PQ3's security is at least as strong as when using classical cryptography alone. The repeated use of the KEM encapsulation in the protocol therefore strictly strengthens the protocol to provide post-compromise security even against a "harvest now, decrypt later" adversary who managed to access some KEM shared secret.

7.4 SECURITY PROOFS

In this section, we describe how we modeled PQ3 and formalized its security goals using Tamarin. We describe our protocol model (Section 7.4.1), the formal security properties (Section 7.4.2), and our proofs (Section 7.4.3). Our protocol model covers PQ3 in its full complexity, including its nested loops, all its cryptographic primitives, and their combinations. We discuss limitations and proof effort in Section 7.4.4.

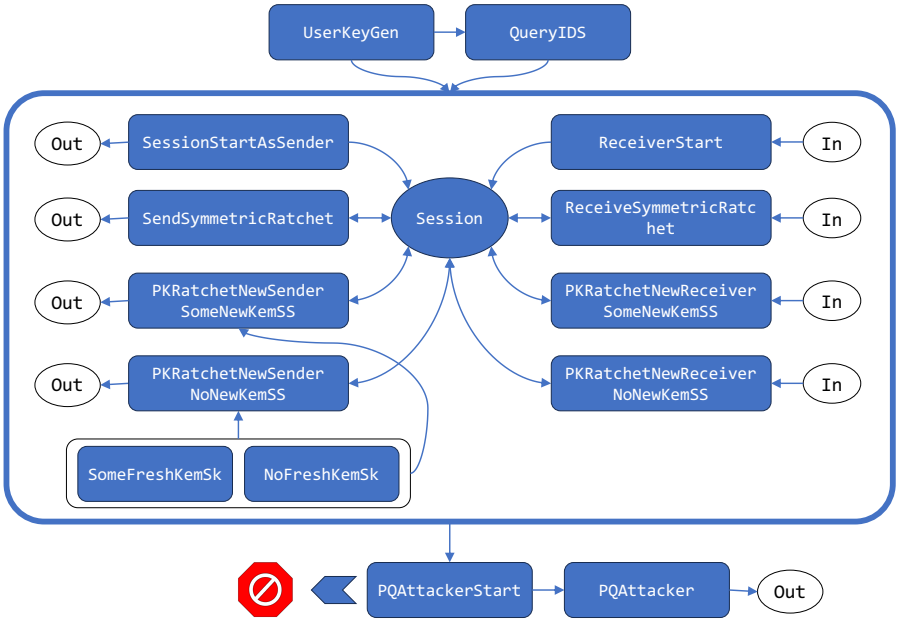


Figure 7.3: Overview of our formal model. Rectangles denote rules and ellipses denote facts, with their respective name printed inside. Arrows denote fact consumption and generation or rule transition. The white rectangle around `Some/NoFreshKemSk` denotes that either of the rules is applied non-deterministically. The rule `PQAttackerStart` can be applied at any point. When this happens, protocol execution halts (modeling a "harvest now, decrypt later" adversary) and thereafter the rule `PQAttacker` can be applied, which reveals any non-post-quantum-secure secret to the adversary. This figure omits rules that reveal key material.

7.4.1 *Protocol Model*

Multiset-Rewriting Rules

We used Tamarin to comprehensively model PQ3 as described in the previous section. We provide an overview of our model's state transition rules in Figure 7.3. Our formal model has three parts. The first part models the generation of long-term signing keys and pre-keys (rule `UserKeyGen`), and IDS queries (rule `QueryIDS`). These are setup rules, which are the same for all participants, independent of whether they start a session as the sender or receiver. The second and third part model the adversary's capabilities and PQ3's protocol flow respectively.

In our model of the adversary's capabilities, we allow the adversary to compromise every private, root, chain, and message key through dedicated reveal rules, unless our security lemmas explicitly forbid a certain key to be revealed. Additionally, we model the "harvest now, decrypt later" capability as follows. Whenever participants generate a non-post-quantum-secure key, like a fresh ephemeral ECDH private key, our model saves the key in a persistent fact (i.e., a fact that is not consumed when it is used in a rule's premise). The adversary can then access any secrets stored this way after the rule `PQAttackerStart` is applied, but from that point on, no honest participant runs PQ3.

Our model of PQ3's protocol flow is depicted as the big blue box in Figure 7.3. The left-hand side depicts all sender-related rules, the right-hand side all receiver-related rules, and in the center is a `Session` fact that stores all information needed to send and receive messages. For example, a `Session` fact stores a participant's most recently generated ECDH and KEM private keys and the corresponding public keys of their peer, as well as any derived root and chain keys.

A new session is started by applying the rule `ReceiverStart` or `SessionStartAsSender`. These are the only two rules that only produce and do not consume a `Session` fact. Most other rules update a session, i.e., they consume and produce a `Session` fact, and they can be applied arbitrarily many times per session. After a new session has started, one of two things can happen. Either the conversation does not change direction and then both participants will apply the symmetric ratchet rules, or the conversation changes direction and the public-key ratchet rules are applied.

When being the receiver, a participant may non-deterministically choose to become sender. When they do, they perform the public-key ratchet. Depending on whether their peer had sent them a new KEM public key previously, they may additionally encapsulate a new KEM shared secret. Also, the new sender may non-deterministically send a new KEM public key themselves to their peer.

A participant changes from the sender to the receiver role when they receive a new message while being in the sender state. When a

```
functions: hkdf/2, suffix/1, prefix/1, concat/2, h/1

equations: concat(prefix(x), suffix(x)) = x

functions: pqpk/1, encap/2, decap/2
equations: decap(encap(k, pqpk(sk)), sk) = k

functions: default/2, Just/1, None/0, unjust/1
equations: default(Just(v), t) = v,
           default(None, v) = v,
           unjust(Just(t)) = t
```

Figure 7.4: Custom functions and equations defined in our formal model.

participant becomes the receiver, they perform the public-key ratchet as well. In one of the two rules, they do so using a decapsulated KEM shared secret, and in the other rule they use a zero-byte sequence instead.

Intuitively, one can consider our model as implementing two nested loops. First, there is the outer, public-key ratchet loop where participants generate new ephemeral ECDH and KEM secret keys and derive root and chain keys. Second, there is the inner, symmetric ratchet loop where participants derive message keys and send messages. The symmetric ratchet loop always runs within one iteration of the public-key ratchet loop.

Equational Theory

Our model uses Tamarin's built-in equational theories for signing, symmetric encryption, and Diffie-Hellman key exchange. These respectively model digital signatures, symmetric encryption under message keys, and ECDH key exchanges. We additionally use Tamarin's natural numbers theory to model message counters.

In addition to these built-in theories, we specify some custom functions and equations, shown in Figure 7.4. First, we specify the functions `hkdf`, `suffix`, and `prefix` for key derivation. The function `hkdf` models an HMAC-based key derivation function (HKDF) [90] and takes two arguments: the first is the source of entropy and the second is a domain-separating tag or salt. The `prefix` and `suffix` functions are used for chain and root key derivations, which are derived by splitting a bit-string into a prefix and suffix of equal length. The function `concat` allows one to recover a value given its prefix and suffix. We do not need to use `concat` in the rules modeling the protocol roles of regular parties in our model, but the adversary can use it to reconstruct a value from the prefix and suffix. Additionally, we specify the unary function `h` to model the pre-key hash used during session establishment, see Section 7.3.

The functions `pqpk`, `encap`, and `decap` model KEM encapsulation and follow the standard symbolic model for asymmetric encryption. Finally, we use the wrapper function `Just` and the constant `None` to model optional values. The function `default` (together with the accompanying equations) unpacks an optional value or replaces it with a default. For example, we use `Just` and `None` to wrap values that are only sent optionally, e.g., the pre-key hash. The function `unjust` allows the adversary to access the contents of any `Just` value they intercept.

7.4.2 *Properties Specified and Proven*

Secrecy

PQ3 aims to satisfy three secrecy properties: message secrecy, forward secrecy, and post-compromise security. We formalize all three properties as a single lemma, which we depict in Figure 7.5.[2] The lemma states that the adversary cannot know a message (line 5) that has been previously sent (line 3), unless the adversary succeeds in at least one of four kinds of compromise, listed below. The kinds of compromise are formulated with respect to the keys referenced by the `Keys` fact. This fact lists all keys and shared secrets used by the sender when sending the respective message, e.g., their most recently sampled ephemeral ECDH public key (`myEcdhPk`) and the most recently encapsulated KEM shared secret (`kemSS`). We sketch a possible attack for each kind of compromise to show that dropping any but the first disjunct yields a counterexample. To learn a message sent with PQ3, the adversary must compromise at least one of:

- The message key used during encryption from either the receiver or sender (lines 6-7 in Figure 7.5). Should the adversary learn the message key, they could simply decrypt the message themself.
- One of the chain keys used in the symmetric ratchet to derive the message key from either the receiver or sender (lines 8-11). Should the adversary learn one of these chain keys, they could simply derive the message key themself.
- The recipient's long-term identity key before the message `msg` was sent (line 12). In this case, the adversary could generate a fresh ECDH ephemeral and KEM encapsulation key and send them to the messaging partner in question. This attack allows the adversary to carry out all communication in their victim's stead.

[2] In the following, we will sometimes shorten the names of facts in lemmas compared to the source files, e.g., `RevealIdentityKey` may become `RevealIDKey`. We also omit some irrelevant variables and replace them with _. These variables are understood to be quantified by the innermost quantifier.

- One of the ephemeral ECDH secret keys, used to derive the most recently established ECDH shared secret, *and* the KEM shared secret (lines 13-24). This allows the adversary to perform a public-key ratchet step themself.

  The adversary can learn an ECDH secret key either through direct compromise (lines 14-16) or using a quantum computing attack should a sufficiently powerful quantum computer be available (line 13). The compromise of the sender's ECDH pre-key has no effect because a sender will always sample a fresh ECDH ephemeral key upon session start.

  The KEM shared secret can be effectively compromised in two ways. First, the adversary can compromise the KEM secret key used for encapsulation (lines 17-20). Second, the adversary can circumvent the need to compromise the KEM shared secret by compromising a root key derived after that KEM shared secret was established (lines 21-24). In the latter case, if the adversary additionally learns an ECDH secret key used in a subsequent public-key ratchet step, they can derive the respective initial chain key themself.

  In addition to the ECDH and KEM shared secret, the adversary also requires the root key from the previous public-key ratchet to perform the current public-key ratchet themself. Our threat model, however, permits this root key to be revealed to the adversary anyway.

Recall that our threat model assumes that the adversary can access all key material unless explicitly forbidden. Our secrecy lemma only forbids the adversary to access key material related to sending the message in question. All key-reveal assumptions in lines 6-24 use the key material introduced in line 4, which in turn is bound to the `Sent` event in line 3 by the variable `#t`. Thus, proving secrecy establishes forward secrecy and post-compromise security as we explain next.

For long-term identity keys, we show that PQ3 provides forward secrecy in that all messages exchanged prior to the compromise of such a key remain secure (see line 12). For most encryption keys (exceptions and details below), we establish forward secrecy and post-compromise security in that to compromise a given message, the adversary must learn the respective key used for that message and the compromise of past or future keys has no effect. Note that "past" and "future" here refer to the points in time when a key was used, not when it was compromised. In particular, this allows us to establish post-compromise security guarantees even *after* the adversary obtained a quantum computer and participants stopped running the protocol. Although participants will no longer rotate keys, as they no longer run the protocol, they will have self-healed from the compromise of any other key than the most recently used one. For an illustration, see Figure 7.6.

```
1  All id me them msg ad myEcdhPk theirEcdhPk kemSS encapPk
2    rk ck mk #t.
3        ( Sent(id,_,me,them,msg,ad) @ #t
4        & Keys(myEcdhPk,theirEcdhPk,kemSS,encapPk,rk,ck,mk) @ #t)
5    ==>   (not Ex #x. K(msg) @ x)
6        | (Ex #x. RevealMessageKey(me,mk) @ #x)
7        | (Ex #x. RevealMessageKey(them,mk) @ #x)
8        | (Ex ckC #x. RevealChainKey(me,ckC) @ #x
9            & (ckC << ck | ckC = ck))
10       | (Ex ckC #x. RevealChainKey(them,ckC) @ #x
11           & (ckC << ck | ckC = ck))
12       | (Ex #x. RevealIDKey(them) @ #x & #x < #t)
13       | ( ( (Ex #x. PQAttack() @ #x)
14           | (Ex #x. RevealECDHPreKey(them,theirEcdhPk) @ #x)
15           | (Ex #x. RevealECDHKey(id,me,myEcdhPk) @ #x)
16           | (Ex #x. RevealECDHKey(_,them,theirEcdhPk) @ #x))
17         & ( (Ex #x. RevealKemKey(me,encapPk) @ #x)
18           | (Ex #x. RevealKemKey(them,encapPk) @ #x)
19           | (Ex #x. RevealKemPreKey(me,encapPk) @ #x)
20           | (Ex #x. RevealKemPreKey(them,encapPk) @ #x)
21           | (Ex k #x. RevealRootKey(me,kemSS,k) @ #x
22               & k << rk)
23           | (Ex k #x. RevealRootKey(them,kemSS,k) @ #x
24               & k << rk)))
```

Figure 7.5: Secrecy lemma. The lemma formalizes that if a message `msg` was sent using the keys referenced by the `Keys` fact, then either the message cannot be known by the adversary (line 5) or the adversary compromised a specific combination of keys (lines 6ff.). Section 7.4.2 explains this lemma, line-by-line, in further details.

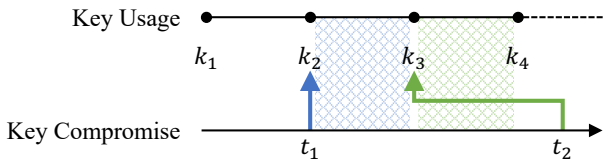


Figure 7.6: A participant derives four initial chain keys (k_1-k_4) over time and the adversary compromises k_2 and k_3 at times t_1 and t_2 respectively. Independent of when it occurs (compare t_1 with t_2), key compromise has a similar and limited effect: The adversary can only learn messages sent before the next initial chain key is derived (the shaded areas).

```
All id ecdhKey1 ecdhKey2 m #t1 #t2 #t3.
      ( Keys(ecdhKey1, ...) @ #t1
      & SessionInfo(id, ...) @ #t1
      & K(ecdhKey1) @ #t2
      & Keys(ecdhKey2, ...) @ #t3
      & Sent(id, ..., m, ...) @ #t3
      & #t3 < #t1)
  ==>   (not Ex #x. K(m) @ #x)
      | (...) // as in secrecy lemma
```

Figure 7.7: Sketch of a potential formalization of ECDH key forward secrecy. Observe that this property is strictly weaker than our secrecy lemma in Figure 7.5 because we only add conjuncts to the implication's left-hand side. Thus, this formalization of forward secrecy is implied by our secrecy lemma.

To provide an example for why our secrecy lemma entails forward secrecy and post-compromise security, consider the lemma modeling ECDH key forward secrecy in Figure 7.7. The lemma resembles our secrecy lemma in Figure 7.5, but additionally assumes that the adversary learned an ECDH key derived before the current message was sent. This modified lemma accurately models ECDH key forward secrecy, but it is strictly weaker than our secrecy lemma. Formally, this is the case because we only strengthen the implication's left-hand side. We also cannot drop any disjunct on the implication's right side because, if we could, our secrecy lemma would not be provable (we sketched attacks on the previous page). Intuitively speaking, the adversary does not gain more power when we explicitly add the event that they learn a respective ECDH key to the trace because we assume that the adversary can access all keys by default anyway.

In general, we establish forward secrecy and post-compromise security guarantees upon key rotation. For some keys, forward secrecy and post-compromise security are only established under further constraints. In these cases, our secrecy lemma precisely defines the point in time at which forward secrecy or post-compromise security are established. We list all forward secrecy and post-compromise security guarantees entailed by our secrecy lemma below and, wherever necessary, describe the constraints on these guarantees. When the adversary does not possess a quantum computer, PQ3 provides:

- Long-term identity key forward secrecy.
- ECDH ephemeral key forward secrecy and post-compromise security.
- ECDH pre-key post-compromise security as soon as a new ECDH ephemeral key is generated by a session's initial recipient.

In practice, PQ3 also provides forward secrecy for ECDH pre-keys as it requires that participants update their pre-keys registered at the

```
All id i s r m ad #t.
    Received(id, i, s, r, m, ad) @ #t
==> ( (Ex id2 #x. Sent(id2, i, s, r, m, ad) @ #x
        & #x < #t)
    | (Ex #x. RevealIDKey(s) @ #x & #x < #t))
```

Figure 7.8: Agreement lemma. For every message-receive event, there must be a corresponding message-send event for which the participants agree on the authenticated data, sender, receiver, and message counter, unless the sender's long-term identity key was previously compromised.

IDS every 2 weeks. As soon as a client registers a new pre-key, they establish forward secrecy for all previous session-start messages sent to them.

Should the adversary at some point break all non-ML-KEM keys using a quantum computer, PQ3 still provides:

- ML-KEM key post-quantum forward secrecy and post-compromise security.
- Chain and message key forward secrecy and post-compromise security. These properties are established unconditionally except for chain key post-compromise security, which is established upon the next public-key ratchet. PQ3 establishes these properties even when the adversary possesses a quantum-computer because these keys depend on KEM-encapsulated secrets.

Note that working out and rigorously proving such fine-grained notions of secrecy is nontrivial and one strongly benefits here from a proof assistant. Overall, our Tamarin proof of secrecy establishes that, in the absence of the sender or recipient being compromised, all keys and messages transmitted are secret. The secrecy property is fine-grained in that compromises can be tolerated in a well-defined sense where the effect of the compromise on the secrecy of data is limited in time and effect as described above. Moreover, we show that PQ3 combines the security of both classical and post-quantum-secure cryptographic primitives. Hence, to break PQ3 one must break both.

Agreement

In contrast to secrecy, formalizing agreement is much simpler. This is because PQ3 relies on the participants' long-term identity keys' security to provide agreement. Compromise of a participant's long-term identity key is both necessary and sufficient to break agreement. It is necessary because an attacker must generate a message signature when trying to spoof a sender, and it is sufficient because a sender need not compromise the sender's encryption keys to send an inauthentic message; they can simply generate their own and send them alongside the faked message.

```
1  All s1 s2 r1 r2 m ad ecdhPk1 mk1 ecdhPk2 mk2 #t1
2    #t2.
3      ( Received(_,_,s1,r1,m,ad) @ #t1
4      & Keys(ecdhPk1,_,_,_,_,mk1) @ #t1
5      & Received(_,_,s2,r2,m,ad) @ #t2
6      & Keys(ecdhPk2,_,_,_,_,mk2) @ #t2)
7  ==> ( (#t1 = #t2)
8      | ( ecdhPk1 = ecdhPk2 & mk1 = mk2
9        & s1 = s2 & r1 = r2
10       & Ex #x. ECDHPreKeyGen(r1, ecdhPk1) @ #x)
11     | (Ex #x. RevealIDKey(s1) @ #x & x < #t1 )
12     | (Ex #x. RevealIDKey(s2) @ #x & x < #t2))
```

Figure 7.9: Injective agreement lemma. It formalizes that for two message-receive events with the same message `m` and authenticated data `ad`, these events must be the same (line 7), or they were sent using the recipients pre-key (lines 8f.), or one sender's identity key was compromised (lines 11ff.).

Our formalization of agreement is split into two Tamarin lemmas (Figures 7.8 and 7.9). The first lemma formalizes agreement: Whenever a participant `r` receives a message `m` and authenticated data `ad`, apparently from `s` and with message counter `i`, then either `s` had previously sent `m` to `r` with counter `i` or that sender's long-term identity has been compromised in the past.

The second lemma formalizes that the agreement is injective [107], meaning that there is a one-to-one mapping from receive-events to send-events. This lemma states that for every two honest message-receive events with the same message and authenticated data, these events must either be identical (`#t1 = #t2`), or a recipient's ECDH pre-key rather than an ephemeral key was used to derive the message key (lines 8-10), or either of the senders were compromised. Compromise of one sender suffices to violate injective agreement because agreement does not entail secrecy. The adversary could learn a message by compromising the ECDH and KEM keys of the session. They could then send the message again, which requires the compromise of a long-term identity key, however, to produce the necessary signature.

During our proof efforts, we noticed a trivial violation of injective agreement, which is covered by lines 8-10. PQ3 cannot provide injective agreement for session-start messages (and messages sent as part of the symmetric ratchet directly thereafter) as pre-keys can be reused for session starts. Thus, recipients will accept session-start messages multiple times. In practice, this case must be addressed by an application's session-handling layer, which defines under which conditions clients will accept session-start messages from devices they already have an existing session with. We shared this finding with Apple researchers who confirmed that the iMessage session-handling layer indeed addresses this case. Put differently, our formal proofs

highlight precisely the assumptions on session-handling needed to securely deploy PQ3.

7.4.3 *Proofs*

We encountered two challenges when verifying PQ3. First, PQ3 employs a nested loop. If not carefully handled, loops result in prover non-termination as they are unrolled infinitely often. Tamarin provides induction to address this problem, but using induction correctly, especially when loops are nested, requires postulating nontrivial auxiliary lemmas. See Section 6.1 for an introduction to Tamarin's induction mechanism.

Second, our threat model considers the leakage of "synthetic" key material, derived using a KDF, and our lemmas naturally must refer to this key material. When proving secrecy, we repeatedly encountered cases similar to the following. Tamarin would consider an honest session sending a message, claiming that the adversary could get the decryption key for this message (violating secrecy) from a completely unrelated session. We call such unrelated sessions *ghost sessions*. In this case, the non-trivial proof goal was to convince Tamarin that the ghost session must be the same as the honest or the peer's session. Note that other protocol models typically only consider the leakage of "atomic" key material, i.e., key material modeled as a fresh term.

To address these two challenges, we use three kinds of auxiliary lemmas.

LOOP-JUMP LEMMAS These lemmas allow one to skip unrolling the steps of a (nested) loop and jump to a "relevant" point in a loop, for example, its beginning or where a specific term was introduced.

