On Secure Messaging

Katriel Cohn-Gordon

Merton College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity Term 2018

To Mike (1948-2017), who showed me who I want to be.

עליו השלום

Abstract

What formal guarantees should a secure messaging application provide? Do the most widely-used protocols provide them? Can we do better? In this thesis we answer these questions and with them give a formal study of modern secure messaging protocols, which encrypt the personal messages of billions of users.

We give definitions and analyses of two protocols: one existing (Signal) and one new (ART). For Signal, we begin by extending and generalising classic computational models, in order to apply them to its complex ratcheting key derivations. With a threat model in mind we also define a security property, capturing strong secrecy and authentication guarantees including a new one which we call “post-compromise security”. We instantiate Signal as a protocol in our model, stating its security theorem and sketching a computational reduction.

Signal only supports encrypting messages between two devices, and so most implementers have built custom protocols on top of it to support group conversations. These protocols usually provide weaker security guarantees, and in particular usually do not have post-compromise security. We propose a new protocol called ART, whose goal is to bring Signal’s strong security properties to conversations with multiple users and devices. We give a design rationale and a precise definition of ART, and again generalise existing computational models in order to formally specify its security properties and sketch a security reduction.

ART has enjoyed widespread interest from industry, and we aim to turn it into an open standard for secure messaging. To that end, we have brought it to the IETF and formed a working group called Messaging Layer Security, with representatives from academia as well as Facebook, Google, Twitter, Wire, Cisco and more. Through MLS, we hope to bring ART’s strong guarantees to practical implementations across industry.

After concluding our analyses we pause for a moment, and start looking towards the future. We argue that for complex protocols like Signal and ART we are reaching the limits of computational methods, and that the future for their analysis lies with symbolic verification tools. To that end we return to the symbolic model and give a number of case studies, in each one showing how a traditional limitation of symbolic models can in fact be seen as a modelling artefact.

Katriel Cohn-Gordon, Merton College

Statement of Originality

I have enjoyed a long-running collaboration with Luke Garratt, a fellow DPhil student of Cas Cremers's. We are joint authors of [57, 59, 60], and content from [59, 60] appears in this thesis. At a high level, my focus has been more on the protocol designs and security models, capturing the properties that we want to prove, while Luke's main contributions are to the reductions and their methodology.

Luke and I jointly led the research for the Signal paper [59]. I defined the protocol specification based on the open-source code, and gave the security property that should hold of it; Luke worked on the security model and on the proof of the main theorem. The paper appears in both of our theses.

I led the research on the ART paper [60], with particular contributions to the protocol design and specification as well as the overall writing. Luke's main contributions were again to the analysis and reduction, Jon Millican wrote our test implementation, and Kevin Milner the Tamarin model.

I also led the research for the improvements to symbolic modelling, working with Ralf Sasse on some details of the Tamarin modelling and later Dennis Jackson on a systematic treatment of the underlying theory.

All writing is my own.

Acknowledgements

As with any thesis, there are too many to thank properly here, but I'll do my best.

First and foremost, Cas, thank you for making research seem easy! There were plenty of times when I ran out of steam, or ideas, or motivation; it never took more than a single meeting (for which you always made time) to get me started again on the right track. Thank you for always having ideas, making time to read everything I wrote, finding and making me opportunities big and small, and shepherding me away from many distractions. I absolutely couldn't have done any of it without your guidance.

Thanks also to the other members of the group—Dennis, Kevin, Luke, Martin, Nick and honorary members Chad, Beth and Yang—for always being there to chat about work and laugh about blockchains, and to Ben and Douglas for their proof wizardry and general sensibleness. Particular credit has to go to Kevin and to Carl Chudyk for making sure I didn't get too much work done in my first year, and to Lue Whalefin, Master of Teacups, for making the proofs work despite my best efforts to change the model out from under him. And thanks to my examiners Mark Ryan and Tim Muller, and to all the anonymous reviewers who helped to improve our science.

There were plenty of other people at Oxford who helped me on my way. Thanks to Andrew Martin and the CDT for the reminder that it's not always about the technical details; to the CDT, EPSRC and Merton College for the cold hard cash; to Julie, Maureen and David for silently doing huge amounts of administrative work behind the scenes so that I didn't have to.

Thanks to the friends I made at Merton and beyond: to my housemates Beth, Michael, Brendan, Bahar, Miriam and Fi for putting up with the board games and disassembled computers, to the Merton MCR who provided so many hilarious evenings and recovery brunches, and to the Cambridge crowd for all the laughs and joy. Special credit to Max "The Fireman" Shepherd for constantly impressing me with new and terrible ideas (I mean, really, who goes cycling with a bouldering mat on their back?!).

Thanks to Jon and Ioannis at Facebook for tempting me away from the ivory tower with global impact and a pastry chef, and to them and the others at Facebook who smoothed over the bumps in my internship. Thanks also to the HR team at a certain other large tech company for reminding me that red tape is everywhere, not just at Oxford.

Thank you to my dear family: Avra for all the teaching, life lessons and love, and Reuben for the science and laughs and crazy stories (not limited to pitching me a tent in your room, just because I asked). Mike, I wish you could have made it to see me hand in, but thank you for setting me on the right path and for giving me the best leg up I could have hoped for.

Finally, Melissa, thank you for keeping me on track as I desperately tried to steer off it, for all the art, the holidays, the cooking (meatballs and otherwise), and for sticking with me the whole time despite the puns.

CONTENTS

Signal and ART 37

Appendices

CHAPTER 1

INTRODUCTION

In 2014, WhatsApp announced that they would turn on end-to-end encryption for their hundreds of millions of users. This was a happy if rather surprising development, given that for the previous several decades most secure messaging applications were limited to chatrooms or small deployments for "the privacy-conscious user[1]".

The protocol WhatsApp chose to use—Signal—has an interesting history. Its genesis was a phone app called TextSecure, building on ideas from a protocol called Off-the-Record Messaging (OTR) and developed among others by the independent security researcher Moxie Marlinspike. A few years later, Marlinspike worked with Trevor Perrin to develop and adapt TextSecure's encryption protocol into what became the Signal Protocol. At the time that WhatsApp started to deploy it, Signal had no precise specification, threat model or claimed security guarantee, unlike dozens of extant academic proposals which often had all of these but did not manage to convince billion-dollar companies to integrate their designs. (This is not to say that Signal got lucky: one key component of its success is that unlike many protocols it was built with real-world constraints in mind, by people with experience in writing secure software and running large systems.)

The story of this thesis starts with Signal's launch: I wanted to fill in the theoretical background to Signal, define some formal guarantees that it might or might not provide, and then either prove that it did meet them or give concrete attacks showing that it did not. This seemed like a noble sort of goal and perhaps even something that might make for a good first paper.