VARIABLE-LINKING LEMMAS These lemmas establish properties of the following kind. Given a fact tag using variables a and b, if two instances of that fact tag have the same value for a, they must have the same value for b.

ADVERSARY-CONSTRUCTION LEMMAS These lemmas formalize how an adversary could learn a term. Typically, the adversary can either construct it or access it using a dedicated reveal rule (which in turn typically implies a contradiction to the threat model). Figure 7.10 depicts our model's adversary-construction lemmas. For example, `CkCompromise` states that the adversary can only know a chain key if they know the value that gets split into the root and chain key (`rkCK`), or they compromised this or a previous chain key.

Loop-jump lemmas are the foundation for proving properties of models including nested loops. Without such lemmas, Tamarin's induction fails to prove even the simplest properties of an outer loop.

Similarly to Example 5 introduced in Section 6.1, the induction hypothesis will not apply in cases where a step in the outer loop is directly preceded by a step in an inner loop. Moreover, adversary-construction lemmas are required to deal with the complicated terms that are computed in nested loops, and variable-linking lemmas are required to address ghost sessions.

We proved secrecy for PQ3 using a series of adversary-construction lemmas, depicted in Figure 7.10, which in turn were proven using the loop-jump and variable-linking lemmas in Figures 7.11 and 7.12. Concretely, when proving secrecy, Tamarin first negates the original lemma and tries to construct a trace satisfying the negated lemma, i.e., Tamarin tries to construct a trace where a message has been sent and the adversary knows it. By solving for how the adversary could learn the message, Tamarin deduces that the adversary must know the message key used for encryption. This allows us to apply the first adversary-construction lemma `MkCompromise`. This lemma expresses that the adversary can only know the message key if they either know the respective chain key (allowing us to apply the next adversary-construction lemma) or if they access a reveal rule (contradicting our threat model assumptions directly). In the case where the adversary knows a respective smaller term, we can apply the next adversary-construction lemma, etc. Finally, the lemmas `ECDHSSCompromise` and `KemSSCompromise` directly contradict the threat model.

We proved these adversary-construction lemma using the loop-jump and variable-linking lemmas depicted in Figures 7.11 and 7.12. A sequence of variable-linking lemmas (depicted in blue) connect message to chain to root keys and to the respective KEM shared secret and ECDH shared secret (Figure 7.11). Loop-jump lemmas (depicted in orange) then connect the shared secrets to the asymmetric key material used to establish them (Figure 7.12). This allows Tamarin to deduce that access to the shared secret requires access to the respective private key material. Beyond the lemmas depicted in Figures 7.11 and 7.12, we only use the three loop-jump lemmas `RootKeyConnectionReceive`, `RootKeyConnectionSend`, and `SessionStart`, which jump from an instance of the symmetric ratchet to the most recent public key ratchet (switching from sender to receiver or receiver to sender respectively) and the session start.

We proved both agreement lemmas much like we proved secrecy, but proving agreement was much simpler. PQ3 provides agreement by signing every message. When trying to prove non-injective agreement, Tamarin immediately finds that to violate agreement, the adversary must generate this signature themself, which in turn requires access to the signing key. The rule that introduces the signing key, however, can directly be established using the `SessionStart` lemma as signing keys are queried only upon session start.

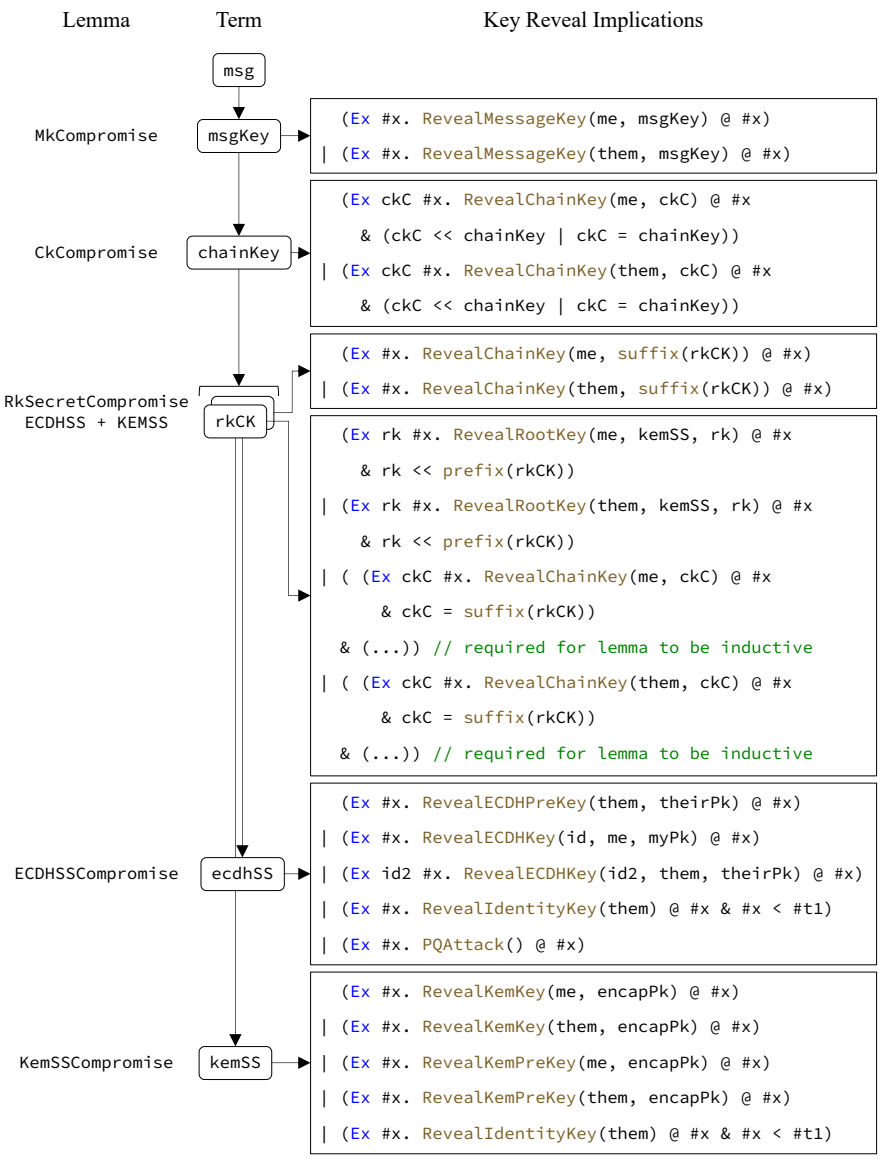


Figure 7.10: Connection between Adversary-Construction Lemmas for Message Secrecy. Arrows denote logical implication. We omit two conjuncts in `RkSecretCompromiseKEMSS` that are only required to prove the lemma by induction. We provide more details on these conjuncts in our formal model (see Section 1.6).

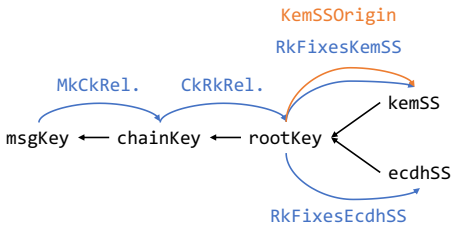


Figure 7.11: Loop-Jump (orange) and Variable-Linking Lemmas (blue) Related to Key Derivation. Black arrows indicate which variables are used to construct other variables, e.g., a message key is derived from a chain key.

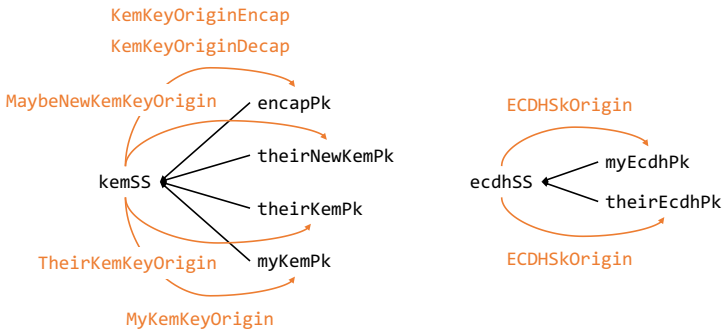


Figure 7.12: Loop-Jump Lemmas (orange) Related to Establishing Shared Secrets. Black arrows indicate which key material can be used to establish which shared secret.

When attempting to prove injective agreement, Tamarin will start by constructing a trace with two honest receive events for the same message. Using variable-linking lemmas, we can establish that these two sessions must use the same ECDH shared secret, and using the respective loop-jump lemmas, we can jump to the rule instantiation where the receiver generated their latest ECDH ephemeral key. This allows Tamarin to derive that the two receive events must have happened in the same session (unless a pre-key was used; but this case is addressed in the lemma directly).

Finally, we only use six auxiliary lemmas not fitting the categories defined above. These lemmas simply limit Tamarin's search space to reduce proof construction time. For example, they show that certain events (like session start) can only occur once per session, or establish well-formedness conditions (for example, that the root key is a subterm of the chain key).

7.4.4 *Discussion*

Scope of Analysis

We do not consider session handling, long-term identity or pre-key rollover, and only consider group messaging implicitly. Our analysis covers the protocol design as described in the documentations we received from Apple. PQ3's implementation is not part of our analysis. Furthermore, as our analysis is based on symbolic models, it abstracts away some details of the concrete implementation, like message lengths and some algorithmic details of the ciphers used.

We did not model session handling as a specification of iMessage's session handling was not available to us. Moreover, PQ3 is not limited in its use to iMessage. Different applications may have different requirements on their session handling. Studying PQ3 in isolation is therefore desirable in its own right.

A security analysis of group aspects, such as members joining or leaving groups, is not part of PQ3 as it is a device-to-device messaging protocol. In practice, group messaging can be implemented using PQ3 by sending messages via pairwise runs of PQ3 to all group members. Such functionality is provided by an application's session-handling layer and is thus outside of our analysis. iMessage implements group messaging using multiple, individual device-to-device sessions, and our analysis establishes the security of each such session.

Beyond the limitations just mentioned, our formal model incorporates all details that were part of the documentation provided to us by Apple. In particular, we did not abstract away any protocol steps that participants may take.

Proof Effort

Our Tamarin model comprises 32 lemmas in total. Next to the auxiliary lemmas used to prove secrecy and agreement (Section 7.4.3), our model includes a *sources lemma*, which aids Tamarin in precomputation steps, and two *executability lemmas*. Executability lemmas effectively "sanity check" a protocol specification by establishing that the participants can run the protocol without adversary involvement. This enhances our confidence that the protocol model faithfully represents the protocol and that its properties do not hold trivially.

All proofs are guided by custom proof heuristics, implemented in Python, and finding the right heuristics to successfully construct proofs required substantial efforts. Checking the proof for the lemma formalizing injective agreement (Section 7.4.2) takes around 7 hours and requires 20 GB of RAM on a server using two Intel Xeon CPU E5-2650 v4 @ 2.20GHz. The proofs of other lemmas require up to 100 GB of RAM to be checked. Overall, we estimate that proving PQ3 took around 2.5 person-months of work, but we also estimate that proving protocols exhibiting challenges similar to iMessage PQ3 would take much less time when following our proof strategy introduced above.

7.5 RELATED WORK

7.5.1 *Formal Analysis of Double Ratchet Protocols in the Symbolic Model*

ProVerif was used to analyze Signal [88] and its post-quantum secure key agreement protocol PQXDH [22], but both analyses are quite limited in scope. The former does not consider Signal's symmetric ratchet and only considers a fixed, finite number of protocol sessions without loops. It was conducted before ProVerif was extended to support a trace induction mechanism [26], which is comparable to Tamarin's trace induction. The latter work analyzes PQXDH in isolation and thus does not consider any of Signal's looping behavior.

[54] analyzes Signal together with its session-handling layer Sesame [109] using Tamarin. The authors show that, when sessions are accounted for, Signal does not achieve post-compromise security despite the double ratchet having this property. To show this, however, they abstract the Signal protocol, over-approximating its security and simplifying its looping structure drastically.

The first work in the symbolic model to consider Signal's looping behavior was conducted in DY* [19]. In the original DY* paper [19], the authors present an analysis of Signal as a case study. They observe: "Notably, Signal has not been mechanically analyzed for an arbitrary number of rounds before. The ProVerif analysis of the Signal protocol in [88] was limited to two messages (three ratcheting rounds), at which point the analysis already took 29 hours. (With CryptoVerif, the analysis of Signal has to be limited to just one ratcheting round)." Their own proof is, however, also limited and only verifies properties for the outermost ratchet.

7.5.2 *Computational Analysis of Double Ratchet Protocols*

In [23], the authors analyze variants of the double ratchet protocol in the Universal Composability (UC) framework. They provide detailed security definitions and consider when keys must be deleted for different properties to hold. An analysis in the UC framework is also given by [39]. In [8, 48], the authors present game-based proofs of security. In particular, [48] presents a formal analysis of Signal in the random oracle model. Their focus is on Signal's key agreement and they reason about loops using induction. [8] carries out game-based proofs for a Signal-like protocol; they provide a rational reconstruction of a generalized protocol that modularly achieves the different kinds of properties one wants from Signal and the use of double ratchets. In all these works, security is shown using pen-and-paper proofs, which are not machine checked, and post-quantum security is not considered.

Concomitantly to our work, Stebila carried out a computational analysis of PQ3 [151], providing a reduction argument for its security. He also formalizes the hybrid cryptography integrated into both PQ3's initialization and double ratchet, and establishes that this provides both forward secrecy and post-compromise security against both classical and "harvest now, decrypt later" adversaries. The modeling of cryptography is, as is standard for computational formalizations, more detailed than in our approach. In contrast, the security model, and the proofs (which are game-based, focused on deriving a bound on the adversary's advantages) are considerably more complex, and proofs are pen-and-paper based, rather than machine checked. Moreover, [151] did not consider replay in its analysis, and during our Tamarin proofs, we uncovered that injective agreement can only be provided under

additional assumptions (not present in [151]) on the session-handling layer.

7.6 CONCLUSION

We have used Tamarin to formally verify the device-to-device messaging protocol PQ3. Our analysis is based on machine-checked proofs of fine-grained secrecy and authentication properties. This provides a high degree of assurance that PQ3 functions securely against an active network adversary who can selectively compromise parties, even when sufficiently powerful quantum computers become available. Additionally, the proven properties give a detailed account of the impact that the compromise of every individual key has. Lastly, we show that Tamarin is up to the task of reasoning about complex protocols with nested loops.

FUTURE WORK Of particular interest would be the formal analysis of PQ3 in conjunction with session handling, as implemented for iMessage. Whether PQ3's security guarantees as established here fully transfer to iMessage remains an open question. For example, [54] established that the Signal application may not provide post-compromise security, although the protocol does due to the implementation of session handling. Furthermore, our formal model could be extended to account for IDS key roll-over, i.e., of long-term identity and pre-keys.

8

CYCLIC INDUCTION FOR SECURITY PROTOCOL VERIFICATION

8.1 INTRODUCTION

In the previous chapter, we used trace induction to prove properties of protocols with nested loops. We presented a security proof for iMessage PQ3, which extends the classic double ratchet algorithm with KEMs to provide post-quantum security. Proving iMessage PQ3 secure using trace induction required substantial efforts and took many person-months (see Sections 7.4.3, 7.4.4). In this chapter, we present a novel, cyclic induction mechanism for the Tamarin prover that drastically simplifies the analysis of protocols using nested loops.

Our cyclic induction mechanism improves upon the state-of-the-art proof automation for modern protocol designs by tackling the non-termination issues of protocol models that use (nested) loops. In contrast to the proof techniques of trace induction (see Section 6.1) and the use of auxiliary lemmas, which require manually stating inductive conjectures and useful auxiliary lemmas, cyclic induction provides more proof automation with fewer auxiliary lemmas.

Recall that Tamarin uses constraint systems to represent the set of executions violating a given property (see Chapter 2). Constraint reduction essentially corresponds to a backwards search from an attack state to an initial state. By the reduction rules' soundness and completeness, reaching an initial state corresponds to an attack, while reducing all constraint systems to contradictions corresponds to a proof. Non-termination manifests itself as a never-ending backwards attack construction.

When explored in Tamarin's interactive mode, these constructions usually exhibit an indefinitely repeating behavioral pattern, such as the one shown on the right-hand branch of Figure 8.1. In this chapter, we formalize when such infinitely repeating patterns contradict the executions' well-foundedness. For that, we use the proof method *infinite descent* that was already used by Euclid, but first formalized by Fermat [124]. They applied it, for instance, to prove the non-existence of solutions to number-theoretic problems.

Modern accounts of deductive reasoning formalize such arguments using *cyclic proofs* (e.g., [32, 36]). Deductive reasoning rules decompose a proof goal into sub-goals, for example, proving $P \wedge Q$ can be reduced to proving P and proving Q. Applying these rules induces a proof tree, whose leaves are given by axioms, e.g., $A \vee \neg A$. Cyclic proofs allow one to prove a statement $P(\sigma(x))$ that instantiates a more general

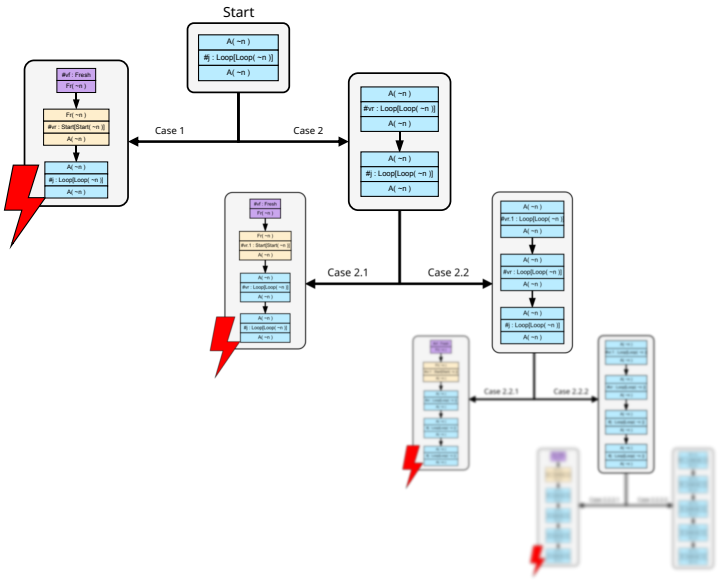


Figure 8.1: Example of non-termination in Tamarin. Without using trace induction, Tamarin can only apply one proof method, which results in a case split. Left-side case leads to a contradiction ($\lightning$) whereas right-side case recurses infinitely.

statement $P(x)$ encountered earlier in the proof by introducing a *backlink* from $P(\sigma(x))$ to $P(x)$. In this way, cyclic proofs generalize proof trees to proof graphs. Moreover, an additional *global soundness condition* ensures that all cycles correspond to well-founded inductive arguments. This form of inductive reasoning does not use any explicit induction rules.

Cyclic proofs' advantage over traditional inductive proofs is that inductive reasoning in general requires a priori ingenuity to choose an inductive conjecture, induction variables, and the position in a proof to apply induction. These choices, sometimes even called "eureka steps" to underscore their difficulty, make the automation of inductive proofs notoriously hard [37, 38]. In contrast, in cyclic proofs, these choices are made *implicitly* and *a posteriori*: The inductive statement and the position where to apply induction arise by the formation of backlinks, whereas the induction variables are determined by the global soundness condition. These properties of cyclic induction can both simplify proofs and offer promising potential for proof automation, even though, in general, undecidability still calls for user-provided inductive statements [37, 38]. Given its advantages over traditional induction, cyclic induction has been applied in many areas of logic and computer science, including first-order logic with inductive predicates [32], equational reasoning about functional programs [82], program logics [134, 156], program synthesis [79], and the μ-calculi [141, 147].

In this chapter, we formalize and implement cyclic proofs for security protocol verification in Tamarin, we prove its soundness, and we show that it substantially improves proof automation. Doing this required overcoming several challenges, both in terms of the logical formalism and proof search.

First, Tamarin uses constraint solving rather than a form of sequent calculus, commonly used for cyclic proofs. It was initially unclear whether cyclic proofs could be adapted to this setting. Besides formulas describing protocol properties, its constraint systems also include complex *graph constraints*, which describe execution steps and their dependencies. The interpretation of constraint systems as sequents that derive contradictions from the given constraints clarified the connection to existing cyclic proof systems and facilitated their adaption to our setting. Moreover, Tamarin's backward search for an initial state suggested the set of timepoints labeling execution steps as the natural well-founded domain for cyclic induction.

Second, we find that backlink formation in cyclic proofs requires two structural rules from sequent calculi, namely, weakening and cut, which we add to Tamarin's constraint reduction rules. *Weakening* generalizes a constraint system by discarding some constraints and *cut* introduces a new constraint and its negation in separate cases. To restrict the search space and achieve a high degree of automation, we use the structural rules in a strictly controlled way: our use of weakening is based on the protocol's looping structure and we use cut only with formulas that are either (i) instances of formulas that already exist in the constraint system or (ii) ordering constraints that preserve some of the weakened information.

Finally, as the backlink search is NP-complete, we devise an effective heuristic to perform backlink checks incrementally and discard impossible cases early.

We evaluate cyclic proofs and compare them with proofs by explicit trace induction. Our results show that our controlled use of the structural rules together with only minimal and generic changes to Tamarin's existing proof search strategies are sufficient to prove many lemmas with no or substantially fewer auxiliary lemmas than needed with trace induction.

CONTRIBUTIONS Our contributions are three-fold. First, we introduce cyclic proofs for protocol verification. In particular, we extend Tamarin's constraint reduction rules with structural rules (weakening and cut), define cyclic proofs for Tamarin, and prove their soundness. Second, we implement this proof system in Tamarin, which requires a major overhaul of Tamarin's internal structure. We also provide a set of heuristics to guide effective proof search in the cyclic proof system, including the controlled application of the structural rules. Third, we evaluate our approach on fourteen case studies ranging from simple

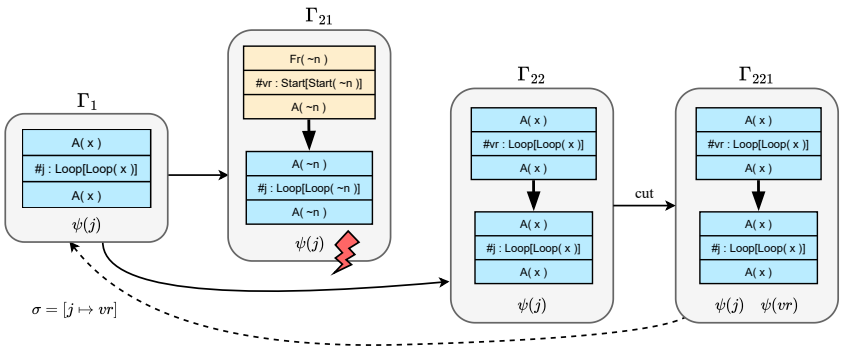


Figure 8.2: Proof of φ_1 using cyclic induction

protocols to a detailed model of the Signal protocol. We show that our implementation effectively reduces the number of auxiliary lemmas required. In particular, we require no auxiliary lemmas to prove message secrecy for Signal.

Our work opens an exciting new area in which automatic induction helps scale protocol verification. Whereas current automatic tools only provide rudimentary support for induction, we provide a fundamentally new and general induction mechanism. A wide variety of protocols will benefit from this. For example, protocols with nested loops, such as those using double ratchets, protocols with arbitrarily many participants, such as consensus protocols, and protocols involving participant groups, such as those employing threshold cryptography.

OUTLINE We proceed as follows. We start by introducing cyclic induction for Tamarin at a high-level in Section 8.2 and then proceed to formally develop it in Section 8.3. We describe our implementation of cyclic induction in Section 8.4 and evaluate it in Section 8.5. Finally, we review related work and draw conclusions in Sections 8.6 and 8.7.

8.2 OVERVIEW OF CYCLIC INDUCTION FOR TAMARIN

Recall the Loop model from Example 5, Section 6.1. The Loop theory models a loop that takes non-deterministically many steps and eventually terminates. We now show how to construct a cyclic proof for the property formalizing that any Loop fact is preceded by a Start fact:

$$\varphi_1 = \forall j, x.\, \mathsf{Loop}(x)@j \implies \exists i.\, \mathsf{Start}(x)@i \wedge i < j.$$

In a proof based on cyclic induction, there is no explicit induction rule or induction hypothesis. Instead, one uses the ordinary constraint reduction rules and tries to detect loops such as the one in Figure 8.1. One folds the associated infinite proof tree into a finite tree with *backlinks* and then tries to prove that the resulting cyclic

structure represents *well-founded* (non-circular!) reasoning. In this section, we provide some intuition for our cyclic proof system, which we subsequently formalize in Section 8.3.

8.2.1 *Backlink Formation*

We can detect loops by discovering constraint systems that are *subsumed* by more general ones appearing earlier in the proof. A constraint system Γ *subsumes* a leaf constraint system Γ' when $\Gamma\sigma \subseteq \Gamma'$ for some substitution σ. In this case, we can introduce a backlink from Γ' to Γ, thereby creating a *pre-proof graph* instead of a tree.

We now construct a cyclic proof of φ_1 for the Loop example. Looking at the system Γ_{22} in Figure 8.2, we see that the Loop node at vr repeats the initial Loop node at j in Γ_1. More precisely, the substitution $\sigma = [j \mapsto vr]$ maps the former node to the latter one. This does not, however, mean that $\Gamma_1\sigma \subseteq \Gamma_{22}$ as $\sigma(\psi(j)) = \psi(vr)$ is missing from Γ_{22}. However, we can derive $\sigma(\psi(j))$ if we add a *cut* constraint reduction rule. Using this rule, we add $\psi(vr)$ to the constraint system Γ_{22}, resulting in the constraint system Γ_{221}, and we show in a separate case that adding its negation leads to a contradiction. Indeed, $\neg\psi(vr)$ contradicts $\psi(j)$ because $vr \prec_{\Gamma_{22}} j$. After cutting in $\psi(vr)$, we have $\Gamma_1\sigma \subseteq \Gamma_{221}$ and can thus add a backlink from Γ_{221} to Γ_1 (dashed arrow in Figure 8.2). This results in a cyclic pre-proof of the property φ_1.

8.2.2 *Well-Founded Reasoning*

To avoid unsound circular reasoning, we must ensure that all cycles in a pre-proof correspond to well-founded inductive arguments. Suppose we have constructed a pre-proof for φ, but $\widehat{\varphi}$ is satisfiable, i.e., $(dg, \theta) \Vdash_E \widehat{\varphi}$ for some dg and θ. Since Tamarin's constraint reduction rules are sound, every satisfiable constraint system in the pre-proof is either solved or has a satisfiable child constraint system.

We build a path $\pi = n_0 n_1 \cdots$ as follows. Let n_0 be $\widehat{\varphi}$ (satisfiable) and n_{i+1} be one of the satisfiable children of n_i, as long as n_i has children in the pre-proof. Suppose π were finite. Then, its last constraint system must be a leaf in the pre-proof and thus cannot be satisfiable because all leafs in the pre-proof must be contradictory axiom constraint systems. Thus π must be infinite.

To establish that an infinite path π corresponds to well-founded reasoning, we require that every cycle in a pre-proof "progresses." Progress ensures that repeating patterns captured by backlinks occur at smaller and smaller timepoints along an infinite path, which contradicts the well-foundedness of timepoints. This follows the intuition that repeatedly solving the same premises, creating increasingly larger constraint systems, does not lead to a finite attack.

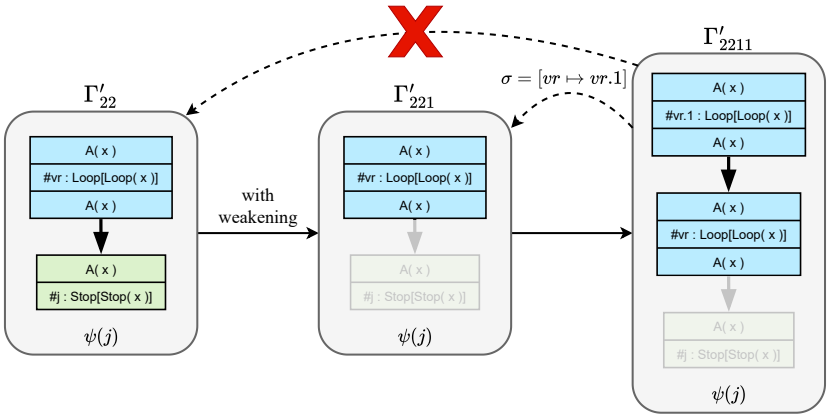


Figure 8.3: Weakening in cyclic induction proof of φ_0

More precisely, we say that a backlink from Γ' to Γ where $\Gamma\sigma \subseteq \Gamma'$ *progresses on* j when there is a temporal variable j such that $\sigma(j) \prec_{\Gamma'} j$ (as is the case in our example above). We ensure progress by requiring that every backlink progresses on at least one temporal variable j. If we were to traverse a backlink infinitely often, then j would decrease infinitely often. However, as temporal variables are interpreted in $\mathbb{N}$, this is impossible.[1]

Additionally, we must ensure that backlinks are mutually compatible if they are part of the same strongly connected subgraph (SCS). It could happen that two backlinks ℓ_1 and ℓ_2 "destroy" each other's progress, for example, if each of them increases the variable on which the other progresses. To prevent this, we will define a notion of *preservation*, which we use to define a *discharge condition* for pre-proofs and an algorithm to check it.

8.2.3 *Explicit Weakening*

We now turn to the property that we wanted to prove initially when introducing Example 5, namely that every Stop fact is preceded by a Start fact:

$$\varphi_0 = \forall j, x.\, \mathsf{Stop}(x)@j \implies \exists i.\, \mathsf{Start}(x)@i \wedge i < j.$$

As we showed in Section 6.1, trying to prove this property using trace induction does not lead to a successful proof. One can find a proof, however, when using φ_1 as an auxiliary lemma.

We next show how to prove this property with cyclic induction but without any auxiliary lemmas. The initial node constraint is a Stop node at j and subsequently solving that node's premise $\mathsf{A}(x)$

[1] In Section 2.4, we introduce temporal variables as being interpreted in $\mathbb{Q}$. We will later show that the temporal variables that we consider for progress must be interpreted in $\mathbb{N}$.

produces two constraint systems, one for Start and one for Loop. The former leads to an immediate contradiction as before. The latter is depicted as Γ'_{22} in Figure 8.3. This figure only shows the backlink in the cyclic proof of φ_0 and omits all non-looping branches. Without any other steps, solving the premise $\mathsf{A}(x)$ again would lead to a constraint system similar to Γ'_{2211}, including the faded-out node. There is no backlink from this constraint system to Γ'_{22} because, to ensure progress, we must map vr to $vr.1$. However, if we do that, we require an edge from $vr.1$ to j in Γ'_{2211}, which is missing.

We solve this problem by simply "deleting" the Stop node at j. This is called *weakening* and is a well-established and sound proof rule for generalization; we define a *weakening rule* in Section 8.3. Without the Stop node, there is a backlink from Γ'_{2211} to Γ'_{221}, as illustrated in Figure 8.3. Note that, in contrast to the previous example, there is no need to use the cut rule here because the timepoint j in the formula $\psi(j)$ is not substituted by σ.

8.3 CYCLIC INDUCTION FOR TAMARIN

In this section, we introduce cyclic induction for Tamarin formally, following the intuition presented in the previous section. We begin by introducing two new constraint solving rules, cut and weakening, in Section 8.3.1. We then introduce cyclic pre-proofs in Section 8.3.2 and in Section 8.3.3 show when pre-proofs correspond to sound proofs, using temporal variables to establish progress. Finally, we present an algorithm to check when pre-proofs are proofs in Section 8.3.4.

8.3.1 *Structural Constraint Reduction Rules*

We extend Tamarin's constraint reduction rules with two additional rules: *weakening* and *cut*, which are well-known structural rules in logic. To maintain important invariants of Tamarin's constraint solving algorithm, we use restricted versions of these rules, but omit these rather technical restrictions for clarity (see Appendix A).

The cut rule $\mathcal{S}_\Delta$ introduces a new set of formulas Δ and proves each formula's validity in separate cases. The weakening rule $\mathcal{S}_W$ simply removes some constraints.

$$\mathcal{S}_\Delta : \Gamma \rightsquigarrow (\Gamma, \Delta) \parallel \|_{\varphi \in \Delta}(\Gamma, \widehat{\varphi}) \qquad \mathcal{S}_W : \Gamma, \Delta \rightsquigarrow \Gamma$$

Clearly, $\mathcal{S}_\Delta$ is sound and complete and $\mathcal{S}_W$ is sound. However, $\mathcal{S}_W$ is incomplete, since $sols(\Gamma \cup \Delta) \subseteq sols(\Gamma)$. Hence, the models found after applying weakening may not constitute attacks.

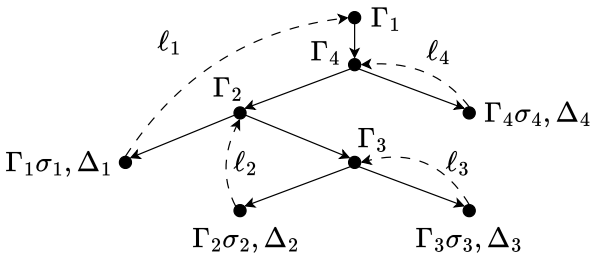


Figure 8.4: A pre-proof $(\mathcal{D}, \mathcal{L}, \Sigma)$. Solid arrows are edges in $\mathcal{E}$ and dashed arrows are backlinks in $\mathcal{L}$.

8.3.2 *Pre-Proofs*

To introduce cyclic reasoning in Tamarin, we augment derivation trees, as defined in Section 2.6, with *backlinks* that introduce cycles. We add a backlink from a leaf Γ' to an inner node Γ on the path from the root to Γ' whenever Γ is a more general constraint system than Γ', i.e., $\Gamma\sigma \subseteq \Gamma'$ for some substitution σ.

Definition 4 (Pre-Proof)**.** A *pre-proof* $\mathcal{P} = (\mathcal{D}, \mathcal{L}, \Sigma)$ of a property φ consists of a derivation tree $\mathcal{D} = (\mathcal{N}, \mathcal{E}, \gamma)$ for φ, a *partial backlink function* $\mathcal{L} : \mathcal{N} \rightharpoonup \mathcal{N}$, and a *partial substitution function* $\Sigma : \mathcal{N} \rightharpoonup \mathbb{S}$, such that the domain of $\mathcal{L}$ and Σ is the set of open leaves of $\mathcal{D}$ and, for each open leaf v with $\sigma_v = \Sigma(v)$, the node $w = \mathcal{L}(v)$ lies on the path from $\mathcal{D}$'s root node v_0 to v and $\sigma_v(\gamma(w)) \subseteq \gamma(v)$.

We will often write σ_v for $\Sigma(v)$ and we also use $\mathcal{L}$ to denote the function's graph, which consists of *backlinks*.

Definition 5 (Pre-proof graph)**.** The *pre-proof graph* of $\mathcal{P} = (\mathcal{D}, \mathcal{L}, \Sigma)$ is $\mathcal{G}(\mathcal{P}) = (\mathcal{N}, \mathcal{E} \cup \mathcal{L}, \gamma)$. For a backlink $\ell = (v, w) \in \mathcal{L}$, we denote the path from w to v in $\mathcal{D}$ by $\pi(\ell)$.

Figure 8.4 gives an example of a pre-proof. Every leaf of the derivation tree either has an associated backlink or yields a contradiction.

A strongly connected subgraph (SCS) of $\mathcal{G}(\mathcal{P})$ is a subgraph of $\mathcal{G}(\mathcal{P})$ that contains at least one edge and where there is a path from every node to every other node. A strongly connected component (SCC) is a maximal strongly connected subgraph. We henceforth only consider SCSes and SCCs that contain at least one edge without explicitly mentioning it.

Note that any SCS S in $\mathcal{G}(\mathcal{P})$ is characterized by the set $\mathcal{L}_S$ of backlinks it contains: S contains the nodes and edges of the paths $\pi(\ell)$ for some $\ell \in \mathcal{L}_S$ and the backlinks in $\mathcal{L}_S$. We say that a set $L \subseteq \mathcal{L}$ *induces* an an SCS S if $L = \mathcal{L}_S$.

8.3.3 *Cyclic Proofs and their Soundness*

We next show when pre-proofs are sound, i.e., when a cyclic pre-proof for a property φ is a cyclic proof that indeed implies that φ holds. Following the intuition presented in Section 8.2.2, the critical part of a sound definition of (cyclic) proofs is defining suitable notions of progress and preservation that ensure that the inductive reasoning embodied in the pre-proof is well-founded.

Definition 6 (Progress and preservation)**.** Given a pre-proof $\mathcal{P} = (\mathcal{D}, \mathcal{L}, \Sigma)$ and a temporal variable $i \in \mathcal{V}_{tmp}$, a backlink $\ell = (v, w) \in \mathcal{L}$:

- *preserves* i if i occurs free in all constraint systems labeling the nodes on $\pi(\ell)$ and $\sigma_v(i) \preceq_{\gamma(v)} i$, and
- *progresses on* i, if ℓ preserves i and $\sigma_v(i) \prec_{\gamma(v)} i$.

Definition 7 (Proof)**.** A pre-proof $\mathcal{P} = (\mathcal{D}, \mathcal{L}, \Sigma)$ of a property φ is a *proof* of φ if it satisfies the following *discharge condition*: for each strongly connected subgraph S of $\mathcal{G}(\mathcal{P})$, there is a temporal variable $i \in \mathcal{V}_{tmp}$ such that some backlink $\ell \in \mathcal{L}_S$ progresses on i, and all backlinks $\ell' \in \mathcal{L}_S$ preserve i.

We next state auxiliary lemmas for our soundness theorem. Point (iii) of 2 is needed, since timepoints are in general interpreted in $\mathbb{Q}$, which is not well-founded. However, timepoints related to action and node constraints are always interpreted in $\mathbb{N}$.

Lemma 2 (Local soundness)**.** Tamarin's constraint reduction rules are sound. In particular, given a rule $\Gamma \rightsquigarrow \{\Gamma_1, \ldots, \Gamma_n\}$ and a model (dg, θ) satisfying Γ, there is valuation θ' such that

(i) (dg, θ') satisfies some Γ_k,

(ii) θ agrees with θ' on all free variables common to Γ and Γ_k, and

(iii) the property that all free temporal variables are valuated in $\mathbb{N}$ is preserved, i.e., if $\theta(fv_{tmp}(\Gamma)) \subseteq \mathbb{N}$ then $\theta'(fv_{tmp}(\Gamma_k)) \subseteq \mathbb{N}$.

Proof. Points (i) and (ii) can be seen by inspection of the rules' soundness proofs in [111, Theorems 4 and 11] and [53, Lemmas 4, 6, and 8].[2] It is easy to see that this part also holds for the cut and weakening rules, $\mathcal{S}_\Delta$ and $\mathcal{S}_W$, introduced in Section 8.3.1. In particular, for all rules we have that either $\theta' = \theta$ or θ' extends θ with mappings for the fresh free variables of Γ_k. The latter is the case for the rules $\mathcal{S}_@$, $\mathcal{S}_{!\mathsf{K}^\uparrow@}$, $\mathcal{S}_\approx$, $\mathcal{S}_\exists$, $\mathcal{S}_\triangleright$, $\mathcal{S}_{!\mathsf{K}^\uparrow\triangleright}$, $\mathcal{S}_{!\mathsf{K}^\downarrow\triangleright}$, $\mathcal{S}_{\dashrightarrow}$, and $\mathcal{S}_{\neg,\mathsf{last}}$, which all introduce fresh variables (see Appendix A). The former is the case for all other rules, except $\mathcal{S}_\Delta$. For $\mathcal{S}_\Delta$, either case may apply, depending on whether the cut introduces fresh (term) variables.

[2] Recall that what we call soundness here is called completeness from the constraint solving perspective of [53, 111].

Regarding point (iii), the preservation of temporal variable valuation in $\mathbb{N}$, the only interesting rules here are those introducing fresh temporal variables, which we cover below. The cases for the other rules follow from the first part of the lemma. Note, in particular, that the cut rule $\mathcal{S}_\Delta$ does not introduce any fresh temporal variables.

$\mathcal{S}_\exists$: We focus on the interesting case of temporal variables here. Consider $\varphi = \exists i.\, f@i \wedge \psi \in \Gamma$ for $i \in \mathcal{V}_{tmp}$. We know that φ must be of this form since all formulas in Γ are guarded by **WF4'**. We can assume without loss of generality that $i \notin fv(\Gamma)$. Hence, we have $\Gamma_1 = \Gamma \cup \{f@i \wedge \psi\}$. Since $(dg, \theta') \Vdash_E \Gamma_1$ for some θ' extending θ's domain with i, we have $(dg, \theta') \Vdash_E f@i$, which implies $i \in idx(I)$ and hence $\theta'(i) \in \mathbb{N}$.

Note that guardedness is defined on vectors of quantified variables, but $\mathcal{S}_\exists$ on a single variable. Naturally, $\exists \vec{x}.\psi$ is equal to $\exists x_0. \ldots \exists x_n.\, \psi$ so we can assume without loss of generality that the timepoint quantified within $\vec{x}$ is solved last (as done above).

$\mathcal{S}_\triangleright$: In this case, $(f \triangleright_v j) \in \Gamma$ and $\Gamma_k = \Gamma \cup \{i : ri, (i, u) \rightarrowtail (j, v)\}$ for a fresh $i \in \mathcal{V}_{tmp}$. Since $(dg, \theta') \Vdash_E \Gamma_k$ and hence $(dg, \theta') \Vdash_E i : ri$ for some θ' extending θ's domain with i, we have $\theta'(i) \in idx(I)$ and hence $\theta'(i) \in \mathbb{N}$.

$\mathcal{S}_{!\mathsf{K}^\uparrow_\triangleright}$: For this rule, we have $(!\mathsf{K}^\downarrow(t) \triangleright_v j) \in \Gamma$ and $\Gamma_1 = \Gamma \cup \{!\mathsf{K}^\uparrow(t)@j, j < i\}$ for a fresh $j \in \mathcal{V}_{tmp}$. Since $(dg, \theta') \Vdash_E \Gamma'$ and hence $(dg, \theta') \Vdash_E !\mathsf{K}^\uparrow(t)@j$ for some θ' extending θ's domain with j, we have $\theta'(j) \in idx(I)$ and hence $\theta'(j) \in \mathbb{N}$.

$\mathcal{S}_{!\mathsf{K}^\downarrow_\triangleright}$, $\mathcal{S}_{\dashrightarrow}$: These cases are similar to $\mathcal{S}_\triangleright$.

This concludes the proof of the lemma. □

Lemma 3. Let $(v, w) \in \mathcal{L}$ be a backlink and (dg, θ) a model satisfying $\gamma(v)$. Then $(dg, \theta \circ \sigma)$ satisfies $\gamma(w)$.

Proof. By the definition of pre-proofs. □

Theorem 1 (Soundness)**.** Let $\mathcal{P}$ be a proof for a given protocol model (R, E) and a guarded trace property φ. Then $R \models_E \varphi$.

Proof (sketch). Suppose for a contradiction that $\hat{\varphi}$ is (R, E)-satisfiable. Then there is a model $(dg, \theta_0) \Vdash_E \hat{\varphi}$. Using Lemmas 2 and 3, we "track" the model dg in the pre-proof graph $\mathcal{G}(\mathcal{P})$ by constructing an infinite sequence $\{(v_k, \theta_k)\}_{k \in \mathbb{N}}$ of pairs of nodes and valuations such that:

1. For all $k \in \mathbb{N}$, $(dg, \theta_k) \Vdash_E \gamma(v_k)$.
2. For all temporal variables $i \in fv(\gamma(v_k))$, $\theta_k(i) \in \mathbb{N}$.