It turned out that doing this required some preliminary theoretical work: when Luke Garratt, Cas Cremers and I tried to figure out exactly what properties we should prove of Signal, it turned out to achieve something stronger than what was defined by any of the existing security models. In trying to pin this property down we ventured into some rather theoretical waters but eventually emerged on the other side with a definition: "Post-Compromise Security (PCS)".

PCS, as the name suggests, is the property that a protocol is secure *after* a

[1] The set of people who install experimental private messaging apps has an unfortunate tendency not to overlap with the set of people with whom one wants to converse.

compromise occurs. This has been described before in the literature (often somewhat contradictorily as both "future" and "backward" security), but the precise definition is rather more complex than the intuitive one, and there was some interesting science to be done in fleshing out the details.

Regardless, with the definition of PCS firmly in hand, we returned to the original task of studying Signal; I rolled up my sleeves and jumped into the Java implementation, the best definition we could find of what the protocol actually did. Working with Ben Dowling and Douglas Stebila, and with a lot of help and explanations from Trevor Perrin, we

(i) gave a more formal specification of Signal, translating from Java to a language more familiar to cryptographers[2],

(ii) built a computational security model into which we could fit our specification, combining the ideas from our PCS paper with some recent "multi-stage" models,

(iii) figured out a property which we agreed captures what Signal actually provided, and

(iv) wrote a traditional game-hopping reduction that Signal provided our property in our model.

This led to our Signal paper, but there were quite a few questions left.

First, while figuring out which parts of the Signal codebase were "the important bits" and which were "just for engineering", we quickly realised that the elegant distinction in our heads didn't actually correspond to a clean separation of concerns. We made do as best as we could, extracting the cryptographic core[3] and leaving out the extra details. However, some of the details which we left out were actually fairly substantial, and one in particular stood out: group messaging.

WhatsApp supports group conversations natively, and since Signal is fundamentally a two-device protocol a new idea was necessary to integrate it into WhatsApp. The new idea was, more or less, to use Signal to distribute a cryptographic key, and then to ignore Signal entirely and just use that key to send messages. This implied that group conversations, unlike pairwise ones, do not provide PCS.

This seemed a bit of a shame, since one of the many great features of Signal is its PCS guarantee. We therefore set out in the noble tradition of academia to go off and solve the problem ourselves, though this time keeping in mind the constraints that Signal had to respect. This led to the design of Asynchronous Ratcheting Trees (ART), which we brought to the Internet Engineering Task Force (IETF) to standardise in the form of Messaging Layer Security (MLS).

Second, while the computational reduction for Signal was just about tractable to produce by hand, it was reaching the limits of what a handful of grad students could do for a paper. Indeed, when we tried to write the same type of reduction for ART, we ended up needing to skip over proving authentication, leaving that to a separate analysis. The main reason is that the number of cases to consider in

[2] $\LaTeX$

[3] cryptographic core, *n.* "The bit we analysed."

these large protocols grows very fast, and the approaches we have to reason about them do not scale well. For reference, the full reduction for Signal is 19 pages of dense technical reasoning, and a reduction [73] for two modes of the Transport Layer Security (TLS) 1.3 handshake is around 14 pages; this is without the model definitions, which are usually protocol-specific.

We'd used pen-and-paper methods in the analyses so far. However, the research group in fact has a lot of expertise in mechanised approaches, and in particular in protocol verification using the Tamarin prover. Moving to tool-assisted reasoning has many benefits, in particular when dealing with protocols with a large number of cases. Indeed, Tamarin was recently used to reason about the security of TLS 1.3, a similarly-complex protocol. It is therefore very tempting to argue that with Signal and ART we are starting to reach the limits of the existing computational approaches, and that it's time to move to the symbolic model.

When reasoning about the same protocol, a symbolic verification is generally considered to provide weaker guarantees than a computational reduction, though formally comparing the two is not easy. In part, this is because the symbolic abstraction hides features of the cryptographic building blocks that often lead to attacks. I therefore wondered how many of these features are really due to limitations of the abstraction, and how many due to the particular modelling choices that were made. This idle wondering led to a number of case studies, and a paper in the works.

Many more questions remained, of course. Group messaging is not the only feature of Signal that we left out; can we analyse the others? The public key infrastructure and the logic for when to retry sending a message are two major omissions, but there are plenty more. Are there secure ways to provide other features users want—URL previews, GIF search, stickers and animations, read receipts, contact discovery, and so on? Can we improve the way computational reductions are built, so that they can scale to protocols like ART? What other parts of the Internet are ready for the Signal treatment?

At this point, time caught up with me, and I wrote this thesis.

1.1 Overview

We open in Chapter 2 with the background we'll need. We first survey past and present secure messaging systems in §2.1, focussing on those which we see as ancestors of the Signal protocol as well as those which achieved more widespread use. This survey leads up to a high-level description of the Signal protocol itself.

In §2.2 we look at the academic literature and theory of protocol analysis. We'll distinguish between symbolic and computational approaches, select the latter for now as our tool of choice, and give enough background to be able to define formal properties of protocols.

We give formal definitions and analyses of two protocols: Signal and ART. We begin in Chapter 3 by extending and generalising computational models, in order to apply them to Signal. This allows us to formally model Signal, including its complex “ratcheting” key derivations. With its threat model in mind we also define a security property, capturing strong secrecy and authentication guarantees including post-compromise security. In Chapter 4 we instantiate Signal as a protocol in our model, stating its security theorem and sketching a proof.

Signal only supports messages between *two* devices, and so most implementers have built custom protocols on top of it to support group conversations. In Chapter 5 we propose a new protocol called ART, whose goal is to bring Signal’s strong security properties to conversations with multiple users and devices. We give a design rationale and a precise definition of ART, and once again generalise existing computational models in order to formally specify its security properties and sketch a security reduction.

We then pause for a moment, reflect on our analyses, and start looking towards the future. We argue in Chapter 6 that for complex protocols like ART we are reaching the limits of computational methods, and that the future for their analysis lies with symbolic verification tools. In §6.2 we return to the symbolic model and give a number of case studies, in each one showing how a traditional limitation of symbolic models can in fact be seen as a modelling artefact.

Finally, we conclude in Chapter 7.

Notation Notation, acronyms, TAMARIN models and a bibliography appear in the Appendices. Acronyms and in-document references are in blue and citations in green; both are clickable in the PDF version of this document. Gender neutral names and pronouns are used throughout, in case you wonder why our trusty A and B are now called Alex and Blake.