Observe that the path $\pi = \{v_k\}_{k\in\mathbb{N}}$ will eventually stay in one of the pre-proof's SCSes, and we can choose this SCS minimally. Then, we choose i to be the variable that is progressed by one backlink ℓ in that SCS according to the definition of proofs (Definition 7). Note that π must traverse ℓ infinitely often. If this were not the case, the SCS was not chosen minimally. Thus, the sequence $\{\theta_k(i)\}_{k\in\mathbb{N}}$ decreases infinitely often, which contradicts the well-foundedness of the natural numbers. □

The above proof sketch misses what happens to i when traversing backlinks other than ℓ and when traversing edges that are not backlinks. For example, the sequence $0, 1, 0, 1, 0, 1, \ldots$ also decreases infinitely often, but this does not contradict the natural numbers' well-foundedness as it also increases infinitely often. In the full proof below, we account for such cases by explicitly constructing an infinite sequence of nodes and valuations ρ such that the valuations preserve i when traversing pre-proof edges other than ℓ.

Proof. Let $\mathcal{P} = (\mathcal{D}, \mathcal{L}, \Sigma)$ be a proof of the guarded trace property φ with $\mathcal{D} = (\mathcal{N}, \mathcal{E}, \gamma)$. Assume for a contradiction that φ is not (R, E)-valid, i.e., there is a trace tr of (R, E) and a valuation θ such that $(tr, \theta) \not\models_E \varphi$. Then there is a dependency graph dg and valuation θ_0 such that $(dg, \theta_0) \Vdash_E \{\hat{\varphi}\}$.

Observe that given the soundness of Tamarin's constraint solving rules, we can "track" the solution dg in the proof graph $G(\mathcal{P})$ using Lemmas 2 and 3. We use these two lemmas to construct an infinite sequence $\rho = \{(v_n, \theta_n)\}_{n\in\mathbb{N}}$ of nodes and valuations such that $\pi = \{v_n\}_{n\in\mathbb{N}}$ is an infinite path in $\mathcal{G}(\mathcal{P})$ and, for all $k \in \mathbb{N}$, we have $(dg, \theta_k) \Vdash_E \gamma(v_k)$ and $\theta_k(fv_{tmp}(\gamma(v_k))) \subseteq \mathbb{N}$.

We set v_0 to be $\mathcal{G}(\mathcal{P})$'s root node. Thus, the sequence ρ starts with $\rho_0 = (v_0, \theta_0)$. Note that $(dg, \theta_0) \Vdash_E \gamma(v_0)$ by assumption, and, as $\hat{\varphi}$ is closed, we have $\theta_0(fv_{tmp}(\gamma(v_0))) = \emptyset \subseteq \mathbb{N}$. Let $\rho_k = (v, \theta)$ be the last element of the sequence ρ constructed so far. By construction $(dg, \theta) \Vdash_E \gamma(v)$ and $\theta(fv_{tmp}(\gamma(v))) \subseteq \mathbb{N}$ hold. There are two cases:

1. v is an inner node of $\mathcal{D}$. Since $(dg, \theta) \Vdash_E \gamma(v)$, we use Lemma 2 to obtain a successor node v' of v in $\mathcal{D}$ and a valuation θ' such that $(dg, \theta') \Vdash_E \gamma(v')$, θ and θ' agree on all free variables common to $\gamma(v)$ and $\gamma(v')$, and $fv_{tmp}(\gamma(v')) \subseteq \mathbb{N}$. We set $\rho_{k+1} := (v', \theta')$.

2. v is a leaf node of $\mathcal{D}$. For axiom leaves v, we have $\gamma(v) = \bot$, which contradicts $(dg, \theta) \Vdash_E \gamma(v)$. Therefore, there must be

a backlink $(v, v') \in \mathcal{L}$ for some node v'. Using Lemma 3, we derive $(dg, \theta \circ \sigma_v) \Vdash_E \gamma(v')$ from $(dg, \theta) \Vdash_E \gamma(v)$. We also have

$$\begin{aligned} & (\theta \circ \sigma_v)(fv_{tmp}(\gamma(v'))) \\ = \; & \theta(fv_{tmp}(\sigma_v(\gamma(v')))) \\ \subseteq \; & \theta(fv_{tmp}(\gamma(v))) && \text{by Definition 4} \\ \subseteq \; & \mathbb{N}. && \text{by construction of } \rho_k \end{aligned}$$

We set $\rho_{k+1} := (v', \theta \circ \sigma_v)$.

We can continue this construction indefinitely to obtain an infinite sequence ρ, as we can never reach an axiom leaf.

Observe that after a certain point n, every node v_k ($k \geq n$) must appear infinitely often on π. Therefore, the set $\{v_n, v_{n+1}, v_{n+2}, \dots\}$ induces an SCS S of $\mathcal{G}(\mathcal{P})$. By Definition 7, there exists a temporal variable $i \in \mathcal{V}_{tmp}$ such that

(i) there exists a backlink $\ell \in \mathcal{L}_S$ that progresses on i, and

(ii) all backlinks $\ell' \in \mathcal{L}_S$ preserve i.

Let ℓ be the backlink that progresses on i in $\mathcal{L}_S$. Then, for $k \geq n$ and every two successive pairs (v_k, θ_k) and (v_{k+1}, θ_{k+1}), one of two cases applies:

1. v_k is an inner node of $\mathcal{D}$. Then, there must be a backlink $\ell' = (v, w) \in \mathcal{L}_S$ such that v_k lies on the path $\pi(\ell')$ from w (inner node) to v (leaf). Since ℓ' preserves i, we know that i occurs free in both $\gamma(v_k)$ and $\gamma(v_{k+1})$ and therefore $\theta_{k+1}(i) = \theta_k(i)$.

2. There exists a backlink $\ell' = (v_k, v_{k+1}) \in \mathcal{L}_S$ and $\theta_{k+1} = \theta_k \circ \sigma_{v_k}$. Since ℓ' preserves i (all backlinks in $\mathcal{L}_S$ do), we have $\sigma_{v_k}(i) \preceq_{\gamma(v_k)} i$ and therefore $\theta_{k+1}(i) \leq \theta_k(i)$ by Lemma 1, point (ii).

   For $\ell' = \ell$ we have progress on i, i.e., $\sigma_{v_k}(i) \prec_{\gamma(v_k)} i$ and hence $\theta_k(\sigma_{v_k}(i)) = \theta_{k+1}(i) < \theta_k(i)$, by the construction of ρ and Lemma 1, point (i).

This shows that the sequence $\{\theta_k(i)\}_{k \geq n}$ monotonically decreases. Since every node of S occurs on the path $\pi(\ell')$ of some backlink ℓ' and all backlinks in $\mathcal{L}_S$ preserve i, we derive that i occurs free in $\gamma(v)$ for all $v \in S$ and therefore $\theta_k(i) \in \mathbb{N}$ for all $k \geq n$. Since the progressing backlink ℓ is traversed infinitely often on the infinite path π in $\mathcal{G}(\mathcal{P})$, the sequence $\{\theta_k(i)\}_{k \geq n}$ strictly decreases infinitely often, which contradicts the well-foundedness of $(\mathbb{N}, <)$. □

8.3.4 *Alternative Discharge Condition*

We give an alternative condition for pre-proofs that can be easily implemented as an algorithm to check whether a pre-proof is a proof.

Progress Orders

This condition is based on progress orders [140, 146][3], which organize backlinks into a partial order, labeled with temporal variables. A progress order that satisfies our alternative discharge condition ensures that there exists a temporal variable decreasing infinitely often along every infinite path. The order identifies such a variable as the $\sqsubseteq$-greatest element for every SCS and, in turn, for every infinite path. We show that this alternative discharge condition is equivalent to the original one in Definition 7.

Definition 8 (Progress order)**.** Let $\mathcal{P} = (\mathcal{D}, \mathcal{L}, \Sigma)$ be a pre-proof. A *progress order* $(\mathcal{L}, \sqsubseteq, \iota)$ for $\mathcal{P}$ is a partial order $(\mathcal{L}, \sqsubseteq)$ on the set of backlinks and a labeling function $\iota : \mathcal{L} \to \mathcal{V}_{tmp}$ assigning to each backlink ℓ a temporal variable $\iota(\ell)$ such that, for every SCS S of $\mathcal{G}(\mathcal{P})$, $\mathcal{L}_S$ has a $\sqsubseteq$-greatest element.

Definition 9 (Alternative discharge condition)**.** A progress order $(\mathcal{L}, \sqsubseteq, \iota)$ for a pre-proof $\mathcal{P} = (\mathcal{D}, \mathcal{L}, \Sigma)$ discharges $\mathcal{P}$ if for all $\ell \in \mathcal{L}$: ℓ progresses on $\iota(\ell)$, and ℓ preserves $\iota(\ell')$, for all $\ell \sqsubseteq \ell'$.

Note that every total order on backlinks is a progress order and for total orders the discharge condition corresponds to a lexicographic order on the temporal variables associated to the backlinks.

Algorithm 1 Returns a discharging progress order $(\mathcal{L}, \sqsubseteq, \iota)$ for a pre-proof $\mathcal{P} = (\mathcal{D}, \mathcal{L}, \Sigma)$ if one exists and fails otherwise.

let $\mathcal{C}$ be a partitioning of $\mathcal{L}$ into sets inducing $\mathcal{G}(\mathcal{P})$'s SCCs
return *progress_order* $\mathcal{P}$ $(\mathrm{Id}_{\mathcal{L}})$ $\emptyset$ $\mathcal{C}$

function *progress_order* $\mathcal{P}$ $(\sqsubseteq)$ ι $\mathcal{C}$ $=$
if $\mathrm{dom}(\iota) = \mathcal{L}$ **then**
  return $(\mathcal{L}, \sqsubseteq, \iota)$
else
  let $L \in \mathcal{C}$ **and** $\ell \in L$ **and** $i \in \mathcal{V}_{tmp}$ **such that**
    ℓ progresses on i and all $\ell' \in L$ preserve i
  if these do not exist **then**
    return failure
  else
    let $\sqsubseteq' = \sqsubseteq \cup \{(\ell', \ell) \mid \ell' \in L\}$
    let $\mathcal{C}_L$ be the partitioning of $L \setminus \{\ell\}$ into subsets
      that induce the SCCs of $(\mathcal{N}, \mathcal{E} \cup L \setminus \{\ell\})$
    let $\mathcal{C}' = (\mathcal{C} \setminus \{L\}) \cup \mathcal{C}_L$
    return *progress_order* $\mathcal{P}$ $(\sqsubseteq')$ $(\iota[\ell \mapsto i])$ $\mathcal{C}'$
  end if
end if

[3] [140, 146] use the term "induction orders", which we avoid for clarity.

Algorithm for Checking Discharge Condition

We give an algorithm that, given a pre-proof $\mathcal{P} = (\mathcal{D}, \mathcal{L}, \Sigma)$, computes a discharging progress order $(\mathcal{L}, \sqsubseteq, \iota)$ if one exists and fails otherwise (Algorithm 1). The algorithm first determines a partitioning $\mathcal{C}$ of the set of backlinks $\mathcal{L}$ into sets inducing $\mathcal{G}(\mathcal{P})$'s SCCs (line 1) and then calls the recursive function *progress_order* (line 2). Besides the (fixed) pre-proof $\mathcal{P}$, this function has three parameters: the current ordering $\sqsubseteq$ on $\mathcal{L}$, the current labeling of backlinks with temporal variables ι, and a set $\mathcal{C}$ of subsets of $\mathcal{L}$, which partitions the set $\mathcal{L} \setminus \mathrm{dom}(\iota)$ into the subsets that induce the SCCs of the graph $(\mathcal{N}, \mathcal{E} \cup \mathcal{L} \setminus \mathrm{dom}(\iota))$. Initially, the relation $\sqsubseteq$ is the identity relation $\mathrm{Id}_{\mathcal{L}}$ on $\mathcal{L}$ and the labeling ι (and hence $\mathrm{dom}(\iota)$) is empty. Note that, since the initial partial order is flat, all elements of $\mathcal{L}$ are minimal.

Each recursion adds some $\ell \in \bigcup \mathcal{C}$ to $\mathrm{dom}(\iota)$. The idea is that progress and preservation for all SCSs S of $\mathcal{G}(\mathcal{P})$ containing backlinks in $\mathrm{dom}(\iota)$ is already covered (in the sense of Definition 7), while these properties remain to be shown for the remaining SCSs, each of which is induced by (a subset of) some $L \in \mathcal{C}$.

The algorithm terminates and returns $(\mathcal{L}, \sqsubseteq, \iota)$ when the labeling ι covers all of $\mathcal{L}$ (line 6). Otherwise, it determines a backlink ℓ in one of the sets $L \in \mathcal{C}$ and an associated variable on which ℓ progresses and which all elements in L preserve (lines 8 and 9). These exist if $\mathcal{P}$ is a proof. Otherwise the algorithm fails (line 11). The backlink ℓ then becomes the greatest element of L (line 13). The partition $\mathcal{C}$ is updated by removing the set L from $\mathcal{C}$ and replacing it by the sets in the partitioning of $L \setminus \{\ell\}$ into subsets inducing the SCCs of the graph $(\mathcal{N}, \mathcal{E} \cup L \setminus \{\ell\})$ (lines 14-16). The mapping $[\ell \mapsto i]$ is then added to ι and the function *progress_order* is recursively called with the updated parameters (line 17).

Example 6 (Computing a progress order)**.** Figure 8.4's pre-proof has four backlinks $\mathcal{L} = \{\ell_1, \ell_2, \ell_3, \ell_4\}$. Suppose the backlinks progress on or preserve the temporal variables i, j, k, and l as follows:

Backlink	Progresses	Also Preserves
ℓ_1	i	
ℓ_2	j	i
ℓ_3	k	i, j
ℓ_4	l	i

We now use the function *progress_order* (Algorithm 1) to compute a progress order as follows. Note that $\mathcal{L}$ induces the single SCC of the pre-proof graph. We start with $\sqsubseteq_0 = \mathrm{Id}_{\mathcal{L}}$, the identity relation on $\mathcal{L}$, ι_0 the empty labeling, and $\mathcal{C}_0 = \{\mathcal{L}\}$. Observing that ℓ_1 progresses on i and all other backlinks preserve i, we place ℓ_1 above the other backlinks in $\sqsubseteq_1$ and set $\iota_1 = \iota_0[\ell_1 \mapsto i]$ as in Figure 8.5a. The updated set $\mathcal{C}_1$ then partitions the set of remaining backlinks $\{\ell_2, \ell_3, \ell_4\}$ into the

(a) Progress order after one iteration.

(b) Final progress order.

Figure 8.5: Progress order construction for Figure 8.4's pre-proof.

two sets $\{\ell_2, \ell_3\}$ and $\{\ell_4\}$, inducing the remaining SCCs (i.e., ignoring ℓ_1's backlink in the pre-proof graph). Next, we see that ℓ_2 progresses on j, which ℓ_3 preserves. Hence, we place ℓ_2 on top of ℓ_3 in $\sqsubseteq_2$ and set $\iota_2 = \iota_1[\ell_2 \mapsto k]$. Removing ℓ_2's backlink in turn, $\mathcal{C}_2$ contains the sets $\{\ell_3\}$ and $\{\ell_4\}$, inducing the only remaining SCCs. As ℓ_3 progresses on k and ℓ_4 progresses on l, we add these as mappings to the final labeling ι in two additional recursive calls of *progress_order*.

The resulting progress order is depicted in Figure 8.5b. Note that, since ℓ_4 does not preserve j and k, requiring a linear progress order as a discharge condition would be too strong for this pre-proof.

The alternative discharge condition is equivalent to the original one from Definition 7, as stated in the following proposition, which we constructively prove based on Algorithm 1.

Proposition 1 (Equivalence of discharge conditions)**.** A pre-proof $\mathcal{P} = (\mathcal{D}, \mathcal{L}, \Sigma)$ is a proof if and only if there exists a progress order $(\mathcal{L}, \sqsubseteq, \iota)$ for $\mathcal{P}$.

Proof. Let $\mathcal{P} = (\mathcal{D}, \mathcal{L}, \Sigma)$ be a pre-proof.

$\Rightarrow$: Suppose that $\mathcal{P}$ is a proof. We show that Algorithm 1 constructs a (forest-shaped) progress order $(\mathcal{L}, \sqsubseteq, \iota)$.

The function *progress_order* maintains the following three invariants.

1. $(\mathcal{L}, \sqsubseteq_k)$ is a (forest-shaped) partial order with minimal elements $\bigcup \mathcal{C}_k$ and $\mathcal{C}_k \cup \{\mathrm{dom}(\iota_k)\}$ partitions $\mathcal{L}$.
2. For all $\mathcal{L}_S \subseteq \mathcal{L}$ inducing an SCS S of $\mathcal{G}(\mathcal{P})$, either $\mathcal{L}_S$ has a $\sqsubseteq_k$-greatest element or there exists some SCS S' of $\mathcal{G}(\mathcal{P})$ such that $\mathcal{L}_S \subseteq \mathcal{L}_{S'}$ and $\mathcal{L}_{S'} \in \mathcal{C}_k$.
3. All $\ell \in \mathrm{dom}(\iota_k)$ satisfy the progress and preservation conditions of Definition 9.

All three invariants hold trivially for the initial $\sqsubseteq_0$, ι_0 and $\mathcal{C}_0$. Suppose now that the invariants hold for $\sqsubseteq_k$ and ι_k and $\mathcal{C}_k$. If $\mathrm{dom}(\iota_k) = \mathcal{L}$,

then it follows from the invariants that $(\mathcal{L}, \sqsubseteq_k, \iota_k)$ is a progress order discharging $\mathcal{P}$.

Otherwise, the algorithm picks $L_k \in \mathcal{C}_k$, $\ell_k \in L_k$ and $i \in \mathcal{V}_{tmp}$ such that ℓ_k progresses on i and all $\ell' \in L_k$ preserve i. These exist by the assumption that $\mathcal{P}$ is a proof. We show that the invariants also hold for the newly constructed $\sqsubseteq_{k+1}$, ι_{k+1}, and $\mathcal{C}_{k+1}$. It is easy to see that the construction preserves the first and third invariants. For the second invariant, consider any SCS S of $\mathcal{G}(\mathcal{P})$. From the induction hypothesis, we know that either (i) $\mathcal{L}_S$ has a $\sqsubseteq_k$-greatest element ℓ or (ii) $\mathcal{L}_S \subseteq \mathcal{L}_{S'}$ and $\mathcal{L}_{S'} \in \mathcal{C}_k$ for some SCS S' of $\mathcal{G}(\mathcal{P})$.

In case (i), we distinguish whether or not $\ell \in L_k$. If $\ell \in L_k$, then ℓ is a minimal element of $\sqsubseteq_k$ by invariant 1 and thus $\mathcal{L}_S = \{\ell\}$. Therefore, ℓ is trivially also the $\sqsubseteq_{k+1}$-greatest element of $\mathcal{L}_S$. Otherwise, we have $\ell \notin L_k$ and thus ℓ is also the $\sqsubseteq_{k+1}$-greatest element of $\mathcal{L}_S$.

For case (ii), let S' be an SCS of $\mathcal{G}(\mathcal{P})$ such that $\mathcal{L}_S \subseteq \mathcal{L}_{S'}$ and $\mathcal{L}_{S'} \in \mathcal{C}_k$. If $\mathcal{L}_{S'} = L_k$ then we further distinguish whether or not $\ell_k \in \mathcal{L}_S$. If $\ell_k \in \mathcal{L}_S$, then this is clearly the $\sqsubseteq_{k+1}$-greatest element of $\mathcal{L}_S$. If $\ell_k \notin \mathcal{L}_S$, it follows from the construction of $\mathcal{C}_{k+1}$, in particular the partitioning of $L_k \setminus \{\ell_k\}$ into subsets inducing the SCCs of $(\mathcal{N}, \mathcal{E} \cup L_k \setminus \{l_k\}, \gamma)$, that there exists some SCS S'' such that $\mathcal{L}_S \subseteq \mathcal{L}_{S''} \subseteq \mathcal{L}_{S'}$ and $\mathcal{L}_{S''} \in \mathcal{C}_{k+1}$. Otherwise, if $\mathcal{L}_{S'} \neq L_k$, we have $\mathcal{L}_{S'} \in \mathcal{C}_{k+1}$ and, by assumption, $\mathcal{L}_S \subseteq \mathcal{L}_{S'}$.

$\Leftarrow$: Suppose that there is a progress order $(\mathcal{L}, \sqsubseteq, \iota)$ that discharges $\mathcal{P}$. Let S be an SCS of $\mathcal{G}(\mathcal{P})$. By Definitions 8 and 9 there is a $\sqsubseteq$-greatest backlink $\ell \in \mathcal{L}_S$ such that ℓ progresses on $\iota(\ell)$ and all $\ell' \sqsubseteq \ell$ preserve $\iota(\ell)$. Hence, $\mathcal{P}$ satisfies the discharge condition from Definition 7 and is thus a proof. □

8.4 IMPLEMENTATION

Our implementation of cyclic proofs required a major overhaul of Tamarin. We changed around 200 files, inserted 61,000 lines of code, and deleted 24,000 lines of code. In this section, we describe this implementation.

When Tamarin searches for proofs, it constructs a proof tree by repeatedly executing *proof methods*, which typically apply sequences of constraint reduction rules. At each node of the proof tree, Tamarin can choose one from the many available proof methods to apply, and heuristics guide which proof methods will be applied automatically by Tamarin. In this section, we describe new proof methods and heuristics for cyclic induction, and how our implementation verifies that constructed pre-proofs are valid.

8.4.1 *Backlink Search*

Cyclic Proof Methods

We implemented three new proof methods: *search backlink*, *cut*, and *minimize for cyclic proofs*, the last of which combines weakening and cut. When used naively, cut and weakening considerably enlarge Tamarin's search space. The set of formulas that could be cut in is infinite, and theoretically, every single constraint could be weakened. Thus, we must restrict the application of these proof methods.

When applying the proof method *search backlink* to a constraint system Γ, our implementation searches for a substitution such that some constraint system on the path from Γ to the root subsumes Γ. Although we implemented several optimizations to the backlink search (see the following section), backlink search is an instance of the subgraph isomorphism problem, which is NP-complete [50]. Thus, we cannot exclude instances where backlink search is computationally expensive. We therefore implemented the backlink search as a proof method so that users or heuristics can avoid backlink search when it is too expensive.

When implementing the backlink search, we observed cases where we expected a backlink, but where "trivial" formula constraints were missing in the leaf constraint system. Whenever a backlink search finds such a "close" match, our implementation suggests cutting in missing constraints. Noteworthy, this is the only way for our implementation to suggest the *cut* proof method.

Finally, *minimize for cyclic proofs* applies weakening combined with cut to restore ordering information lost by weakening edge constraints. Weakening can be required to find backlinks (see Section 8.2.3). The proof method weakens all nodes reachable from the earliest node containing a loop fact and weakens nodes that provide premises of weakened nodes. This proof method implements a simple weakening heuristic, which we discuss further in Section 8.5.2. Implementing automated weakening has multiple benefits:

1. It enables finding some backlinks in the first place.
2. The resulting constraint systems are small, lowering the cost of backlink search.
3. Weakening can lead to termination rather than non-termination when no proof is found. As weakened constraints systems are small, they are more likely to become solved. Tamarin will report "unfinishable" for such constraint systems as they may not be counterexamples. This informs the user that automated methods to tackle loops failed. Without cyclic induction, failing to tackle a loop typically results in non-termination.

Backlink Search Algorithm

We optimized our implementation of backlink search between two constraint systems by exploiting that substitutions must progress. To ensure progress, a substitution associated to a backlink must map at least some nodes to nodes that occur earlier in the constraint system. We generalize this observation and implement backlink search as a *directed acyclic graph (DAG)-prefix* search. DAG nodes are node constraints and $f@i$ formula constraints. DAG edges are edge constraints and $i < j$ formula constraints. These DAGs capture many important properties of constraint systems and provide structure to guide the backlink search. The DAG-prefix search attempts to match the smallest (w.r.t. to the edge-relation) nodes in the smaller DAG to the smallest nodes in the larger DAG and iteratively refines the resulting substitution by attempting to match the children of already matched nodes with one another. Should the substitution at some point match the two DAGs, we check whether it also applies to remaining constraints.

To speed up the DAG-prefix search, we color the DAG nodes in such a way that (a) we can check in constant time whether two nodes have the same color, and (b) two nodes can be matched only if they have the same color. Concretely, we color nodes annotated with rule instances using their rule name and nodes not yet annotated with rule instances with the list of action facts present at that node.

Technically, this search is incomplete. For example, it might be necessary to map a smallest node in the one graph to some node in the middle of the other graph. However, as constraint systems are constructed incrementally, it is likely that we considered such mappings earlier during proof search and need not consider them again.

8.4.2 *Proof Search*

Heuristics

In this section, we describe how we adapted Tamarin's heuristics that rank proof methods during proof search. Tamarin supports two main heuristics: a general-purpose "smart" heuristic and an "injective" heuristic that is tailored for theories that heavily use loops. We amended both of these heuristics as follows.

Tamarin recognizes some proof methods as *looping* in that after applying them, the same proof method is typically available again. To avoid immediate non-termination, such proof methods will be applied in a round-robin fashion. We modified Tamarin's heuristics such that proof methods related to cyclic proofs are recognized as looping and prioritized like other looping proof methods.

The other heuristic we implemented is that Tamarin will search backlinks before minimizing for cyclic proofs. This ensures that weak-

ening does not preclude backlink formation. Beyond that, we implemented no further heuristics. This shows that cyclic proofs can successfully be implemented with little guidance, as will become clear when we present our case studies in Section 8.5.

Discharging Pre-Proofs

So far, we showed how we implemented proof methods for finding backlinks, and how we rank these proof methods to find cyclic pre-proofs. The final step of a cyclic proof is to check whether the pre-proof discharges, i.e., whether Definition 9 applies. Implementing Algorithm 1 directly proved to be challenging as Tamarin's code-base is recursion-oriented. We thus slightly modified Algorithm 1 as follows.

The proof search in Tamarin recursively associates nodes with *results*, which can be: a solution was found, the constraint system is unfinishable, contradictory, or has a backlink. Tamarin successively applies proof methods until it encounters constraint systems for which it can directly decide their result. These constraint systems become the pre-proof's leaves. Tamarin determines the inner nodes' results by combining their children's results. For example, the results solved and contradictory are combined to solved.

We modified Tamarin's proof search such that it recursively checks whether the pre-proof discharges. Our implementation inverts Algorithm 1. Observe that Algorithm 1 non-deterministically splits a set of SCCs into increasingly smaller SCSes to create a progress order. For every SCS, Algorithm 1 checks that it discharges. Instead of decomposing SCCs into SCSes, we compose SCSes to SCCs. Our implementation initialy stores each backlink in a singleton SCS, checks that this SCS discharges, and returns it. It then recursively composes SCSes into larger SCSs, eventually becoming SCCs, while maintaining the invariant that each SCS discharges. If at any point we fail to combine SCSes into a discharging SCS, we mark the proof as unfinishable.

There might be progress orders found by Algorithm 1 that our implementation does not find because our implementation traverses the pre-proof's SCSes following its tree structure, but Algorithm 1 searches non-deterministically. However, we never encountered proofs for which our implementation failed to find a progress order.

8.5 CASE STUDIES AND DISCUSSION

We evaluated cyclic proofs on fourteen case studies, which include two models of the Signal protocol. Our evaluation shows that cyclic induction consistently reduces the number of auxiliary lemmas required to prove a conjecture compared to trace induction. In fact, cyclic induction typically requires *no* auxiliary lemmas. Many properties that previously required auxiliary lemmas, can now be proven without

them. In particular for Signal, we observe that cyclic induction reduces model complexity, requiring fewer annotations, and simplifies proof search. There are only five exceptions. For two lemmas, cyclic induction requires the same auxiliary lemmas as trace induction. For one lemma, cyclic induction requires some, but fewer, auxiliary lemmas than trace induction. For another lemma, cyclic induction requires a different auxiliary lemma than trace induction. Finally, for one lemma, we were unable to construct a proof when using cyclic induction; however, we specifically engineered this lemma to be unprovable with cyclic induction. We discuss this further under limitations in Section 8.5.2.

We divide our case studies into three sets. The first set contains models that originate from Tamarin's standard examples and three theories developed by us to exhibit similar challenges as iMessage PQ3 (see Section 7.4.3). These models were developed to exhibit particular challenges of real-world looping protocols. The second set contains two models of Signal. The third set contains what we call *destructor-based theories*. We find that destructor-based theories are not well-suited for cyclic induction proofs and discuss this further in our limitations section.

We provide an overview of all case studies in Tables 8.1-8.3. We analyzed the exact same theories and lemmas per induction scheme, i.e., theories were not modified for specific schemes. Each table's first column shows to which set a theory belongs. Every set contains multiple theories (second column), which contain multiple lemmas (third column). We number lemmas for clarity. The columns grouped by "Provable..." show whether a given lemma is provable with: no induction ("w/o I"), trace induction ("w/ TI"), or cyclic induction ("w/ CI"). ✓ means that the lemma is provable. (X) means that the lemma is provable with the given scheme when using the referenced lemmas from the same theory as the auxiliary lemmas. For example, the lemma "Secrecy" from the "Crypto API" theory requires the lemma "Invariant" as an auxiliary lemma (cf. Table 8.1). Some lemmas are provable without induction but require inductive, auxiliary lemmas. Because such lemmas are also provable with induction, we show the auxiliary lemmas required to prove the conjecture with induction in gray. No symbol means that we were unable to find a proof with the respective scheme. In the two columns named "Auto," we mark whether Tamarin was able to automatically construct a proof when using no induction or trace induction and when using cyclic induction respectively. The two columns grouped by "Using..." show whether the proof methods cut ($\mathcal{S}_\Delta$) or minimize for cyclic proofs ($\mathcal{S}_W$) were used in a cyclic proof. "User" in the column for $\mathcal{S}_W$ marks that user-specified weakening was required (see Section 8.5.2).

Theory	Lemma	Provable…					Using…	
		w/o I	w/ TI	…auto	w/ CI	…auto	$\mathcal{S}_\Delta$	$\mathcal{S}_W$
Loop	(1) Start before Loop		✓	✓	✓	✓	✓	
	(2) Start before Stop	(1)	(1)	✓	✓	✓		✓
	(3) Loop before Stop		✓	✓	✓	✓		
	(4) Stop unique	(3)	(3)	✓	✓			✓
Hash Chain	(1) Loop Start		✓	✓	✓	✓	✓	
	(2) Loop First		✓	✓	✓	✓		
	(3) Loop Success Ord		✓	✓	✓	✓		
	(4) Loop+Success inv.		✓	✓	✓		✓	✓+ User
	(5) Loop+Success	(3)-(4)	(3)-(4)	✓	(4)		✓	User
	(6) Success inv.		✓	✓	✓	✓		✓
	(7) Success	(6)	(6)	✓	✓	✓		✓
Crypto API	(1) Invariant		✓	✓	✓			
	(2) Secrecy	(1)	(1)	✓	✓			✓
Key Renegotiation	Secrecy		✓	✓	✓			
Create, Use, Destroy	(1) Use		✓	✓	✓	✓		✓
	(2) Destroy		(1)	✓	✓		✓	✓
Alternating Loop	(1) Outer Loop Step		✓	✓	✓	✓	✓	
	(2) Fresh Seed		(1)	✓	✓	✓		✓
	(3) Seed Secrecy		(1)	✓	✓	✓		
	(4) Key Secrecy		✓	✓	✓	✓	✓	
Nested Loop	(1) Outer Loop Step		✓	✓	✓	✓	✓	
	(2) Fresh Seed		(1)	✓	✓	✓		✓
	(3) Seed Construction		(1)	✓	✓	✓		✓
	(4) Key Construction		✓	✓	✓	✓		
	(5) Key Secrecy	(1)-(4)	(1)-(4)	✓	✓	✓		✓
Revealing Loop	(1) Loop Start		✓	✓	✓	✓		
	(2) Seeds Match		✓	✓	✓	✓		
	(3) IDs Match		✓	✓	✓	✓		
	(4) Secrecy	(1)-(2)	(1)-(2)	✓	(3)		✓	
	(5) Secrecy Variant	(1)-(2)	(1)-(2)	✓	✓		✓	

Table 8.1: Example case studies to compare induction schemes. For a detailed explanation, see the beginning of Section 8.5.

Theory	Lemma	Provable…					Using…	
		w/o I	w/ TI	…auto	w/ CI	…auto	$\mathcal{S}_\Delta$	$\mathcal{S}_W$
Signal 1	(1) Outer Loop		✓	✓	✓	✓	✓	✓
	(2) Rk Secrecy		(1)		✓			✓
	(3) Ck Secrecy		✓	✓	✓	✓		
	(4) Secrecy	(2)-(3)	(2)-(3)	✓	✓			✓
Signal 2	(1) Outer Loop		✓	✓	✓	✓	✓	✓
	(2) Rk Secrecy	(1)	(1)		✓			✓
	(3) Ck Secrecy		✓	✓	✓	✓		
	(4) Secrecy	(2)-(3)	(2)-(3)	✓	✓			✓

Table 8.2: Signal case studies to compare induction schemes. For a detailed explanation, see the beginning of Section 8.5.

Theory	**Lemma**	**Provable…**					**Using…**	
		w/o I	**w/ TI**	**…auto**	**w/ CI**	**…auto**	$\mathcal{S}_\Delta$	$\mathcal{S}_W$
Up and Down	(1) Correctness Invariant		✓	✓	✓	✓	✓	✓
	(2) Correctness End		(1)	✓	✓	✓	✓	✓
	(3) Gen Start		✓	✓	✓	✓	✓	✓
	(4) Destr Seed		(3)	✓	✓	✓	✓	✓
	(5) Gen Unique		✓	✓	✓	✓	✓	✓
	(6) Correspondence		(1)+(5)	✓				
Loop Exits	(1) Correspondence		✓	✓	✓	✓	✓	✓
	(2) Correspondence Cut		✓	✓	✓		✓	✓
TESLA 1	Authentic		✓	✓	✓		✓	
TESLA 2	(1) Unique Keys		✓	✓	✓	✓	✓	
	(2) Key Expiry		(1)	✓	(1)		✓	
	(3) Authentic		(1)-(2)	✓	(1)-(2)		✓	

Table 8.3: Destructor-based case studies to compare induction schemes. For a detailed explanation, see the beginning of Section 8.5.

8.5.1 *The Signal Case Study*

Signal is the most widely used, end-to-end encrypted messaging protocol and uses a nested loop in its double ratchet construction. It is so complex that previous attempts to prove its security in the symbolic model either drastically abstracted the protocol or even developed entirely new tools to tackle it. See Chapters 6 and 7 for a more detailed introduction to double ratchet protocols.

To evaluate cyclic induction, we proved message secrecy for two models of Signal (1 and 2). In our models of Signal, we assume that clients use authentic long-term and pre-key material for session establishment. Clients exchange messages over an insecure network, and they derive encryption keys using the double-ratchet and X3DH [110, 121]. We fully modelled the looping behavior of the double-ratchet, without abstracting it in any way. However, we did not model any features of Signal beyond the double-ratchet and X3DH, and, in particular, did not model skipped messages. The two models differ in that Signal 1 only allows long-term key reveal, whereas Signal 2 also allows revealing pre-keys and ephemeral keys.

We formalize message secrecy for Signal as follows:

$$
\begin{aligned}
& \forall m, a, b, t.\mathsf{Send}(m, a, b)@t \\
\implies\ & \neg\exists x.\mathsf{K}(m)@x \\
& \vee \exists x.\mathsf{LtkReveal}(a)@x \vee \exists x.\mathsf{LtkReveal}(b)@x \\
& [\vee \exists x.\mathsf{RevealPre}(a)@x \vee \exists x.\mathsf{RevealPre}(b)@x \\
& \vee \exists x.\mathsf{RevealEph}(a, b)@x \vee \exists x.\mathsf{RevealEph}(b, a)@x].
\end{aligned}
$$

The part marked in square brackets applies to Signal 2 only. The lemma formalizes that a message m sent from participant a to participant b remains confidential unless one of the following key compromises occurs: (i) One participant's long-term key was revealed (LtkReveal), or (ii) pre-key material of one participant was revealed (RevealPre), or (iii) ephemeral key material from a session between a and b was revealed (RevealEph).

When using cyclic induction, we could prove message secrecy as formalized above for both Signal case studies without requiring any auxiliary lemmas. The proofs for the Signal case studies contain 55 backlinks for Signal 1 and 31 backlinks for Signal 2. Figure 8.6 depicts the proof graph of the Signal 2 case study and illustrates a complex proof structure. Nevertheless, Tamarin can find a proof automatically when supplied with a simple heuristic that instructs Tamarin to deprioritize solving equations that can lead to case splits. For example, such equations describe the structure of Diffie-Hellman shared secrets, and we observed that solving these equations leads to a state-space blowup during proof construction. With this simple heuristic (along with the heuristics presented in Section 8.4.2), Tamarin

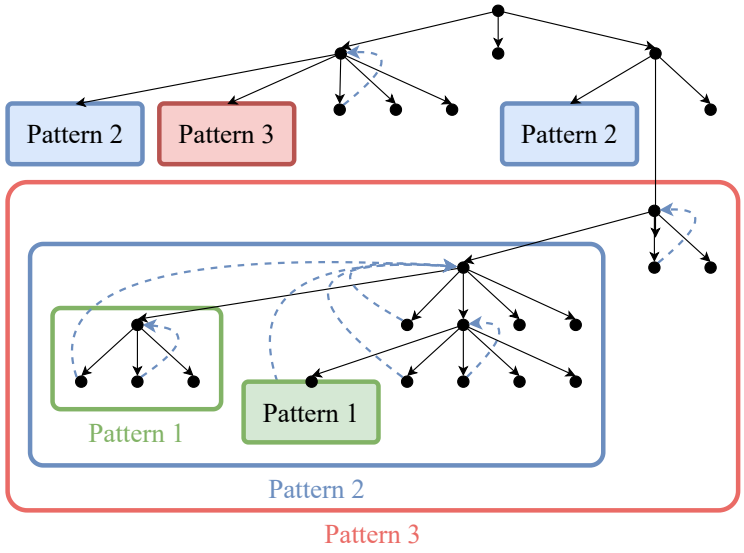


Figure 8.6: Proof graph of Secrecy for Signal 2. We abstract repeating patterns. The tree contains 31 backlinks.

finds a proof for each Signal case study in around 40 seconds, showing that cyclic induction can indeed be used to find complex, inductive proofs with little guidance.

For comparison, we also proved message secrecy using trace induction for both Signal case studies. In both cases, we needed to write three very similar auxiliary lemmas to prove message secrecy. Moreover, writing these lemmas required changing the model substantially, adding more annotations to help formalize the lemmas. Our Signal case study shows that cyclic induction drastically simplifies the analysis of protocols like Signal when compared to trace induction and that it is easy to find a cyclic proof for both Signal case studies.

8.5.2 *Limitations*

Destructor-based Models

During our case studies, we found that cyclic induction often does not improve, and sometimes even hinders, proof construction for models that are *destructor-based*. We call models destructor-based if they use not only looping constructor rules, but also looping destructor rules. Constructor rules use the terms in their premises to *construct* larger terms in their conclusions. In contrast, destructor rules *destruct* the terms in their premises, creating smaller terms in their conclusions. We encountered this limitation when proving the TESLA protocol, but illustrate it on the smaller model called "Up and down."

The "Up and down" model contains two loops. The first loop generates a seed x and applies a hash function h an arbitrary amount

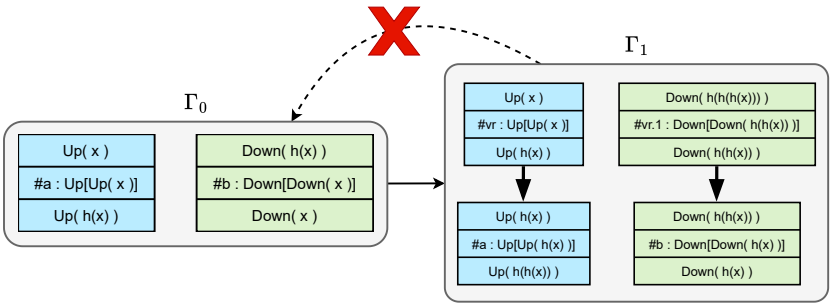


Figure 8.7: Illustration of diverging terms. Blue nodes are constructor rules, green nodes are destructor rules.

of times to that seed. The second loop starts when the first loop ends, and non-deterministically often removes the hash function. Thus, the first loop constructs a term $h^n(x)$, and the second loop destructs that term resulting in $h^m(x)$ $(m \leq n)$.

We call the problem of destructor-based models *diverging terms*. During proof construction for destructor-based theories, one will often encounter cases where both the construction and destruction loops are instantiated, as illustrated in Figure 8.7. One will generally be unable to find backlinks as the looping rules' terms diverge. Observe that the terms in the premises in Γ_0 in Figure 8.7 are x and $h(x)$. When solving for the constructor rule's premise, the term in the added premise rule remains x. However, when solving for the destructor rule's premise, the term in the added premise rule is $h(h(h(x)))$. Thus, to find a substitution matching the previous, smaller constraint system, we must both match x with x and $h(x)$ with $h(h(h(x)))$, i.e., x with $h(h(x))$, which is impossible.

While we were able to construct proofs using cyclic induction for destructor-based models, sometimes still reducing the number of auxiliary lemmas (see Tables 8.1-8.3), we found it easier to construct proofs for the TESLA case studies with trace induction than with cyclic induction. TESLA is a broadcast authentication protocol for which we analyze two variants. For TESLA, the auxiliary lemmas originally provided for trace induction helped mitigate the issue of diverging terms. However, finding backlinks when using these auxiliary lemmas requires care, e.g., weakening and backlink search must only be applied to a few, selected nodes in the derivation tree, and applying them elsewhere typically yields "unfinishable."

Heuristics

Our heuristics are simple (see Section 8.4.2), but likely not optimal. Excluding Signal and TESLA, all our case studies are automatically provable when using trace induction but not when using cyclic induction. This comparison, however, is unfair as many trace induction

proofs require writing auxiliary lemmas, which is an inherently manual task. With cyclic proofs, far fewer auxiliary lemmas are required. Thus, cyclic induction paves the way for future work on improved proof automation.

There are, however, exceptions, and proving some lemmas with cyclic induction required auxiliary lemmas or user-specified weakening. User-specified weakening means that weakening is required to find a proof, but using *minimize for cyclic proof* is insufficient because it would weaken either too much or too little. We leave it as future work to develop better heuristics for cyclic proofs.

Performance

Backlink search is NP-complete, and we thus cannot exclude that modelers encounter theories where proof search takes considerable time. In practice, however, we find that our implementation as presented in Section 8.4 is performant. We timed proof construction on a MacBook with an Apple M2 Max CPU and 32 GB of memory, and proving all lemmas of our simple case studies took no longer than 0.25 seconds per model. Even complex, real-world case studies such as Signal can be proven quickly. Proving each Signal case study took around 40 seconds, but 30 seconds were spent on loading the theory and only 10 seconds on constructing the proof. Proving TESLA took 4 seconds.

We find that cyclic induction takes on average 0.35 seconds or 2% respectively longer than trace induction per theory (2.4 seconds and 7% at maximum) with one exception. The exception is TESLA 1, which takes 1.7 seconds longer (69% increase). Comparing the verification times of cyclic and trace induction directly, however, is unfair as trace induction requires more auxiliary lemmas than cyclic induction. First, auxiliary lemmas must be conjectured by the user, and this process cannot be easily timed. Second, we included the verification time of auxiliary lemmas for cyclic induction. For example, if we only verify Secrecy for cyclic induction but all lemmas for trace induction, then the proof time for Signal 1 increases only by 0.29 seconds (1%) and the time for Signal 2 decreases by 2.37 seconds (-5%).