1.2 Publications

The technical contributions in this thesis are in part drawn from my papers

> Katriel Cohn-Gordon, Cas J. F. 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 2017, Paris, France, April 26-28, 2017*. IEEE, 2017, pp. 451–466. DOI: 10.1109/EuroSP.2017.27. URL: https://doi.org/10.1109/EuroSP.2017.27

and

> Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican and Kevin Milner. *On Ends-to-Ends Encryption: Asynchronous Group Messaging with Strong Security Guarantees*. Cryptology ePrint Archive, Report 2017/666. http://eprint.iacr.org/2017/666. 2017

(to appear in CCS 2018) and

> Katriel Cohn-Gordon and Cas Cremers. *Mind the Gap: Where Provable Security and Real-World Messaging Don't Quite Meet.* Cryptology ePrint Archive, Report 2017/982. http://eprint.iacr.org/2017/982. 2017

Some theoretical groundwork was laid in

> Katriel Cohn-Gordon, Cas J. F. Cremers and Luke Garratt. 'On Post-compromise Security'. In: *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016.* IEEE Computer Society, 2016, pp. 164–178. DOI: 10.1109/CSF.2016.19. URL: https://doi.org/10.1109/CSF.2016.19

and the designs in Chapter 5 are the foundation of

> Richard Barnes, Jon Millican, Emad Omara, Katriel Cohn-Gordon and Raphael Robert. *The Messaging Layer Security Protocol.* Internet-Draft draft-barnes-mls-protocol-00. IETF Secretariat, Feb. 2018. URL: https://www.ietf.org/internet-drafts/draft-barnes-mls-protocol-00.txt

CHAPTER 2
BACKGROUND

This chapter is unpublished and my own work. §2.1.3 is informed by the description of Signal from [59].

In this background chapter we cover two main topics. First, in §2.1 we give a brief history of secure messaging and a detailed description of the Signal messaging protocol, highlighting especially its modelling and analysis challenges and the new ideas it presented. Second, in §2.2 we give an overview of formal techniques for proving protocol security, focusing in particular on the Bellare-Rogaway (B-R) game-based formalism and its many successors. By the end of the chapter, we will be in a position to give a formal model for the Signal protocol.

2.1 Secure Messaging: State of the Art

We start with an overview of widely-used messaging systems, focusing on those which we see as inspirations for or ancestors of the Signal protocol.

2.1.1 Historical

From the early days of the ARPANET, computer networks have been used for person-to-person communication in the form of email and instant messaging: MIT's Compatible Time-Sharing System (CTSS) implemented a MAIL command, and Internet standards began to settle on an email header format from as early as 1973 [29]. Security, however, is a rather later addition: while systems such as Pretty Good Privacy (PGP) [47] and s/MIME [147] have been around for decades, and TLS is becoming more widely adopted for connections between SMTP servers [91], end-to-end email encryption is not widespread. In other words, most email messages sent today are trivially readable by people other than their intended peers.

Instant messaging has also existed in one form or another since the early days of the Internet, beginning with simple Unix commands for users of the same computer to communicate to one another. It achieved widespread adoption in the 1990s through

$$\begin{array}{rl}
A \text{ and } B: & \text{agree on a shared key } k_{AB} \\
A \rightarrow B: & \{M_1\}_{k_{AB}} \\
A \rightarrow B: & \{M_2\}_{H(k_{AB})} \\
B \rightarrow A: & \{M_3\}_{H(H(k_{AB}))} \\
\vdots &
\end{array}$$

Figure 2.1: An example of symmetric ratcheting similar to that used in SCIMP.

several commercial offerings such as AIM, ICQ, MSN and Yahoo! Messenger. The Jabber protocol, standardised as the Extensible Messaging and Presence Protocol (XMPP) in RFC6120 [154] promoted interoperability of messaging clients and allowed for open-source programs such as Pidgin to amalgamate connections through various different services. As with email, early instant messaging protocols did not have security features and messages were transmitted over the network in plain text. Even modern, commercial instant messaging applications do not generally adopt end-to-end encryption, opting instead to allow the service providers to read all messages passing through their platform. (WhatsApp is of course the main exception to this observation.)

2.1.1.1 Secure Instant Messaging

There have been many systems designed to build security into instant messaging protocols. Unger et al. [163] evaluate and systematise current secure messaging systems, measuring their security, usability and ease of use. We focus on those systems which we consider as the core inspirations for the Signal protocol, referring the reader to Unger et al. [163] for a more detailed survey.

SCIMP [137], or the Silent Circle Instant Messaging Protocol, is a protocol designed by Silent Circle for their secure messaging app. It used an early *symmetric* ratcheting design, in which new encryption keys are derived deterministically from previous ones whenever they are used (Figure 2.1). If an adversary learns the encryption key being used at a particular time, say $H(H(k_{AB}))$, they cannot go back and derive earlier encryption keys (say k_{AB}) without inverting the function H. Assuming that this is hard, this scheme thus leads to a form of forward secrecy. (This design is similar to so-called “forward-secure” symmetric encryption [26].)

SCIMP was an early example of a deployed stateful protocol, because its hash chains of keys are stored in local protocol state.

OTR Off-the-Record Messaging is one of the earliest encrypted instant messaging protocols [37, 146]; the name refers both to a protocol and to a plugin for various instant messaging applications. The plugin allows users who share a secret passphrase (or who know each other's public key) to authenticate sent and received messages in an end-to-end manner.

OTR introduced the concept of *ratcheting* in order to achieve a fine-grained notion of key freshness. Users attach fresh ephemeral Diffie–Hellman (DH) public keys to each authenticated message, and use the DH shared keys derived from one message to encrypt the next one (Figure 2.2).

$$\begin{array}{rl}
A \to B: & g^{x_1} \\
B \to A: & g^{y_1} \\
A \to B: & g^{x_2},\ \{M_1\}_{H(g^{x_1 y_1})} \\
B \to A: & g^{y_2},\ \{M_2\}_{H(g^{x_2 y_1})} \\
A \to B: & g^{x_3},\ \{M_3\}_{H(g^{x_2 y_2})} \\
\vdots &
\end{array}$$

Figure 2.2: A sequence of message exchanges in OTR. Figure based on [37, §4.1].

The new message key derivations for each round trip ensure that the lifetime of each message key is very short. For example, we see above that the key $k_{21} = H(g^{x_2 y_1})$ is used only for encrypting the single message M_1. This reduced key lifetime means that compromise of a particular key reveals only a limited set of messages. In particular, OTR achieves forward secrecy in short time periods.

TextSecure [122] was the first iteration of Open Whisper Systems's secure messaging app. It contained the first definition of what later became known as Signal's "Double Ratchet", which effectively combines the ideas of OTR's asymmetric ratchet with SCIMP's symmetric design. TextSecure's combined ratchet was originally called the "Axolotl Ratchet", though the name Axolotl was used by some to refer to the entire protocol. There were three major versions of TextSecure: v1 was based on OTR; v2 added the Axolotl Ratchet and v3 included some changes to the cryptographic primitives and the wire protocol. Signal is based on TextSecure v3.

Open Whisper Systems subsequently merged TextSecure v3 and RedPhone (a secure telephony app) and renamed the result Signal. The underlying protocol was also renamed to Signal, and Axolotl to the Double Ratchet.

iMessage was perhaps the first messaging protocol to demonstrate end-to-end encryption on a global scale. It is a proprietary protocol built by Apple providing end-to-end encryption between groups of devices, avoiding many of the usability issues of previous designs. Notably, to ease usage Apple perform all of the key management and distribution for iMessage keys, and (as of this writing) users have no way to verify that they are doing so correctly. Garman et al. [85] found a number of severe flaws in iMessage that seriously undermined its security.