8.6 RELATED WORK

8.6.1 *Formal Analysis of Looping Protocols*

For a comparison with DY*, see Chapter 6. We discussed related works on the formal analysis of double ratchet protocols in the symbolic model in Section 7.5.1. As we pointed out there, previous works analyzing such protocols did either not capture their looping behavior or abstracted them. Our formal analysis of iMessage PQ3 captured all of PQ3's details, but this required substantial manual intervention and

stating 32 auxiliary lemmas (see Section 7.4.4). In contrast, our cyclic proofs for Signal require no auxiliary lemmas and only one, simple heuristic.

ProVerif's induction was used to verify election protocols [43] and protocols using authenticated data structures [44]. We discuss [44] in future work. The election protocol model in [43] includes two loops, one to fix the number of voters (non-deterministically counting up), and one to tally their votes (counting down). These loops resemble the "Up and down" model (see Section 8.5.2), for which we proved properties similar to those in [43].

ProVerif's induction is fairly new and there are few case studies using it. It remains to be seen whether ProVerif's induction scales to protocols like Signal, which we proved as part of our case studies with little effort. However, given that ProVerif's induction mechanism closely resembles Tamarin's trace induction, we expect that there are limitations similar to Tamarin's.

8.6.2 *Cyclic Proof Systems and Tools*

Proof Systems and Applications

Cyclic induction proof systems have been developed and used in diverse areas of logic and computer science. These include first-order logic with inductive predicates [32, 36], Peano arithmetic [143], Kleene algebras [56, 132], modal and first-order μ-calculi [6, 7, 141, 147], higher-order fixed point arithmetic [89], equational reasoning about functional programs [82], reasoning about process languages [55, 141], program logics [33, 134, 156], and program synthesis [79].

In many cases, cyclic proof systems are at least as powerful as systems based on explicit induction rules. An interesting theoretical question is whether they are equivalent. This has been settled in the positive for first-order μ-calculus [147] and Peano arithmetic [143] and in the negative for first-order logic with inductive predicates [17].

Tools for Cyclic Proofs

The Erlang Verification Tool was an early implementation of cyclic proofs for proving first-order μ-calculus properties of Erlang programs. It supports induction, co-induction, and their combination (alternating fixed points). Brotherston et al. [34] implemented a proof system for entailment proofs in separation logic using a deep embedding in HOL Light. Cyclist [35] is a generic stand-alone tool for cyclic proofs, which can be instantiated to different logics, e.g., for program termination in separation logic [134] and temporal properties of pointer programs [156]. CycleQ [82] is a tool for equational reasoning about functional programs. Cypress [79] uses cyclic reasoning for the deductive synthesis of heap-manipulating programs from separation

logic specifications. Our work is the first use of cyclic proof systems for security protocol verification. This required addressing several challenges, as discussed in the introduction.

8.7 CONCLUSION

We have introduced cyclic induction reasoning in Tamarin, proved its soundness, implemented it, and evaluated it on fourteen case studies, showing that it can be used to easily find proofs for complex protocols such as Signal. Cyclic induction exploits repeating patterns in Tamarin's constraints systems, and thereby avoids previous sources of non-termination during proof construction. Moreover, cyclic induction fundamentally changes how one constructs inductive proofs. Tamarin's previous induction scheme, trace induction, required writing inductive lemmas and could only be applied at the start of a proof. In contrast, cyclic induction enables Tamarin to automatically, and on-the-fly, discover inductive proofs. By finding repeating patterns in graphs, cyclic induction avoids the need for auxiliary lemmas in many practically relevant case studies. In contrast to writing auxiliary lemmas, finding repeating graph patterns is much better suited for automation.

Although our work is based on Tamarin, the ideas are, in principle, transferable to other tools such as ProVerif. As ProVerif's induction is fairly new, there are few case studies using it. However, given that ProVerif's induction mechanism closely resembles Tamarin's trace induction, we expect that there are limitations similar to Tamarin's.

FUTURE WORK We believe that cyclic induction can benefit from further optimized heuristics and algorithms for constructing cyclic proofs automatically. There are many promising options here. For example, there is an enormous body of work on the problem of *subgraph matching* (e.g., [31, 152, 166]). Applying results from this field could further improve our implementation of backlink search. Additionally, one could explore better search strategies. For example, backtracking when negated branches of a cut do not lead to a contradiction, automated weakening of formulas that are not required to derive contradictions in base cases, and much more. Another interesting line of work is exploring whether cyclic induction can simplify proving properties of protocols using authenticated data structures such as Merkle Hash Trees in the symbolic model, which was initially suggested by [44]. Finally, our cyclic induction framework does not yet apply to Tamarin's mode for proving observational equivalence [13]. Investigating whether it does also remains as future work.

9 CONCLUSION

In this thesis, we presented how to achieve provable, system-wide security guarantees and advanced the state-of-the-art of protocol verification in the symbolic model.

In Part i, we presented two novel systems that use social authentication and accountability respectively to provably provide long-term key authentication guarantees. We formalized social authentication in Chapter 4, and proved that SOAP, a SOcial Authentication Protocol, provides it. Moreover, SOAP is a practical design, which we showcased with two functional prototypes. In Chapter 5, we presented and formally analyzed ADEM, an Authentic Digital EMblem, which provides authentication guarantees by relying on an accountability mechanism. ADEM implements a digital emblem to mark digital infrastructure as protected under IHL, analogous to the emblems of the Red Cross, Red Crescent, and Red Crystal.

In Part ii, we showed how complex, looping protocols can be formally analyzed in their full complexity when using trace and cyclic induction in the Tamarin prover. In Chapter 7, we formally proved iMessage PQ3 secure using trace induction. We showed that PQ3 provides complex and fine-grained security guarantees against a powerful adversary with quantum-computing capabilities. In Chapter 8, we adapted cyclic induction to the security protocol domain. We showed how cyclic induction can be used to prove protocols like Signal secure while requiring considerably less effort than when using trace induction. With cyclic induction, Tamarin now can prove many properties that previously required complex auxiliary conjectures.

OUTLOOK SOAP and ADEM are both great examples for when the symbolic model has its advantages over the computational setting. For one, we proved properties, such as social authentication, that are difficult to faithfully capture in the computational setting. For the other, we analyzed both systems during their development and the aid of automated tools such as Tamarin made this possible in the first place. Machine-assisted proof search enables quick iteration of designs and desired security guarantees, helping protocol designers explore both their problem and their solution space.

We hope that our contributions encourage the analysis of more protocols during their design. We see two more directions that could be explored to help reach this goal. First, automated symbolic provers would benefit from better tooling for developing protocol models. Current automated provers do not support modularization well, which in

turn hinders incremental modeling, e.g. by assuming secure channels at first and replacing them with more accurate models later. Modularization likely requires support for composing different protocols. There is a line of research establishing compositionality results (e.g., [47, 71, 72, 78]), however, these results have not yet been implemented in state-of-the-art protocol verification tools. Additionally, protocol models are often hard to test. Typically, modelers prove executability lemmas, which show that a protocol model admits traces without the adversary being active. However, we experienced that these lemmas are sometimes harder to prove than actual security properties, and they can easily miss modeling flaws.

Second, protocol models could be better integrated into the development lifecycle of the protocol itself. There is research on connecting protocol implementations to protocol analysis, but proposed approaches either require considerable manual effort [11, 148] or focus on proving that a specific implementation provides certain, protocol-level security guarantees [10, 21]. We suggest exploring how protocol models created during protocol development and therefore before any implementation work started can provide more value. For example, they could serve as reference specifications or be used to generate test vectors.

BIBLIOGRAPHY

[1] Apple Security Engineering and Architecture (SEAR). *Advancing iMessage Security: iMessage Contact Key Verification*. Apple Security Research. Oct. 27, 2023. URL: https://security.apple.com/blog/imessage-contact-key-verification/ (visited on 01/12/2024).

[2] Apple Security Engineering and Architecture (SEAR). *iMessage with PQ3: The New State of the Art in Quantum-Secure Messaging at Scale*. Apple Security Research. Feb. 21, 2024. URL: https://security.apple.com/blog/imessage-pq3/ (visited on 05/27/2024).

[3] Josh Aas, Richard Barnes, Benton Case, Zakir Durumeric, Peter Eckersley, Alan Flores-López, J. Alex Halderman, Jacob Hoffman-Andrews, James Kasten, Eric Rescorla, Seth Schoen, and Brad Warren. "Let's Encrypt: An Automated Certificate Authority to Encrypt the Entire Web". In: Conference on Computer and Communications Security (CCS). New York, NY, USA, Nov. 6, 2019. DOI: 10.1145/3319535.3363192.

[4] *About End-to-End Encryption*. WhatsApp Help Center. URL: https://faq.whatsapp.com/820124435853543?helpref=faq_content (visited on 05/05/2025).

[5] *ADEM Specification*. Version v1.0. Aug. 30, 2023. URL: https://github.com/adem-wg/adem-spec/releases/tag/v1.0 (visited on 04/16/2025).

[6] Bahareh Afshari, Sebastian Enqvist, and Graham E Leigh. "Cyclic proofs for the first-order μ-calculus". In: *Logic Journal of the IGPL* 32.1 (Jan. 25, 2024). ISSN: 1367-0751. DOI: 10.1093/jigpal/jzac053.

[7] Bahareh Afshari and Graham E. Leigh. "Cut-Free Completeness for Modal Mu-Calculus". In: *32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. Reykjavik, Iceland, June 2017. DOI: 10.1109/LICS.2017.8005088.

[8] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. "The Double Ratchet: Security Notions, Proofs, and Modularization for the Signal Protocol". In: *Advances in Cryptology – EUROCRYPT 2019*. Cham, 2019. DOI: 10.1007/978-3-030-17653-2_5.

[9] *Android Intents with Chrome*. Chrome Developers. URL: https://developer.chrome.com/docs/multidevice/android/intents/ (visited on 10/03/2022).

[10] Linard Arquint, Malte Schwerhoff, Vaibhav Mehta, and Peter Müller. "A Generic Methodology for the Modular Verification of Security Protocol Implementations". In: Conference on Computer and Communications Security (CCS). New York, NY, USA, Nov. 21, 2023. DOI: 10.1145/3576915.3623105.

[11] Linard Arquint, Felix A. Wolf, Joseph Lallemand, Ralf Sasse, Christoph Sprenger, Sven N. Wiesner, David Basin, and Peter Müller. "Sound Verification of Security Protocols: From Design to Interoperable Implementations". In: Symposium on Security and Privacy (S&P). May 2023. DOI: 10.1109/SP46215.2023.10179325.

[12] Richard Barnes, Jacob Hoffman-Andrews, Daniel McCarney, and James Kasten. *Automatic Certificate Management Environment (ACME)*. Request for Comments RFC 8555. Internet Engineering Task Force, Mar. 2019. 95 pp. DOI: 10.17487/RFC8555.

[13] David Basin, Jannik Dreier, and Ralf Sasse. "Automated Symbolic Proofs of Observational Equivalence". In: Conference on Computer and Communications Security (CCS). New York, NY, USA, Oct. 12, 2015. DOI: 10.1145/2810103.2813662.

[14] David Basin, Ralf Sasse, and Jorge Toro-Pozo. "The EMV Standard: Break, Fix, Verify". In: 2021 IEEE Symposium on Security and Privacy (S&P). May 2021. DOI: 10.1109/SP40001.2021.00037.

[15] David Basin, Patrick Schaller, and Jorge Toro-Pozo. "Inducing Authentication Failures to Bypass Credit Card PINs". In: 32nd USENIX Security Symposium. Anaheim, CA, USA, 2023. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/basin (visited on 05/11/2025).

[16] Mihir Bellare, Asha Camper Singh, Joseph Jaeger, Maya Nyayapati, and Igors Stepanovs. "Ratcheted Encryption and Key Exchange: The Security of Messaging". In: *Advances in Cryptology – CRYPTO 2017*. Cham, 2017. DOI: 10.1007/978-3-319-63697-9_21.

[17] Stefano Berardi and Makoto Tatsuta. "Classical System of Martin-Lof's Inductive Definitions Is Not Equivalent to Cyclic Proofs". In: *Logical Methods in Computer Science* Volume 15, Issue 3 (Aug. 1, 2019). ISSN: 1860-5974. DOI: 10.23638/LMCS-15(3:10)2019.

[18] Tim Berners-Lee, Roy T. Fielding, and Larry M. Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. Request for Comments RFC 3986. Internet Engineering Task Force, Jan. 2005. 61 pp. DOI: 10.17487/RFC3986.

[19] Karthikeyan Bhargavan, Abhishek Bichhawat, Quoc Do, Pedram Hosseyni, Ralf Küsters, Guido Schmitz, and Tim Würtele. "DY* : A Modular Symbolic Verification Framework for Executable Cryptographic Protocol Code". In: 6th European Symposium on Security and Privacy (EuroS&P). Sept. 6, 2021. DOI: 10.1109/EuroSP51992.2021.00042.

[20] Karthikeyan Bhargavan, Abhishek Bichhawat, Quoc Huy Do, Pedram Hosseyni, Ralf Küsters, Guido Schmitz, and Tim Würtele. "An In-Depth Symbolic Security Analysis of the ACME Standard". In: Conference on Computer and Communications Security (CCS). New York, NY, USA, Nov. 13, 2021. DOI: 10.1145/3460120.3484588.

[21] Karthikeyan Bhargavan, Maxime Buyse, Lucas Franceschino, Lasse Letager Hansen, Franziskus Kiefer, Jonas Schneider-Bensch, and Bas Spitters. *Hax: Verifying Security-Critical Rust Software Using Multiple Provers*. Mar. 17, 2025. URL: https://eprint.iacr.org/2025/142. Pre-published.

[22] Karthikeyan Bhargavan, Charlie Jacomme, Franziskus Kiefer, and Rolfe Schmidt. "Formal Verification of the PQXDH Post-Quantum Key Agreement Protocol for End-to-End Secure Messaging". In: 33rd USENIX Security Symposium. 2024. URL: https://www.usenix.org/conference/usenixsecurity24/presentation/bhargavan.

[23] Alexander Bienstock, Jaiden Fairoze, Sanjam Garg, Pratyay Mukherjee, and Srinivasan Raghuraman. "A More Complete Analysis of the Signal Double Ratchet Algorithm". In: *Advances in Cryptology – CRYPTO 2022*. Cham, 2022. DOI: 10.1007/978-3-031-15802-5_27.

[24] B. Blanchet. "A Computationally Sound Mechanized Prover for Security Protocols". In: 2006 IEEE Symposium on Security and Privacy (S&P). May 2006. DOI: 10.1109/SP.2006.1.

[25] Bruno Blanchet. "Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif". In: *Foundations and Trends® in Privacy and Security* 1.1–2 (Oct. 30, 2016). ISSN: 2474-1558, 2474-1566. DOI: 10.1561/3300000004.

[26] Bruno Blanchet, Vincent Cheval, and Véronique Cortier. "ProVerif with Lemmas, Induction, Fast Subsumption, and Much More". In: Symposium on Security and Privacy (S&P). May 2022. DOI: 10.1109/SP46214.2022.9833653.

[27] Olivier Blazy, Ioana Boureanu, Pascal Lafourcade, Cristina Onete, and Léo Robert. "How Fast Do You Heal? A Taxonomy for Post-Compromise Security in Secure-Channel Establishment". In: 32nd USENIX Security Symposium. 2023. URL:

https://www.usenix.org/conference/usenixsecurity23/presentation/blazy.

[28] Josh Blum, Simon Booth, Brian Chen, Oded Gal, Maxwell Krohn, Julia Len, Karan Lyons, Antonio Marcedone, Mike Maxim, Marry Ember Mou, Armin Namavari, Jack O'Connor, Surya Rien, Miles Steele, Matthew Green, Lea Kissner, and Alex Stamos. *Zoom Cryptography Whitepaper*. Zoom Video Communications, Inc., Nov. 21, 2023. URL: https://github.com/zoom/zoom-e2e-whitepaper/blob/v4.3/zoom_e2e.pdf (visited on 12/05/2023).

[29] Sharon Boeyen, Stefan Santesson, Tim Polk, Russ Housley, Stephen Farrell, and David Cooper. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. Request for Comments RFC 5280. Internet Engineering Task Force, May 2008. 151 pp. DOI: 10.17487/RFC5280.

[30] Nikita Borisov, Ian Goldberg, and Eric Brewer. "Off-the-Record Communication, or, Why Not to Use PGP". In: *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*. New York, NY, USA, Oct. 28, 2004. DOI: 10.1145/1029179.1029200.

[31] Sarra Bouhenni, Saïd Yahiaoui, Nadia Nouali-Taboudjemat, and Hamamache Kheddouci. "A Survey on Distributed Graph Pattern Matching in Massive Graphs". In: *ACM Comput. Surv.* 54.2 (Feb. 9, 2021). ISSN: 0360-0300. DOI: 10.1145/3439724.

[32] James Brotherston. "Cyclic Proofs for First-Order Logic with Inductive Definitions". In: *Automated Reasoning with Analytic Tableaux and Related Methods*. Berlin, Heidelberg, 2005. DOI: 10.1007/11554554_8.

[33] James Brotherston, Richard Bornat, and Cristiano Calcagno. "Cyclic Proofs of Program Termination in Separation Logic". In: *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL. New York, NY, USA, Jan. 7, 2008. DOI: 10.1145/1328438.1328453.

[34] James Brotherston, Dino Distefano, and Rasmus Lerchedahl Petersen. "Automated Cyclic Entailment Proofs in Separation Logic". In: *Automated Deduction – CADE-23*. Berlin, Heidelberg, 2011. DOI: 10.1007/978-3-642-22438-6_12.

[35] James Brotherston, Nikos Gorogiannis, and Rasmus L. Petersen. "A Generic Cyclic Theorem Prover". In: *Programming Languages and Systems*. Berlin, Heidelberg, 2012. DOI: 10.1007/978-3-642-35182-2_25.

[36] James Brotherston and Alex Simpson. "Sequent Calculi for Induction and Infinite Descent". In: *Journal of Logic and Computation* 21.6 (Dec. 1, 2011). ISSN: 0955-792X. DOI: 10.1093/logcom/exq052.

[37] Alan Bundy. "Chapter 13 - The Automation of Proof by Mathematical Induction". In: *Handbook of Automated Reasoning*. Amsterdam, Jan. 1, 2001. DOI: 10.1016/B978-044450813-3/50015-1.

[38] Alan Bundy, David Basin, Dieter Hutter, and Andrew Ireland. *Rippling: Meta-Level Guidance for Mathematical Reasoning*. Cambridge, 2005. DOI: 10.1017/CBO9780511543326.

[39] Ran Canetti, Palak Jain, Marika Swanberg, and Mayank Varia. "Universally Composable End-to-End Secure Messaging". In: *Advances in Cryptology – CRYPTO 2022*. Cham, 2022. DOI: 10.1007/978-3-031-15979-4_1.

[40] Germano Caronni. "Walking the Web of Trust". In: *Proceedings IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*. WET ICE 2000. Gaithersburg, MD, USA, June 2000. DOI: 10.1109/ENABL.2000.883720.

[41] Melissa Chase, Apoorvaa Deshpande, Esha Ghosh, and Harjasleen Malvai. "SEEMless: Secure End-to-End Encrypted Messaging with Less Trust". In: Conference on Computer and Communications Security (CCS). London, UK, Nov. 11–15, 2019. DOI: 10.1145/3319535.3363202.

[42] *Check Your Keys for End-to-End Encrypted Chats on Messenger*. Messenger Help Center. URL: https://www.facebook.com/help/messenger-app/147596532316790 (visited on 05/05/2025).

[43] Vincent Cheval, Véronique Cortier, and Alexandre Debant. "Election Verifiability with ProVerif". In: 36th Computer Security Foundations Symposium (CSF). July 2023. DOI: 10.1109/CSF57540.2023.00032.

[44] Vincent Cheval, José Moreira, and Mark Ryan. "Automatic Verification of Transparency Protocols". In: 8th European Symposium on Security and Privacy (EuroS&P). July 2023. DOI: 10.1109/EuroSP57164.2023.00016.

[45] Chromium. *CRLSets*. URL: https://chromium.googlesource.com/playground/chromium-org-site/+/refs/heads/main/Home/chromium-security/crlsets.md (visited on 08/18/2023).

[46] Laurent Chuat, AbdelRahman Abdou, Ralf Sasse, Christoph Sprenger, David Basin, and Adrian Perrig. "SoK: Delegation and Revocation, the Missing Links in the Web's Chain of Trust". In: 2020 IEEE European Symposium on Security and Privacy (EuroS&P). Genoa, Italy, Sept. 2020. DOI: 10.1109/EuroSP48549.2020.00046.

[47] Stefan Ciobâca and Véronique Cortier. "Protocol Composition for Arbitrary Primitives". In: 23rd Computer Security Foundations Symposium (CSF). July 2010. DOI: 10.1109/CSF.2010.29.

[48] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. "A Formal Security Analysis of the Signal Messaging Protocol". In: 2017 IEEE European Symposium on Security and Privacy (EuroS&P). Apr. 2017. DOI: 10.1109/EuroSP.2017.27.

[49] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. "On Post-compromise Security". In: *29th Computer Security Foundations Symposium (CSF)*. June 2016. DOI: 10.1109/CSF.2016.19.

[50] Stephen A. Cook. "The Complexity of Theorem-Proving Procedures". In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. New York, NY, USA, May 3, 1971. DOI: 10.1145/800157.805047.

[51] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. "A Comprehensive Symbolic Analysis of TLS 1.3". In: Conference on Computer and Communications Security (CCS). New York, NY, USA, Oct. 30, 2017. DOI: 10.1145/3133956.3134063.

[52] Cas Cremers, Marko Horvat, Sam Scott, and Thyla van der Merwe. "Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication". In: 2016 IEEE Symposium on Security and Privacy (S&P). May 2016. DOI: 10.1109/SP.2016.35.

[53] Cas Cremers, Charlie Jacomme, and Philip Lukert. "Subterm-Based Proof Techniques for Improving the Automation and Scope of Security Protocol Analysis". In: *36th Computer Security Foundations Symposium (CSF)*. July 2023. DOI: 10.1109/CSF57540.2023.00001.