SafeSlinger is a secure messaging protocol and app with a focus on usability, proposed by Farb et al. [79]. It uses an underlying primitive—DH key trees—in a similar fashion to our proposal in Chapter 5, but without the asynchronicity or efficiency properties we make use of.

Others There are dozens of other "secure messaging" applications, with widely varying usage; most are encrypted in transit but give the service provider access to message plaintext. Viber has close to one billion users and implements their own variant of the Signal protocol; Wire reimplemented Signal's Double Ratchet, Wickr invented their own non-Signal protocol for which they claim forward secrecy, and Threema invented one which does not even claim forward secrecy. (Telegram is a widely-used app which claims to provide secure messaging but has invented its own extremely complex and unstudied protocol using nonstandard cryptographic constructions.)

By studying the widely-used Signal protocol instead of any particular application, we aim for our work to be broadly useful and applicable.

2.1.2 Signal enters the field

We move now to the Signal protocol itself, which we will describe and analyse in considerable detail. The Signal messaging protocol is

> a ratcheting forward secrecy protocol that works in synchronous and asynchronous messaging environments [119, 122].

Before jumping in, however, it is worth taking a moment to consider the two main reasons that we consider Signal to be an interesting topic of analysis: challenge and impact.

Challenge Signal introduced a number of new ideas which, at the time, were not well understood from a formal analysis standpoint. Understanding how to describe, specify, model and prove theorems about such systems is important not just for our specific analysis but also to increase the expressivity of our formal models in general.

First, while ostensibly a messaging application, Signal's core can also be considered a form of key exchange protocol, with security goals close to those studied in the latter field. However, the key exchange and messaging components are closely intertwined, and it does not factor neatly into the composition of a key exchange and a transport layer protocol.

Second, Signal is inherently stateful: encryption keys and protocol actions at a given time do not just depend on the immediately preceding handshake but also on data derived from messages before that. Most existing key exchange models consider protocols whose state only lasts until they derive a session key, but this is not sufficient to capture Signal's long-lived conversations.

Third, Signal's security properties include not just standard notions of confidentiality and integrity for end-to-end encryption but also intricate forms of forward security and *post-compromise security*. The latter was only recently described by Cohn-Gordon, Cremers and Garratt [57], formalising the intuition of "healing" from the iterated DH exchanges. Existing key exchange models generally rule out attacks against sessions whose endpoints' long-term keys have been compromised, even though Signal can resist some such attacks.

Impact While most secure messaging applications are used by tens or hundreds of thousands, Signal's adoption by WhatsApp [166] brought it to up to one billion users, or a significant percentage of all humans on the planet. Even setting that aside, Signal is widely adopted by other apps, including Skype [120], Google Allo [124], Facebook Messenger [78], Silent Circle [137], and Pond [113]. The OMEMO Multi-End Message and Object Encryption (OMEMO) extension adds a Signal-based Double Ratchet to XMPP, and is used by Cryptocat v2, Conversations, and ChatSecure.

Signal is now informally considered a standard for two-party end-to-end encryption, and especially in light of the new Signal Foundation [125] we foresee that its usage and uptake will only continue to increase. Increased confidence in Signal's underlying designs thus carries over to many other contexts.

2.1.3 Signal's design

The Signal protocol sends encrypted messages between two parties who have each been allocated long-term public/private "identity" key pairs. Encryption keys are derived through a combination of the techniques shown in Figures 2.1 and 2.2: an asymmetric ratchet like that of OTR is used to derive a sequence of secret values known as "root keys", and a symmetric ratchet like that of SCIMP is applied to the root keys to derive message keys. Additional modifications allow for asynchronous setup and out-of-order messages.

Signal conversations are organised into long-lived exchanges which we call *sessions*, using terminology from the key exchange literature. We can roughly separate sessions into four phases, although phase boundaries in implementations are rather loose.

Keys

Signal distinguishes between at least ten different classes of key, depicted in Table 2.1, so for ease of reading we'll introduce a standardised notation. Keys are written in italics and end with the letter k. For asymmetric key pairs, the corresponding public key ends with the letters pk, and is always computed by group exponentiation with base g and the private key as the exponent: $pk = g^k$. If the identity of the agent A who generates a key is unclear we mark this in subscript (i.e. k_A), but omit this where it is clear.

To identify these keys uniquely, we write the index of the stage deriving a key k in superscript; thus, rk_A^0 would be the root key derived by A in the initial stage [0], and $mk_A^{\textbf{sym-ri}:x,y}$ the message key derived in stage [**sym-ri**:x,y]. We'll defer the full definitions of stage identifiers to §3.2 on page 40; the important part is that they

asymmetric	ipk_A	ik_A	A's long-term identity key pair
	$prepk_B$	$prek_B$	B's medium-term (signed) prekey pair
	$eprepk_B$	$eprek_B$	B's ephemeral prekey pair
	epk_A	ek_A	A's ephemeral key pair
	$rchpk_A^a$	$rchk_A^a$	A's a^{th} ratchet key pair
symmetric		$ck_A^{\textbf{sym-ir}:a,y}$	y^{th} key in A's a^{th} send chain
		$ck_A^{\textbf{sym-ri}:a,y}$	y^{th} key in A's a^{th} receive chain
		$mk_A^{\textbf{sym-ir}:a,y}$	y^{th} message key in A's a^{th} send chain
		$mk_A^{\textbf{sym-ri}:a,y}$	y^{th} message key in A's a^{th} receive chain
		rk_A^a	A's a^{th} root key

Table 2.1: Keys used in the Signal protocol. Asymmetric key pairs show public (pk) and private (k) components.

contain a stage type (e.g. **sym-ri**), an asymmetric update counter x, and sometimes a symmetric update counter y. Not all stages derive all keys: for example, there is no $rk^{\textbf{sym-ri}:x,y}$, since root keys are not affected by symmetric updates.

The naming scheme for keys is also role-agnostic: in intended operation, keys will be equal iff they have the same name. As with stages, agents have different intended uses for the same key: for example, the initiator would use the key $mk^{\textbf{sym-ir}:x,y}$ for encrypting a message to send, and the responder would use the same key to decrypt it on receipt.

2.1.3.1 Registration

When installing Signal, users generate some cryptographic data, specifically:

(i) a long-term "identity" asymmetric secret key ik
(ii) a medium-term "signed prekey" $prek$
(iii) multiple short-term "one-time prekeys" $eprek$

The public keys corresponding to these values are uploaded to a semi-trusted key distribution server, together with a signature on $prek$ using ik.

This data includes long-term identity keys, as is standard for key exchange protocols. However, it also includes a batch of precomputed ephemeral keys known as "prekeys", which will be used in future exchanges. These are designed to enable forward secrecy in conversations with offline recipients: conventional Authenticated Key Exchange (AKE) wisdom says that to achieve forward secrecy with a DH exchange, the recipient must provide an ephemeral key specific for the exchange. Since recipients may be offline at any time, Signal has them generate and upload ephemeral keys in advance.

If the one-time ephemeral keys stored at the server are exhausted, the session

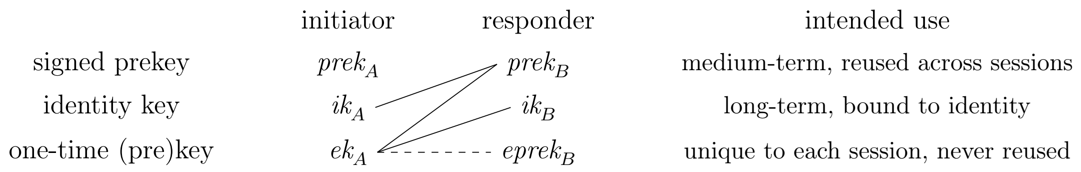


Figure 2.3: DH private keys used in the Signal Key Exchange KDF. An edge between two private keys (e.g., ik_A and $prek_B$) indicates that their DH value ($g^{ik_A \cdot prek_B}$) is included in the final KDF computation. The dashed line is optional: it is omitted from the session key derivation if $eprek_B$ is not sent. Note the asymmetry: when Alex initiates a session with Blake, Alex's signed prekey is not used at all. Our freshness conditions in §4.1 on page 55 will be partially based on this graph.

can go ahead using only a medium-term key. To the best of our knowledge, Signal implementations will not notify the user if this has occurred; we see no technical obstacle to doing so, although the benefit would have to be weighed against the risk of notification fatigue. A similar form of key reuse is studied by Menezes and Ustaoglu [132].

2.1.3.2 Session setup

A conversation begins in Signal when an agent connects to a key distribution server and requests credentials for a peer. Signal does not dictate how this connection should happen, and we will trust that its outputs are correct; there is an out-of-band verification mechanism ("fingerprints") which we do not study. The key distribution server returns the long-term identity key g^{ik} of the peer, as well as a medium-term signed prekey g^{prek} and optionally (depending on the server's actions) an ephemeral prekey g^{eprek}.

After fetching credentials, the session initiator performs a Signal Key Exchange, a proprietary one-round key exchange protocol also referred to as "TripleDH" or "X3DH"[1]. While many possible variants of such protocols have been explored in-depth in the literature (examples include NAXOS [112], (H)MQV [106, 115], Kudla-Paterson [109], and SIGMA [107]), the session key derivation used here is new and not based on one of these standard protocols.

The initiator Alex begins the Signal Key Exchange computation by generating a new ephemeral key ek_A, and then performs three or four group exponentiations as depicted in Figure 2.3. The concatenation of the resulting shared secrets is then passed through a key derivation function ($\mathrm{KDF_r}$, Figure 2.5b) to derive initial shared secrets which are stored in the session state and used for message encryption.

From the initial shared secret, the initiator derives a symmetric value we call their sending "chain key".

[1] The key exchange protocol was sometimes referred to as TripleDH, from the three DH shared secrets always used in the Key Derivation Function (KDF) (although in most configurations four shared secrets are used). The name QuadrupleDH has also been used for the variant which includes the long-term/long-term DH value, not as might be expected the variant which includes the one-time prekey.

Alex attaches the ephemeral public key used for the initial Signal Key Exchange computation to the first Signal message, as well as identifiers for their medium-term and possibly ephemeral keys. On receiving this message, Blake retrives the private keys corresponding to the identity, signed pre-, and one-time pre-key which Alex used, subsequently performing the responder version of the Signal Key Exchange computation in order to derive the same shared secrets.

2.1.3.3 Exchanging messages

Sending For message encryption, agents derive two sequences of symmetric keys using a Pseudorandom Function (PRF) chain: one for sending and one for receiving.

To send a message in an existing Signal session, an agent A first retrives their latest sending chain key $ck_A^{\textbf{sym-ir}:x,y}$ from their session state. Recall that this notation indicates that A has performed x asymmetric updates, and y updates on this particular sending chain. A passes $ck_A^{\textbf{sym-ir}:x,y}$ through a key derivation function to derive two new values: an updated sending chain key and an encryption key $mk_A^{\textbf{sym-ir}:x,y}$ for the message.

$$\begin{aligned} ck_A^{\textbf{sym-ir}:x,y+1} &\leftarrow \mathrm{HMAC}_{ck_A^{\textbf{sym-ir}:x,y}}(\mathrm{const}_a) \\ mk_A^{\textbf{sym-ir}:x,y} &\leftarrow \mathrm{HKDF}(\mathrm{HMAC}_{ck_A^{\textbf{sym-ir}:x,y}}(\mathrm{const}_b), \mathrm{const}_1) \end{aligned}$$

The values $\mathrm{const}_\star$ are constant, public labels used to prevent type confusions. The constants const_1 and const_2 distinguish the invocations of HKDF used for root keys and chain keys, and the constants const_a and const_b distinguish the invocations of HMAC used to derive chain keys and message keys.

The sequence $ck_A^{\textbf{sym-ir}:x,y} \to ck_A^{\textbf{sym-ir}:x,y+1} \to ck_A^{\textbf{sym-ir}:x,y+2}$ forms Signal's symmetric ratchet, analogous to that of SCIMP. This key derivation is also depicted in Figure 2.5a.

Before sending the message, A retrieves their latest asymmetric "ratchet public key" from session state, or generates a new one if $i = 0$. They encrypt the message under mk^i using an authenticated encryption scheme and including the ratchet key as associated data, and send it over the untrusted network.

To send a subsequent message A repeats these steps with the same ratchet key: they apply a Keyed-Hash Message Authentication Code (HMAC) to their current sending chain key to update it and derive a new message key, encrypt the message with the derived message key and with the ratchet key as authenticated data, and transmit the resulting value.

Remark 1 ($mk \neq ck$)*.* One could imagine a simpler design in which chain keys ck were also used to encrypt messages. There are two benefits to Signal's approach. First, by deriving a separate value it enforces a form of key separation, which is generally a good cryptographic design principle: keys should only be used for a single purpose. Second, it enables out-of-order message receipt, as described in §3.4.1 on page 47.

Receiving Receiving and decrypting messages is very similar. When an agent receives a message they retrieve their latest receiving chain key from session state, apply the KDF to derive the next chain and message key and use the derived message key to decrypt the message. They also store the received ratchet public key in their session state.

Note that agents using Signal are able to decrypt messages received out of order, by applying the KDF multiple times and storing the resulting derived message keys in their session state until their respective messages are received. The open source implementation of Signal has a hard-coded limit of 2000 messages or five asymmetric updates, after which old keys are deleted even if they have not yet been used.