[54] Cas Cremers, Charlie Jacomme, and Aurora Naska. "Formal Analysis of Session-Handling in Secure Messaging: Lifting Security from Sessions to Conversations". In: 32nd USENIX Security Symposium. 2023. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/cremers-session-handling.

[55] Mads Dam and Dilian Gurov. "μ-Calculus with Explicit Points and Approximations". In: *Journal of Logic and Computation* 12.2 (Apr. 1, 2002). ISSN: 0955-792X. DOI: 10.1093/logcom/12.2.255.

[56] Anupam Das and Damien Pous. "A Cut-Free Cyclic Proof System for Kleene Algebra". In: *Automated Reasoning with Analytic Tableaux and Related Methods*. Cham, 2017. DOI: 10.1007/978-3-319-66902-1_16.

[57] *Digital Emblems (Diem)*. IETF Datatracker. URL: https://datatracker.ietf.org/group/diem/about/ (visited on 05/16/2025).

[58] Yevgeniy Dodis, Daniel Jost, Shuichi Katsumata, Thomas Prest, and Rolfe Schmidt. *Triple Ratchet: A Bandwidth Efficient Hybrid-Secure Signal Protocol*. Mar. 13, 2025. URL: https://eprint.iacr.org/2025/078. Pre-published.

[59] Jannik Dreier, Lucca Hirschi, Sasa Radomirovic, and Ralf Sasse. "Automated Unbounded Verification of Stateful Cryptographic Protocols with Exclusive OR". In: *31st Computer Security Foundations Symposium (CSF)*. July 2018. DOI: 10.1109/CSF.2018.00033.

[60] Paul Hankes Drielsma, Sebastian Mödersheim, Luca Viganò, and David Basin. "Formalizing and Analyzing Sender Invariance". In: *Proceedings of the 4th International Conference on Formal Aspects in Security and Trust*. Berlin, Heidelberg, Aug. 26, 2006.

[61] *Enable Apps for Websites Using App URI Handlers*. URL: https://docs.microsoft.com/en-us/windows/uwp/launch-resume/web-to-app-linking (visited on 07/15/2022).

[62] Ross Evans, Matthew McKague, and Douglas Stebila. *ProofFrog: A Tool For Verifying Game-Hopping Proofs*. Mar. 5, 2025. Cryptology ePrint Archive: 2025/418. URL: https://eprint.iacr.org/2025/418. Pre-published.

[63] Joan Feigenbaum, Aaron Johnson, and Paul Syverson. "Probabilistic Analysis of Onion Routing in a Black-Box Model". In: *ACM Transactions on Information and System Security* 15.3 (Nov. 30, 2012). ISSN: 1094-9224. DOI: 10.1145/2382448.2382452.

[64] Daniel Fett, Pedram Hosseyni, and Ralf Küsters. "An Extensive Formal Security Analysis of the OpenID Financial-Grade API". In: Symposium on Security and Privacy (S&P). May 2019. DOI: 10.1109/SP.2019.00067.

[65] Daniel Fett, Ralf Küsters, and Guido Schmitz. "A Comprehensive Formal Security Analysis of OAuth 2.0". In: Conference on Computer and Communications Security (CCS). New York, NY, USA, Oct. 24, 2016. DOI: 10.1145/2976749.2978385.

[66] Daniel Fett, Ralf Küsters, and Guido Schmitz. "An Expressive Model for the Web Infrastructure: Definition and Application to the Browser ID SSO System". In: Symposium on Security and Privacy (S&P). May 2014. DOI: 10.1109/SP.2014.49.

[67] Daniel Fett, Ralf Küsters, and Guido Schmitz. "The Web SSO Standard OpenID Connect: In-depth Formal Security Analysis and Security Guidelines". In: *30th Computer Security Foundations Symposium (CSF)*. Aug. 2017. DOI: 10.1109/CSF.2017.20.

[68] *Formal Proofs for ADEM*. Version v1.6. Apr. 16, 2025. URL: https://github.com/adem-wg/adem-proofs/releases/tag/v1.6 (visited on 04/16/2025).

[69] *Formal Proofs for SOAP*. Jan. 17, 2024. URL: https://github.com/soap-wg/soap-proofs/releases/tag/usenix (visited on 04/16/2025).

[70] Lorenzo Franceschi-Bicchierai. *How a Third-Party SMS Service Was Used to Take Over Signal Accounts*. Vice. Aug. 17, 2022. URL: https://www.vice.com/en/article/qjkvxv/how-a-third-party-sms-service-was-used-to-take-over-signal-accounts (visited on 08/30/2022).

[71] Sébastien Gondron and Sebastian Mödersheim. "Vertical Composition and Sound Payload Abstraction for Stateful Protocols". In: 34th Computer Security Foundations Symposium (CSF). June 2021. DOI: 10.1109/CSF51468.2021.00038.

[72] Thomas Groß and Sebastian Modersheim. "Vertical Protocol Composition". In: 24th Computer Security Foundations Symposium (CSF). June 2011. DOI: 10.1109/CSF.2011.23.

[73] Sven Hammann, Ralf Sasse, and David Basin. "Privacy-Preserving OpenID Connect". In: Asia Conference on Computer and Communications Security (ASIACCS). New York, NY, USA, Oct. 5, 2020. DOI: 10.1145/3320269.3384724.

[74] *Handling Android App Links*. Android Developers. URL: https://developer.android.com/training/app-links (visited on 07/15/2022).

[75] Dick Hardt. *The OAuth 2.0 Authorization Framework*. Request for Comments RFC 6749. Internet Engineering Task Force, Oct. 2012. 76 pp. DOI: 10.17487/RFC6749.

[76] Maximilian Häring, Julia Angelika Grohs, Eva Tiefenau, Matthew Smith, and Christian Tiefenau. "Can Johnny Be a Whistleblower? A Qualitative User Study of a Social Authentication Signal Extension in an Adversarial Scenario". In: Twentieth Symposium on Usable Privacy and Security (SOUPS). Aug. 12, 2024. URL: https://www.usenix.org/conference/soups2024/presentation/haring.

[77] Amir Herzberg and Hemi Leibowitz. "Can Johnny Finally Encrypt? Evaluating E2E-encryption in Popular IM Applications". In: *Proceedings of the 6th Workshop on Socio-Technical Aspects in Security and Trust*. New York, NY, USA, Dec. 5, 2016. DOI: 10.1145/3046055.3046059.

[78] Andreas V. Hess, Sebastian A. Mödersheim, and Achim D. Brucker. "Stateful Protocol Composition". In: *Computer Security*. Cham, 2018. DOI: 10.1007/978-3-319-99073-6_21.

[79] Shachar Itzhaky, Hila Peleg, Nadia Polikarpova, Reuben N. S. Rowe, and Ilya Sergey. "Cyclic Program Synthesis". In: *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. New York, NY, USA, June 18, 2021. DOI: 10.1145/3453483.3454087.

[80] Dennis Jackson, Cas Cremers, Katriel Cohn-Gordon, and Ralf Sasse. "Seems Legit: Automated Analysis of Subtle Attacks on Protocols That Use Signatures". In: Conference on Computer and Communications Security (CCS). New York, NY, USA, Nov. 6, 2019. DOI: 10.1145/3319535.3339813.

[81] Frederic Jacobs. "Invited Talk: Designing iMessage PQ3: Quantum-Secure Messaging at Scale". RWC 2024 (Toronto, Canada). Mar. 26, 2024. URL: https://www.youtube.com/watch?v=RVbHElGe518 (visited on 05/29/2024).

[82] Eddie Jones, C.-H. Luke Ong, and Steven Ramsay. "CycleQ: An Efficient Basis for Cyclic Equational Reasoning". In: *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. New York, NY, USA, June 9, 2022. DOI: 10.1145/3519939.3523731.

[83] Michael Jones, John Bradley, and Nat Sakimura. *JSON Web Signature (JWS)*. Request for Comments RFC 7515. Internet Engineering Task Force, May 2015. 59 pp. DOI: 10.17487/RFC7515.

[84] Roger Piqueras Jover. "Security Analysis of SMS as a Second Factor of Authentication: The Challenges of Multifactor Authentication Based on SMS, Including Cellular Security Deficiencies, SS7 Exploits, and SIM Swapping". In: *Queue* 18.4 (Aug. 31, 2020). ISSN: 1542-7730. DOI: 10.1145/3424302.3425909.

[85] Georgios Kalogridis, Costas Efthymiou, Stojan Z. Denic, Tim A. Lewis, and Rafael Cepeda. "Privacy for Smart Meters: Towards Undetectable Appliance Load Signatures". In: *2010 First IEEE International Conference on Smart Grid Communications*. SmartGridComm. Gaithersburg, MD, USA, Oct. 2010. DOI: 10.1109/SMARTGRID.2010.5622047.

[86] *Keybase*. URL: https://keybase.io/ (visited on 07/18/2022).

[87] *Keybase Book: Learn about Your Keybase Account*. URL: https://book.keybase.io/account#proofs (visited on 07/18/2022).

[88] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. "Automated Verification for Secure Messaging Protocols and Their Implementations: A Symbolic and Computational Approach". In: 2017 IEEE European Symposium on Security and Privacy (EuroS&P). Apr. 2017. DOI: 10.1109/EuroSP.2017.38.

[89] Mayuko Kori, Takeshi Tsukada, and Naoki Kobayashi. "A Cyclic Proof System for HFL_ℕ". In: *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*. Vol. 183. Dagstuhl, Germany, 2021. DOI: 10.4230/LIPIcs.CSL.2021.29.

[90] Hugo Krawczyk. "Cryptographic Extraction and Key Derivation: The HKDF Scheme". In: *Advances in Cryptology – CRYPTO 2010*. Berlin, Heidelberg, 2010. DOI: 10.1007/978-3-642-14623-7_34.

[91] Ehren Kret and Rolfe Schmidt. *The PQXDH Key Agreement Protocol*. Revision 3. May 24, 2023. URL: https://signal.org/docs/specifications/pqxdh/pqxdh.pdf.

[92] Robert Künnemann, Ilkan Esiyok, and Michael Backes. "Automated Verification of Accountability in Security Protocols". In: *32nd Computer Security Foundations Symposium (CSF)*. June 2019. DOI: 10.1109/CSF.2019.00034.

[93] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. "Accountability: Definition and Relationship to Verifiability". In: Conference on Computer and Communications Security (CCS). New York, NY, USA, Oct. 4, 2010. DOI: 10.1145/1866307.1866366.

[94] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine Generals Problem". In: *ACM Transactions on Programming Languages and Systems* 4.3 (July 1, 1982). ISSN: 0164-0925. DOI: 10.1145/357172.357176.

[95] Ben Laurie. "Certificate Transparency: Public, Verifiable, Append-Only Logs". In: *Queue* 12.8 (Aug. 1, 2014). ISSN: 1542-7730. DOI: 10.1145/2668152.2668154.

[96] Ben Laurie, Adam Langley, and Emilia Kasper. *Certificate Transparency*. Request for Comments RFC 6962. Internet Engineering Task Force, June 2013. 27 pp. DOI: 10.17487/RFC6962.

[97] Ben Laurie, Adam Langley, Emilia Kasper, Eran Messeri, and Rob Stradling. *Certificate Transparency Version 2.0*. Request for Comments RFC 9162. Internet Engineering Task Force, Dec. 2021. 53 pp. DOI: 10.17487/RFC9162.

[98] Seonwoo Lee, Robert J. Baxley, Mary Ann Weitnauer, and Brett Walkenhorst. "Achieving Undetectable Communication". In: *IEEE Journal of Selected Topics in Signal Processing* 9.7 (Oct. 2015). ISSN: 1941-0484. DOI: 10.1109/JSTSP.2015.2421477.

[99] Julia Len, Melissa Chase, Esha Ghosh, Daniel Jost, Balachandar Kesavan, and Antonio Marcedone. "ELEKTRA: Efficient Lightweight Multi-dEvice Key TRAnsparency". In: Conference on Computer and Communications Security (CCS). New York, NY, USA, Nov. 21, 2023. DOI: 10.1145/3576915.3623161.

[100] Julia Len, Melissa Chase, Esha Ghosh, Kim Laine, and Radames Cruz Moreno. "OPTIKS: An Optimized Key Transparency System". In: 33rd USENIX Security Symposium. 2024. URL: https://www.usenix.org/conference/usenixsecurity24/presentation/len.

[101] Sean Lawlor Lewi Kevin. *Deploying Key Transparency at WhatsApp*. Engineering at Meta. Apr. 13, 2023. URL: https://engineering.fb.com/2023/04/13/security/whatsapp-key-transparency/ (visited on 04/24/2023).

[102] Felix Linker, Ralf Sasse, and David Basin. *A Formal Analysis of Apple's iMessage PQ3 Protocol*. Zenodo, Jan. 21, 2025. DOI: 10.5281/zenodo.14710687.

[103] Felix Linker, Christoph Sprenger, Cas Cremers, and David Basin. *Looping for Good: Cyclic Proofs for Security Protocols*. Apr. 11, 2025. DOI: 10.5281/zenodo.15183936.

[104] Mihael Liskij, Xuhua Ding, Gene Tsudik, and David Basin. "Oblivious Digital Tokens". In: 34th USENIX Security Symposium. 2025. URL: https://www.usenix.org/conference/usenixsecurity25/presentation/liskij.

[105] Torsten Lodderstedt, John Bradley, Aney Labunets, and Daniel Fett. *OAuth 2.0 Security Best Current Practice*. Internet Draft draft-ietf-oauth-security-topics-19. Internet Engineering Task Force, Dec. 16, 2021. 52 pp. URL: https://www.ietf.org/archive/id/draft-ietf-oauth-security-topics-24.html (visited on 12/22/2023).

[106] Torsten Lodderstedt, Mark McGloin, and Phil Hunt. *OAuth 2.0 Threat Model and Security Considerations*. Request for Comments RFC 6819. Internet Engineering Task Force, Jan. 2013. 71 pp. DOI: 10.17487/RFC6819.

[107] G. Lowe. "A Hierarchy of Authentication Specifications". In: *Proceedings 10th Computer Security Foundations Workshop*. June 1997. DOI: 10.1109/CSFW.1997.596782.

[108] Harjasleen Malvai, Lefteris Kokoris-Kogias, Alberto Sonnino, Esha Ghosh, Ercan Oztürk, Kevin Lewi, and Sean F. Lawlor. "Parakeet: Practical Key Transparency for End-to-End Encrypted Messaging". In: *30th Annual Network and Distributed System Security Symposium*. NDSS. San Diego, California, USA, 2023. URL: https://www.ndss-symposium.org/ndss-paper/parakeet-practical-key-transparency-for-end-to-end-encrypted-messaging/.

[109] Moxie Marlinspike and Trevor Perrin. *The Sesame Algorithm: Session Management for Asynchronous Message Encryption*. Revision 2. Apr. 14, 2017. URL: https://signal.org/docs/specifications/sesame/sesame.pdf (visited on 05/01/2025).

[110] Moxie Marlinspike and Trevor Perrin. *The X3DH Key Agreement Protocol*. Revision 1. Nov. 4, 2016. URL: https://signal.org/docs/specifications/x3dh/x3dh.pdf.

[111] Simon Meier. "Advancing Automated Security Protocol Verification". Doctoral Thesis. ETH Zurich, 2013. DOI: 10.3929/ethz-a-009790675.

[112] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. "The TAMARIN Prover for the Symbolic Analysis of Security Protocols". In: *Computer Aided Verification (CAV)*. Berlin, Heidelberg, 2013. DOI: 10.1007/978-3-642-39799-8_48.

[113] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. "CONIKS: Bringing Key Transparency to End Users". In: *24th USENIX Security Symposium*. Washington, D.C., USA, 2015. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/melara.

[114] Ralph C. Merkle. "A Digital Signature Based on a Conventional Encryption Function". In: *Advances in Cryptology — CRYPTO '87*. Berlin, Heidelberg, 1988. DOI: 10.1007/3-540-48184-2_32.

[115] P. Mockapetris. *Domain Names - Concepts and Facilities*. Request for Comments RFC 1034. Internet Engineering Task Force, Nov. 1987. 55 pp. DOI: 10.17487/RFC1034.

[116] P. Mockapetris. *Domain Names - Implementation and Specification*. Request for Comments RFC 1035. Internet Engineering Task Force, Nov. 1987. 55 pp. DOI: 10.17487/RFC1035.

[117] *Module-Lattice-Based Key-Encapsulation Mechanism Standard*. FIPS 203. National Institute of Standards and Technology, Aug. 13, 2024. DOI: 10.6028/NIST.FIPS.203.

[118] Kevin Morio and Robert Künnemann. "Verifying Accountability for Unbounded Sets of Participants". In: *34th Computer Security Foundations Symposium (CSF)*. June 2021. DOI: 10.1109/CSF51468.2021.00032.

[119] moxieo. *Safety Number Updates*. Signal Messenger. URL: https://signal.org/blog/safety-number-updates/ (visited on 01/17/2022).

[120] MozillaWiki. *CA/Revocation Checking in Firefox*. URL: https://wiki.mozilla.org/CA/Revocation_Checking_in_Firefox (visited on 08/18/2023).

[121] Trevor Perrin and Moxie Marlinspike. *The Double Ratchet Algorithm*. Revision 1. Nov. 20, 2016. URL: https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf.

[122] Andreas Pfitzmann and Marit Hansen. *A Terminology for Talking about Privacy by Data Minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management*. Aug. 2010. URL: http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf (visited on 12/22/2022).

[123] J. Postel. *User Datagram Protocol*. Request for Comments RFC 768. Internet Engineering Task Force, Aug. 1980. 3 pp. DOI: 10.17487/RFC0768.

[124] *Proof by Infinite Descent*. In: *Wikipedia*. Dec. 24, 2024. URL: https://en.wikipedia.org/w/index.php?title=Proof_by_infinite_descent&oldid=1264957154 (visited on 04/08/2025).

[125] *React*. Meta, July 19, 2022. URL: https://github.com/facebook/react (visited on 07/19/2022).

[126] International Committee of the Red Cross. *Annex I (to Protocol Additional I to the Geneva Conventions of 1949) : Regulations Concerning Identification, 6 June 1977*. June 6, 1977. URL: https://ihl-databases.icrc.org/en/ihl-treaties/api-annex-i-1977 (visited on 09/07/2023).

[127] International Committee of the Red Cross. *Protocol Additional to the Geneva Conventions of 12 August 1949, and Relating to the Protection of Victims of International Armed Conflicts (Protocol I), 8 June 1977*. June 8, 1977. URL: https://ihl-databases.icrc.org/en/ihl-treaties/api-1977 (visited on 09/07/2023).

[128] International Committee of the Red Cross. *The Geneva Conventions of August 12, 1949*. Aug. 12, 1949. URL: https://www.icrc.org/en/publication/0173-geneva-conventions-august-12-1949 (visited on 09/07/2023).

[129] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. Request for Comments RFC 8446. Internet Engineering Task Force, Aug. 2018. 160 pp. DOI: 10.17487/RFC8446.

[130] "Resolutions of the 34th International Conference of the Red Cross and Red Crescent". In: *International Review of the Red Cross* 106.927 (Dec. 2024). ISSN: 1816-3831, 1607-5889. DOI: 10.1017/S1816383124000729.

[131] Tilman Rodenhäuser, Mauro Vignati, Larry Maybee, and Hollie Johnston. *Digitalizing the Red Cross, Red Crescent and Red Crystal Emblems*. International Committee of the Red Cross, Nov. 3, 2022. URL: https://www.icrc.org/en/document/icrc-digital-emblems-report (visited on 11/30/2022).

[132] Jan Rooduijn, Dexter Kozen, and Alexandra Silva. "A Cyclic Proof System for Guarded Kleene Algebra with Tests". In: *Automated Reasoning*. Cham, 2024. DOI: 10.1007/978-3-031-63501-4_14.

[133] Rich Rosenbaum. *Using the Domain Name System To Store Arbitrary String Attributes*. Request for Comments RFC 1464. Internet Engineering Task Force, May 1993. 4 pp. DOI: 10.17487/RFC1464.

[134] Reuben N. S. Rowe and James Brotherston. "Automatic Cyclic Termination Proofs for Recursive Procedures in Separation Logic". In: *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*. New York, NY, USA, Jan. 16, 2017. DOI: 10.1145/3018610.3018623.

[135] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore. *OpenID Connect Core 1.0 Incorporating Errata Set 1*. Nov. 8, 2014. URL: https://openid.net/specs/openid-connect-core-1_0.html (visited on 09/24/2021).

[136] N. Sakimura, J. Bradley, M. Jones, and E. Jay. *OpenID Connect Discovery 1.0 Incorporating Errata Set 1*. Nov. 8, 2014. URL: https://openid.net/specs/openid-connect-discovery-1_0.html (visited on 07/15/2022).

[137] Nat Sakimura, John Bradley, and Naveen Agarwal. *Proof Key for Code Exchange by OAuth Public Clients*. Request for Comments RFC 7636. Internet Engineering Task Force, Sept. 2015. DOI: 10.17487/RFC7636.

[138] Benedikt Schmidt. "Formal Analysis of Key Exchange Protocols and Physical Protocols". Doctoral Thesis. ETH Zurich, 2012. DOI: 10.3929/ethz-a-009898924.

[139] Benedikt Schmidt, Simon Meier, Cas Cremers, and David Basin. "Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties". In: *25th Computer Security Foundations Symposium (CSF)*. June 2012. DOI: 10.1109/CSF.2012.25.

[140] Ulrich Schöpp. "Formal Verification of Processes". MA thesis. University of Edinburgh, 2001. URL: https://ulrichschoepp.de/Docs/msc.pdf (visited on 04/08/2025).

[141] Ulrich Schöpp and Alex Simpson. "Verifying Temporal Properties Using Explicit Approximants: Completeness for Context-free Processes". In: *Foundations of Software Science and Computation Structures*. Berlin, Heidelberg, 2002. DOI: 10.1007/3-540-45931-6_26.