2.1.3.4 Asymmetric ratcheting

In addition to the symmetric updates to chain keys described above, new chains of chain keys are created for every message round trip using an asymmetric ratchet based on the ratchet keys.

Specifically, whenever an agent receives a new ratchet public key g^y, they in turn generate a new ratchet private key x' (replacing their old ratchet private key x), derive a DH shared secret g^{xy} between the latest pair of ratchet keys, and start two new message key chains:

$$\begin{aligned}
tmp &\leftarrow \text{ first half of HKDF}(rk_A^x,\ g^{xy},\ \text{const}_2)\\
ck_A^{\textbf{asym-ri}:x,0} &\leftarrow \text{ second half of HKDF}(rk_A^x,\ g^{xy},\ \text{const}_2)\\
rk_A^{x+1} &\leftarrow \text{ first half of HKDF}(tmp,\ g^{x'y},\ \text{const}_2)\\
ck_A^{\textbf{asym-ir}:x,0} &\leftarrow \text{ second half of HKDF}(tmp,\ g^{x'y},\ \text{const}_2)
\end{aligned}$$

Again, const_2 is a fixed, public value and this key derivation is also depicted in Figure 2.5b. The first chain, using the recipient's old ratchet key with the new ratchet key they just received, will be used as a receiving chain and its message keys used to decrypt incoming messages. The second chain, using the recipient's new ratchet key, will be used as a sending chain and its message keys used to encrypt outgoing messages.

2.1.3.5 Hiding Metadata

Some security protocols aim to hide more than just their secret keys or encrypted messages: they also want to conceal from an adversary even the fact that two people are communicating, or how much data they exchange, or when they do so. Tor [71] is probably the most famous member of this class of "metadata-protecting" protocols, and a number of messaging protocols [42, 93, 113] build on it. Protecting metadata is important because many adversaries are more interested in the metadata of communications than their actual content—consider for example a repressive goverment monitoring the conversation partners of known activists.

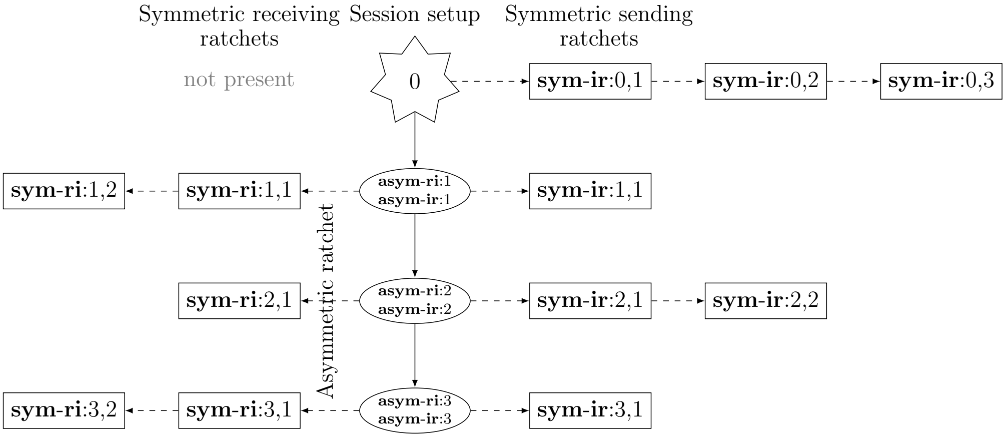
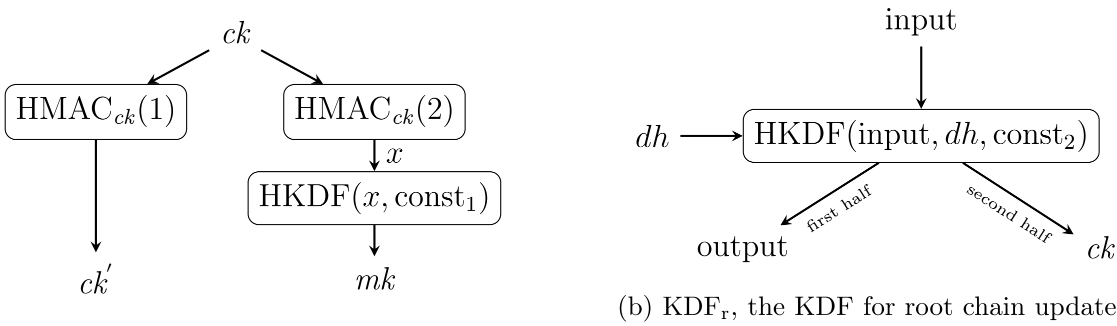


Figure 2.4: An example tree of *stages* that one party might use in one session of Signal. The content of each node is a name for the stage, defined in §3.2 on page 40. The vertical chain in the centre of the diagram represents the OTR-style asymmetric ratcheting, and the horizontal chains to each side the SCIMP-style symmetric ratcheting for deriving encryption and decryption keys. This figure is from [59].



(a) $\mathrm{KDF}_{\mathrm{m}}$, the KDF for message chain updates. Note that new chain keys are *not* computed using HMAC-based Key Derivation Function (HKDF); instead, they use only a HMAC.

(b) $\mathrm{KDF}_{\mathrm{r}}$, the KDF for root chain updates. dh is the DH value derived for this stage update, and the result is a new root key as well as an output chain key. Note that this KDF is used twice in Signal's root key updates: $rk \to tmp \to rk'$.

Figure 2.5: Key derivation functions for root and chain keys in Signal: keys flow along edges, and boxes apply their functions to their input. This figure is from [59].

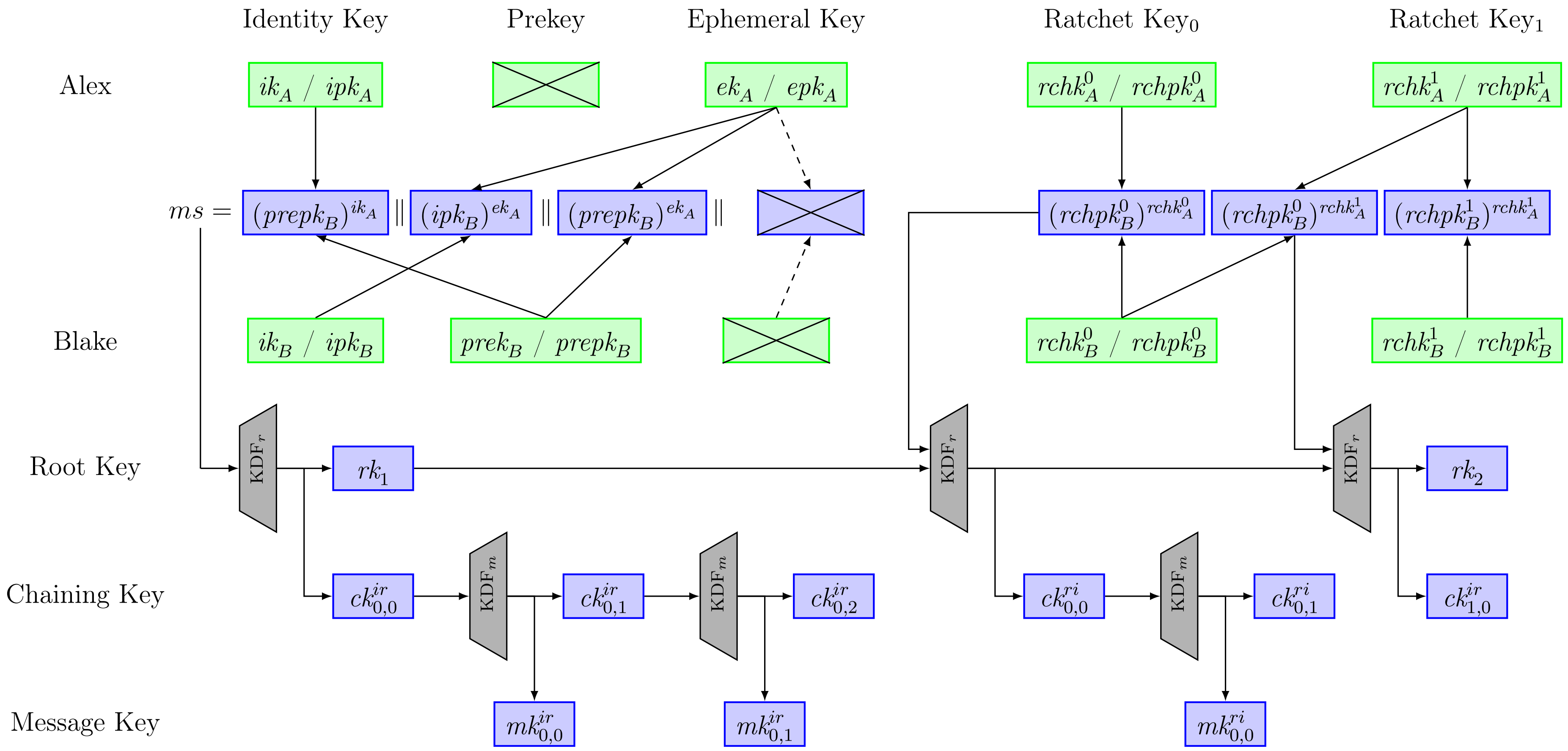


Figure 2.6: Depiction of the Signal Protocol's key schedule. Here, 🟩 denotes asymmetric keys that are generated by either party, and 🟦 denotes symmetric secrets that are derived from them. The top left quadrant is the initial handshake, and then the asymmetric ratchet continues in the top right quadrant; message and chain keys are shown in the bottom half. This figure is from [58].

However, protecting metadata presents many significant challenges over and above those of secure messaging. In order to hide communication partners from a network adversary, for example, systems must generate fake network traffic; otherwise, the adversary can identify communication partners by correlating incoming and outgoing messages. We see metadata protection as one of the major open problems in the field of secure messaging, but it is not one which we aim to tackle in this thesis.

2.2 Security Definitions

We hope that the readers of this thesis will already be firmly convinced of the need for rigorous security arguments, especially for security protocols. To avoid excessive choir-preaching, therefore, we refrain from making that argument again. Bellare [19], Goldreich [87] and Stern [160], among many eminent others, make much better arguments on the subject than the author could hope to achieve.

Similarly, in this section and afterwards we will use a fair amount of standard mathematical and cryptographic notation. Instead of boring the reader with endless definitions of sets, functions, groups, DH assumptions, encryption, signatures, and so on, we give brief definitions and further pointers in Appendix A.

2.2.1 Security Goals

Instead, we turn for a moment to the question of what properties we might aim to prove. There are many possible goals for an AKE protocol, and instead of trying to explain all of them in detail we define here a modern set of properties often considered necessary or important for AKE protocols used “in the wild”. We take some definitions from Boyd and Mathuria [38].

In an AKE protocol,

> a shared secret is derived by two (or more) parties as a function of information contributed by, or associated with, each of these, (ideally) such that no party can predetermine the resulting value. [133]

We will want session keys to be fresh i.e., newly generated, and known only to the two parties to the protocol. This latter is often called implicit (vs. explicit) authentication, since it guarantees that only the other party could know the key but does not promise that they in fact do know it.

We also wish to have some guarantees if some agents’ long-term keys are compromised.

Forward secrecy is achieved if compromise of the long-term key of a set of agents does not compromise the session keys established in previous protocol runs involving those agents. That is, after a session has completed and derived a key, learning the long-term key of an agent participating in that session does not help to compute the session key. Forward secrecy is generally achieved through an ephemeral DH key exchange.

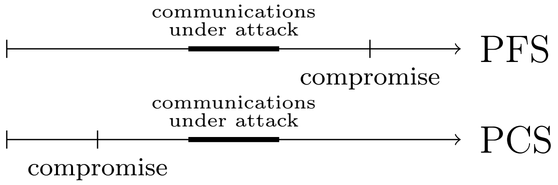


Figure 2.7: Attack scenarios of forward secrecy and PCS, with the communications under attack marked in **bold** and time flowing from left to right. Perfect Forward Secrecy (PFS) protects stages against later compromise; PCS protects stages against earlier compromise. This figure is from [57].

Key Compromise Impersonation (KCI) resistance is achieved when compromising the long-term key of an agent Alex does not allow the adversary to impersonate other agents *to Alex*. KCI attacks usually occur when a protocol does not sufficiently break symmetry, such that knowing either participant's key suffices to derive the session key. Basin, Cremers and Horvat [16] generalise KCI to Actor Key Compromise (AKC).

PCS [57] is achieved if compromise of the long-term key of an agent does not compromise the session keys established in subsequent protocol runs involving that agent. Thus, PCS is in a sense dual to forward secrecy, as depicted in Figure 2.7. PCS is impossible in general, because compromising an agent and immediately impersonating them produces the same observable outputs as if they had not been compromised. However, there are restricted scenarios in which it can be achieved.

2.2.1.1 Aside: Unknown Key Share attacks

Unknown Key Share (UKS) resistance [30] is achieved when a protocol is not vulnerable to the rather poorly-understood class of identity misbinding attacks known as UKS. In such an attack, the adversary $\mathcal{A}$ causes two agents Alex and Blake to share a key, but while Blake believes that the key is shared with Alex, Alex believes that the key is shared with $\mathcal{A}$. This disagreement can lead to practical attacks. For example, Blake might receive a message "you're fired" authenticated under the key shared with Alex, whereas in fact Alex intended to fire $\mathcal{A}$ not Blake. For historical reasons, this is often described as an attack on Alex, though in fact the true attack is on Blake, who has established a session with an honest peer. Resisting such attacks is generally good, though there is significant confusion in this area.