[142] Maliheh Shirvanian, Nitesh Saxena, and Jesvin James George. "On the Pitfalls of End-to-End Encrypted Communications: A Study of Remote Key-Fingerprint Verification". In: *Proceedings of the 33rd Annual Computer Security Applications Conference*. New York, NY, USA, Dec. 4, 2017. DOI: 10.1145/3134600.3134610.

[143] Alex Simpson. "Cyclic Arithmetic Is Equivalent to Peano Arithmetic". In: *Foundations of Software Science and Computation Structures*. Berlin, Heidelberg, 2017. DOI: 10.1007/978-3-662-54458-7_17.

[144] *SOAP Signal Android Prototype*. May 3, 2023. URL: https://github.com/soap-wg/Signal-Android/releases/tag/ccs-proto (visited on 04/16/2025).

[145] *SOAP Web-Based Prototype*. Jan. 17, 2024. URL: https://github.com/soap-wg/soap-web/releases/tag/usenix (visited on 04/16/2025).

[146] Christoph Sprenger and Mads Dam. "On global induction mechanisms in a μ-calculus with explicit approximations". In: *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications* 37.4 (2003). ISSN: 1290-385X. DOI: 10.1051/ita:2003024.

[147] Christoph Sprenger and Mads Dam. "On the Structure of Inductive Reasoning: Circular and Tree-Shaped Proofs in the μCalculus". In: *Foundations of Software Science and Computation Structures*. Berlin, Heidelberg, 2003. DOI: 10.1007/3-540-36576-1_27.

[148] Christoph Sprenger, Tobias Klenze, Marco Eilers, Felix A. Wolf, Peter Müller, Martin Clochard, and David Basin. "Igloo: Soundly Linking Compositional Refinement and Separation Logic for Distributed System Verification". In: *Proceedings of the ACM on Programming Languages* 4 (OOPSLA Nov. 13, 2020). DOI: 10.1145/3428220.

[149] Emily Stark, Joe DeBlasio, and Devon O'Brien. "Certificate Transparency in Google Chrome: Past, Present, and Future". In: *IEEE Security & Privacy* 19.6 (Nov. 2021). ISSN: 1558-4046. DOI: 10.1109/MSEC.2021.3103461.

[150] *Start Integrating Google Sign-In into Your Android App | Google Sign-In for Android | Google Developers*. URL: https://developers.google.com/identity/sign-in/android/start-integrating (visited on 10/04/2022).

[151] Douglas Stebila. *Security Analysis of the iMessage PQ3 Protocol*. Mar. 1, 2024. URL: https://eprint.iacr.org/2024/357 (visited on 02/14/2025). Pre-published.

[152] Xibo Sun, Shixuan Sun, Qiong Luo, and Bingsheng He. "An In-Depth Study of Continuous Subgraph Matching". In: *Proc. VLDB Endow.* 15.7 (Mar. 1, 2022). ISSN: 2150-8097. DOI: 10.14778/3523210.3523218.

[153] Nikhil Swamy, Cătălin Hriţcu, Chantal Keller, Aseem Rastogi, Antoine Delignat-Lavaud, Simon Forest, Karthikeyan Bhargavan, Cédric Fournet, Pierre-Yves Strub, Markulf Kohlweiss, Jean-Karim Zinzindohoue, and Santiago Zanella-Béguelin. "Dependent Types and Multi-Monadic Effects in F*". In: *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. New York, NY, USA, Jan. 11, 2016. DOI: 10.1145/2837614.2837655.

[154] Paul Syverson and Matthew Traudt. "Self-Authenticating Traditional Domain Names". In: *2019 IEEE Cybersecurity Development*. SecDev. Tysons Corner, VA, USA, Sept. 2019. DOI: 10.1109/SecDev.2019.00030.

[155] *Telegram FAQ*. Telegram. URL: https://telegram.org/faq?setln=en#q-what-is-this-39encryption-key-39-thing (visited on 12/19/2023).

[156] Gadi Tellez and James Brotherston. "Automatically Verifying Temporal Properties of Pointer Programs with Cyclic Proof". In: *Journal of Automated Reasoning* 64.3 (Mar. 1, 2020). ISSN: 1573-0670. DOI: 10.1007/s10817-019-09532-0.

[157] *Twilio Incident: What Signal Users Need to Know*. Signal Support. URL: https://support.signal.org/hc/en-us/articles/4850133017242-Twilio-Incident-What-Signal-Users-Need-to-Know- (visited on 08/30/2022).

[158] *Use Contact Key Verification on iPhone*. iPhone User Guide. URL: https://support.apple.com/en-gb/guide/iphone/iph654dd8c53/ios (visited on 05/05/2025).

[159] Elham Vaziripour, Devon Howard, Jake Tyler, Mark O'Neill, Justin Wu, Kent Seamons, and Daniel Zappala. "I Don't Even Have to Bother Them!: Using Social Media to Automate the Authentication Ceremony in Secure Messaging". In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. CHI '19. New York, NY, USA, May 2019. DOI: 10.1145/3290605.3300323.

[160] *Verify End-To-End Encryption: Trusted Contacts List*. Viber. URL: https://help.viber.com/hc/en-us/articles/9061180581661-Verify-End-To-End-Encryption-Trusted-Contacts-List (visited on 12/19/2023).

[161] Théophile Wallez, Jonathan Protzenko, Benjamin Beurdouche, and Karthikeyan Bhargavan. "TreeSync: Authenticated Group Management for Messaging Layer Security". In: 32nd USENIX Security Symposium. Anaheim, CA, USA, Aug. 2023. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/wallez.

[162] Théophile Wallez, Jonathan Protzenko, and Karthikeyan Bhargavan. *TreeKEM: A Modular Machine-Checked Symbolic Security Analysis of Group Key Agreement in Messaging Layer Security*. Mar. 4, 2025. URL: https://eprint.iacr.org/2025/410 (visited on 04/30/2025). Pre-published.

[163] Von Welch, Mary Thompson, Douglas E. Engert, Steven Tuecke, and Laura Pearlman. *Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile*. Request for Comments RFC 3820. Internet Engineering Task Force, June 2004. 37 pp. DOI: 10.17487/RFC3820.

[164] *What Do the Three Colored Dots next to a Contact Mean? – Threema*. URL: https://threema.ch/en/faq/levels_expl (visited on 12/19/2023).

[165] Tarun Kumar Yadav, Devashish Gosain, Amir Herzberg, Daniel Zappala, and Kent Seamons. "Automatic Detection of Fake Key Attacks in Secure Messaging". In: Conference on Computer and Communications Security (CCS). New York, NY, USA, Nov. 7, 2022. DOI: 10.1145/3548606.3560588.

[166] Zhijie Zhang, Yujie Lu, Weiguo Zheng, and Xuemin Lin. "A Comprehensive Survey and Experimental Study of Subgraph Matching: Trends, Unbiasedness, and Interaction". In: *Proc. ACM Manag. Data* 2.1 (Mar. 26, 2024). DOI: 10.1145/3639315.

A

TAMARIN'S CONSTRAINT REDUCTION RULES

Below we list the constraint reduction rules added in this thesis and those from [53, 111].

Structural Constraint Reduction Rules added in Chapter 8

$\mathcal{S}_{\Delta}$: $\Gamma \rightsquigarrow (\Gamma, \Delta) \parallel \|_{\varphi \in \Delta}(\Gamma, \widehat{\varphi})$

if all $\varphi \in \Delta$ are guarded trace formulas such that $fv_{tmp}(\varphi) \subseteq fv_{tmp}(\Gamma)$ holds.

$\mathcal{S}_{W}$: $\Gamma, \Delta \rightsquigarrow \Gamma$

if Γ satisfies the well-formedness conditions **WF2**, **WF3**, and **WF5** (see proof of Theorem 1).

Basic Constraint Reduction Rules

$\mathcal{S}_{@}$: $\Gamma \rightsquigarrow \|_{ri \in R}\|_{f' \in acts(ri)}(i : ri, f = f', \Gamma)$

if $(f@i) \in \Gamma$ and $f \neq !\mathrm{K}^{\uparrow}(t)$ and $f@i \notin_E as(\Gamma)$.

$\mathcal{S}_{!\mathrm{K}^{\uparrow}@}$: $\Gamma \rightsquigarrow \|_{ri \in ND}\|_{f \in acts(ri)}(i : ri, !\mathrm{K}^{\uparrow}(t) = f, \Gamma)$

if $(!\mathrm{K}^{\uparrow}(t)@i) \in \Gamma$ and $!\mathrm{K}^{\uparrow}(t)@i \notin_E as(\Gamma)$ and $t \notin \mathcal{V}_{msg} \cup \mathcal{V}_{pub} \cup \mathcal{C}_{pub}$.

$\mathcal{S}_{=}$: $\Gamma \rightsquigarrow \|_{\sigma \in unify_E^{vars(\Gamma)}(t_1, t_2)}(\Gamma\sigma)$

if $(t_1 = t_2) \in \Gamma$ and $t_1 \neq_E t_2$.

$\mathcal{S}_{\doteq}$: $\Gamma \rightsquigarrow \Gamma\{i/j\}$

if $(i \doteq j) \in \Gamma$ and $i \neq j$.

$\mathcal{S}_{\bot}$: $\Gamma \rightsquigarrow \bot$

if $\bot \in \Gamma$.

$\mathcal{S}_{\neg,@}$: $\Gamma \rightsquigarrow \bot$

if $\neg(f@i) \in \Gamma$ and $(f@i) \in_E as(\Gamma)$.

$\mathcal{S}_{\neg,=}$: $\Gamma \rightsquigarrow \bot$

if $\neg(t_1 = t_2) \in \Gamma$ and $t_1 =_E t_2$.

$\mathcal{S}_{\neg,\doteq}$: $\Gamma \rightsquigarrow \bot$

if $\neg(i \doteq i) \in \Gamma$.

$\mathcal{S}_{\neg,\prec}$: $\Gamma \rightsquigarrow (i < j, \Gamma) \parallel (i \doteq j, \Gamma)$

if $\neg(j < i) \in \Gamma$ and neither $i \prec_\Gamma j$ nor $i = j$.

$\mathcal{S}_{\vee}$: $\Gamma \rightsquigarrow (\varphi_i, \Gamma) \parallel (\varphi_2, \Gamma)$

if $(\varphi_1 \vee \varphi_2) \in \Gamma$ and $\varphi_1 \notin_E \Gamma$ and $\varphi_2 \notin_E \Gamma$.

$\mathcal{S}_{\wedge}$: $\Gamma \rightsquigarrow (\varphi_1, \varphi_2, \Gamma)$

if $(\varphi_1 \wedge \varphi_2) \in \Gamma$ and not $\{\varphi_1, \varphi_2\} \subseteq_E \Gamma$.

$\mathcal{S}_{\exists}$: $\Gamma \rightsquigarrow (\varphi\{y/x\}, \Gamma)$

if $(\exists x{:}s.\, \varphi) \in \Gamma$ and $y{:}s$ fresh and $\varphi\{w/x\} \notin_E \Gamma$ for every term w of sort s.

$\mathcal{S}_{\forall}$: $\Gamma \rightsquigarrow (\sigma(\psi), \Gamma)$

if $(\forall \vec{x}.\neg(f@i) \vee \psi) \in \Gamma$ and $dom(\sigma) = set(\vec{x})$ and $\sigma(f@i) \in_E as(\Gamma)$ and $\sigma(\psi) \notin_E \Gamma$.

$\mathcal{S}_{\rhd}$: $\Gamma \rightsquigarrow \left\|_{ri \in R}\right\|_{u \in idx(concs(ri))} (i : ri, (i, u) \rightarrowtail (j, v), \Gamma)$

if $(f \rhd_v j) \in \Gamma$ and $f \neq !\mathsf{K}^{\uparrow}(t)$ and $f \neq !\mathsf{K}^{\downarrow}(t)$ and i fresh and there is no c s.t. $(c \rightarrowtail (j, v)) \in \Gamma$.

$\mathcal{S}_{!\mathsf{K}^{\uparrow}\rhd}$: $\Gamma \rightsquigarrow (!\mathsf{K}^{\uparrow}(t)@j, j < i, \Gamma)$

if $(!\mathsf{K}^{\uparrow}(t) \rhd_v i) \in \Gamma$ and j fresh and $t \notin_E kn^{\uparrow}_{\prec i}(\Gamma)$.

$\mathcal{S}_{!\mathsf{K}^{\downarrow}\rhd}$: $\Gamma \rightsquigarrow (i : [\mathsf{Out}(y)] \rightarrow [!\mathsf{K}^{\downarrow}(y)], (i, 1) \dashrightarrow (j, v), \Gamma)$

if $(!\mathsf{K}^{\downarrow}(t) \rhd_v j) \in \Gamma$ and i, y fresh and not $\exists c.(c \rightarrowtail (j, v)) \in \Gamma \vee (c \dashrightarrow (j, v)) \in \Gamma$.

$\mathcal{S}_{\dashrightarrow}$: $\Gamma \rightsquigarrow (c \rightarrowtail p, \Gamma) \parallel \left\|_{ri \in ND}\right\|_{u \in idx(prems(ri))} (i : ri, c \rightarrowtail (i, u), (i, 1) \dashrightarrow p, \Gamma)$

if $(c \dashrightarrow p) \in \Gamma$ and i fresh and $\forall p'.(c \rightarrowtail p') \notin \Gamma$ and $\forall x \in \mathcal{V}_{msg}.(c, !\mathsf{K}^{\downarrow}(x)) \notin cs(\Gamma)$.

Constraint Reduction Rules for Trace Induction

$\mathcal{S}_{\mathsf{last},\prec}$: $\Gamma \rightsquigarrow \bot$

if $(\mathsf{last}(i)) \in \Gamma$ and $i \prec_\Gamma j$ and $(j : ri) \in \Gamma$.

$\mathcal{S}_{\mathsf{last},\mathsf{last}}$: $\Gamma \rightsquigarrow (i \doteq j, \Gamma)$

if $\{\mathsf{last}(i), \mathsf{last}(j)\} \subseteq \Gamma$ and $i \neq j$.

$\mathcal{S}_{\neg,\mathsf{last}}$: $\Gamma \rightsquigarrow (i < j, \mathsf{last}(j), \Gamma) \parallel (\mathsf{last}(j), j < i, \Gamma)$

if $\neg(\mathsf{last}(i)) \in \Gamma$ and j fresh and neither $i \prec_\Gamma k$ nor $k \prec_\Gamma i$ for any $(\mathsf{last}(k)) \in \Gamma$.

Constraint Reduction Rules for Dependency Graph Well-Formedness

$\mathcal{DG}_{lbl}$: $\Gamma \rightsquigarrow ri = ri', \Gamma$
if $i : ri, i : ri' \subseteq \Gamma$ and $ri \neq_E ri'$.

$\mathcal{DG}_{\prec}$: $\Gamma \rightsquigarrow \bot$
if $i \prec_\Gamma i$.

$\mathcal{DG}_{\rightarrowtail}$: $\Gamma \rightsquigarrow (f = f', \Gamma)$
if $c \rightarrowtail p \in \Gamma$ and $(c, f) \in cs(\Gamma)$ and $(p, f') \in ps(\Gamma)$ and $f \neq_E f'$.

$\mathcal{DG}_{\triangleright}$: $\Gamma \rightsquigarrow (f \triangleright_v i, \Gamma)$
if $((i, v), f) \in ps(\Gamma)$ and $(f \triangleright_v i) \notin_E \Gamma$.

$\mathcal{DG}_{in}$: $\Gamma \rightsquigarrow (i \doteq j, \Gamma)$
if $\{(i, v) \rightarrowtail p, (j, u) \rightarrowtail p\} \subseteq \Gamma$ and $i \neq j$ and $u = v$, or
$\Gamma \rightsquigarrow \bot$
if instead $u \neq v$.

$\mathcal{DG}_{out}$: $\Gamma \rightsquigarrow (i \doteq j, \Gamma)$
if $\{c \rightarrowtail (i, v), c \rightarrowtail (j, u)\} \subseteq \Gamma$ and c linear in Γ and $i \neq j$ and $u = v$, or
$\Gamma \rightsquigarrow \bot$
if instead $u \neq v$.

$\mathcal{DG}_{Fr}$: $\Gamma \rightsquigarrow (i \doteq j, \Gamma)$
if $\{i : \emptyset \dashv[]\rightarrow \mathsf{Fr}(m), j : \emptyset \dashv[]\rightarrow \mathsf{Fr}(m)\} \subseteq_E \Gamma$ and $i \neq j$.

Constraint reduction rules for dependency graph normal form

$\mathcal{N}1$: $\Gamma \rightsquigarrow \bot$
if $(i : ri) \in \Gamma$ and ri not $\downarrow^{\mathcal{R}}_E$-normal.

$\mathcal{N}2$: $\Gamma \rightsquigarrow \bot$
if $(i : !\mathsf{K}^{\downarrow}(\langle t_1, t_2\rangle) \dashv[!\mathsf{K}^{\uparrow}(\langle t_1, t_2\rangle)]\rightarrow !\mathsf{K}^{\uparrow}(\langle t_1, t_2\rangle)) \in_E \Gamma$.

$\mathcal{N}3_{\uparrow}$: $\Gamma \rightsquigarrow (i \doteq j, \Gamma)$
if $\{!\mathsf{K}^{\uparrow}(t)@i, !\mathsf{K}^{\uparrow}(t)@j\} \subseteq_E as(\Gamma) \cup \Gamma$ and $i \neq j$.

$\mathcal{N}3_{\downarrow}$: $\Gamma \rightsquigarrow (i \doteq j, \Gamma)$
if $\{((i, 1), !\mathsf{K}^{\downarrow}(t)), ((j, 1), !\mathsf{K}^{\downarrow}(t))\} \subseteq_E cs(\Gamma)$ and $i \neq j$.

$\mathcal{N}4$: $\Gamma \rightsquigarrow (i < j, \Gamma)$
if $((i, 1), !\mathsf{K}^{\downarrow}(t)) \in_E cs(\Gamma)$ and $!\mathsf{K}^{\uparrow}(t)@j \in_E as(\Gamma) \cup \Gamma$ and not $i \prec_\Gamma j$.

Constraint reduction rules for subterms and natural numbers

The rules below were introduced in [53]. [53] also introduces two complex constraint reduction rules FRESH−ORDER and MONOTONICITY, which we omit here.

$\mathcal{S}_{\mathsf{RECURSE}}$: $\Gamma \rightsquigarrow \|_{1\le i\le n}(t = t_i, \Gamma) \parallel (t \sqsubset t_i, \Gamma)$

if $t \sqsubset f(t_1, \ldots, t_n) \in \Gamma$ and f is neither a reducible operator nor associate-commutative.

$\mathcal{S}_{\mathsf{AC-RECURSE}}$: $\Gamma \rightsquigarrow (\exists x.t \circ x = t_1 \circ \cdots \circ t_n, \Gamma) \parallel \|_{1\le i\le n}(t \sqsubset t_i, \Gamma)$

if $t \sqsubset t_1 \circ \cdots \circ t_n \in \Gamma$ and $\circ$ is an associative-commutative operator, not reducible, and $\circ \neq +$, and $t_1, \ldots, t_n$ are chosen maximally w.r.t. $\circ$.

$\mathcal{S}_{\mathsf{NEG-RECURSE}}$: $\Gamma \rightsquigarrow \|_{1\le i\le n}(t \neq t_i, \neg(t \sqsubset t_i), \Gamma)$

if $\neg(t \sqsubset f(t_1, \ldots, t_n)) \in \Gamma$ and f is neither associative-commutative nor reducible.

$\mathcal{S}_{\mathsf{NEG-AC-RECURSE}}$: $\Gamma \rightsquigarrow (\forall x.t \circ x \neq t_1 \circ \cdots \circ t_n, \Gamma) \parallel \|_{1\le i\le n}\neg(t \sqsubset t_i, \Gamma)$

if $\neg(t \sqsubset t_1 \circ \cdots \circ t_n) \in \Gamma$ and $\circ$ is an associative-commutative operator, not reducible, and $\circ \neq +$, and $t_1, \ldots, t_n$ are chosen maximally w.r.t. $\circ$.

$\mathcal{S}_{\mathsf{CHAIN}}$: $\Gamma \rightsquigarrow \bot$

if $\{t_0 \sqsubset x_0, \ldots, t_n \sqsubset x_n\} \subseteq \Gamma$ and x_i is a syntactic subterm of $t_{(i+1) \bmod (n+1)}$ and not below a reducible operator.

$\mathcal{S}_{\mathsf{NEG}}$: $\Gamma \rightsquigarrow (\neg(s \sqsubset t), \Gamma) \parallel (s \neq t, \Gamma)$

if $\{\neg(s \sqsubset t), t \sqsubset r\} \subseteq \Gamma$.

$\mathcal{S}_{\mathsf{NEG-NAT}}$: $\Gamma \rightsquigarrow (t \sqsubset s + 1, \Gamma)$

if $\neg(s{:}nat \sqsubset t{:}nat) \in \Gamma$.

$\mathcal{S}_{\mathsf{NAT}}$: $\Gamma \rightsquigarrow (\exists x.s + x = t, \Gamma) \parallel \Gamma[s \mapsto s{:}nat]$

if $s \sqsubset t{:}nat \in \Gamma$ and s is a term of sort *nat* or a variable of sort *msg*.

$\mathcal{S}_{\mathsf{INVALID}}$: $\Gamma \rightsquigarrow \bot$

if $s \sqsubset t \in \Gamma$ and the sorts of s and t are one of (i) *fresh* and *nat*, (ii) *pub* and *nat*, (iii) arbitrary and *pub*, (iv) arbitrary and *fresh*.

$\mathcal{S}_{\mathsf{UTVPI}}$: $\Gamma \rightsquigarrow \|_{1\le i\le n}(a_i = b_i, \Gamma)$

if $\{s_0 \sqsubset t_0, \ldots, s_n \sqsubset t_n\} \subseteq \Gamma$ and $a_i = b_i$ are determined according to the UTVPI-algorithm (see [53]) and s_i and t_i are terms of sort *nat*.