Frosch et al. identified a UKS attack against TextSecure arising from the fact that its keys are not bound to the identities intended to derive them. Since UKS attacks can be subtle, we explain the attack slightly differently. (The original description called the target of the attack $\mathcal{P}_e$ while we use Blake, and the peer to the attack $\mathcal{P}_a$ while we use Alex.)

In a UKS attack, the target of the attack is a session of an honest agent (Blake) that tries to communicate with another honest agent (Alex). Furthermore, in a UKS attack the adversary does not learn the session key: instead, it convinces Blake to accept a session key under false beliefs. Specifically,

(i) Blake accepts a key k to be used for communication with Alex,

(ii) Blake believes that Alex accepted k to be used for communication with Blake, but

(iii) Alex in fact accepted k to be used with a malicious agent Eli.

Thus, the beliefs held by Alex and Blake about their shared key k differ: Alex thinks k is for communication with Eli, but Blake thinks it is for communication with Alex.

Depending on the user interface and implementation details of the protocol, this can be a substantial authentication failure. Consider for example the scenario where Blake receives and accepts a message "you're fired" from a manager Alex, who actually intended to fire Eli. Here, the UKS attack on the protocol leads to Blake incorrectly accepting a message.

The UKS attack on TextSecure v3. In the UKS attack from [83, Figure 7], the adversary is $\mathcal{P}_b$ who aims to attack $\mathcal{P}_e$. When $\mathcal{P}_a$ tries to communicate with $\mathcal{P}_b$ as its peer, $\mathcal{P}_b$ lies about its public identity key and presents $\mathcal{P}_e$'s key as its own. Note that $\mathcal{P}_e$ is the target of the attack (not $\mathcal{P}_a$, who is communicating with a dishonest agent). The adversary subsequently reroutes $\mathcal{P}_a$'s messages to $\mathcal{P}_e$, who is the real target of the attack. Ultimately, $\mathcal{P}_a$ and $\mathcal{P}_e$ compute the same keys. $\mathcal{P}_e$ assumes that the keys are shared with $\mathcal{P}_a$, which is correct. However, $\mathcal{P}_a$ thinks they are shared with $\mathcal{P}_b$.

Preventing UKS attacks. The fix for UKS attacks can be trivial for libraries with access to agent identities: including both agents' identities in the key derivation function stops the attack. Intuitively, this fix achieves the same goal as prefixing each message with its intended recipient. Indeed, in the example above, if Blake receives a message beginning "Hi, Eli" it is clear that something has gone wrong.

However, UKS attacks are not prevented by the Signal core (i.e., the key agreement and Double Ratchet analysed in this chapter), roughly because its derived keys still do not depend on the expected identities of the communication partners. The UKS attack could be prevented at the application level, for example by including the assumed peer identities (e.g., the peer's phone number) in initial message exchanges.

2.2.2 Methodology

At a high level, there are two main ways to prove these types of guarantee about a particular protocol: symbolic and computational.

The first approaches [72, 139] to proving protocol security were symbolic, working in a term algebra where primitives are modelled as abstract function symbols. For example, a symbolic abstraction of symmetric encryption might be as formal functions `enc` and `dec` satisfying $\texttt{dec}(k, \texttt{enc}(k, m)) = m$ for all k and m. Protocol rules are also modelled in this term algebra, and the goal is to show that an adversary cannot construct e.g., a term used as a session key. If this is the case, and we make the "perfect cryptography assumption" [63], then we can rule out corresponding attacks on the real protocol.

Computational models, on the other hand, came about as a generalisation of the cryptographic games used to define security of primitives such as signatures and encryption. They do not make a perfect cryptography assumption, instead directly assuming hardness of the security games for the primitives they use. One can think of them as operating at a lower level, closer to functions and bits than the terms from symbolic models.

Neither formalism is inherently better, and when choosing between them there are a number of tradeoffs. Computational reductions rule out larger classes of attack, but are more complex to produce and verify; thus, they are generally applied to smaller systems. Our analyses of Signal and ART in this thesis are computational, but we argue in §6.2 that in fact we are reaching the limits of feasibility and that we should reconsider symbolic approaches for any further study.

2.2.2.1 Scope Reduction

Regardless of the formalism chosen, we have to strip out a lot of the complex parts of any protocol before it becomes amenable to study. We remark here on a few standard components of most security protocols which are usually excluded from their analyses.

Public Key Infrastructure (PKI) Almost all models assume that everyone knows their peers' public keys. To deploy a protocol studied in these models, there must be some way for agents to receive these public keys in a trustworthy fashion. For example, TLS uses the much-loved X.509 PKI, with all its attendant quirks.

Most messaging protocols nominate the service provider as a trusted third party for key distribution, so for example WhatsApp is trusted to distribute the public keys corresponding to a given phone number. (The Signal app includes an out-of-band "fingerprint" or "safety number" channel, but many users will not understand or perform this verification and we do not model it in our analysis.)

A malicious or coerced service provider could intercept Alex's messages by providing malicious keys, and even silently read or inject messages if they also replace Alex's keys for the recipient. A security proof which assumes a correct PKI does not say anything about this class of attack.

There are some new designs based on Google's original Certificate Transparency [114] which aim to remove some trust in the service provider. Perhaps the closest to being in widespread use is Key Transparency [89], which is in turn based on CONIKS [130] but with some changes made by Google; other examples include [17, 100, 153, 170, 171]. However, one downside of this family of solutions is that they are generally very complex, with a number of different subsystems: log servers, monitors, auditors, and gossip between clients. For most of them, this means that in fact the precise guarantees they provide are not simple to formalise[2], and therefore do not directly

[2] For example, using Key Transparency for multiple devices which share the same identity key

compose with security models' assumptions about their PKI.

Recent work on **detection** by Milner et al. [136] considers techniques for users to detect disagreement, for example, on the keys they have been provided. Their work could provide a simpler way for users to notice malicious key distributions under an adversary model similar to that of PCS.

Random Number Generators Many primitives and protocols require a source of uniformly random numbers, and security proofs generally assume that users' Random Number Generators (RNGs) correctly produce these. As above, RNGs must be instantiated with some concrete source of randomness in a deployment, and if they are not truly random then many guarantees may fail to hold.

For example, we'll see in §6.2.2.1 on page 111 that many signature algorithms require a random number as input, and an adversary who learns that random number and a valid signature can derive the private signature key. At the protocol level, sessions of both Signal and TLS are insecure without a correct RNG, even if the long-term keys were indeed generated randomly: an adversary who learns the outputs of either side's RNGs can compute their TLS master secret using the honest algorithm.

Some protocol models allow the adversary to learn or affect the output of agents' RNGs, which overapproximates certain types of side channel attacks. In our model of Signal we allow compromise of some RNG outputs, but not of those which are used to generate session keys under attack. Because we do not consider message encryption or the signatures on medium-term keys, we also do not deal with the random coins used in those algorithms.

Side channels Side channel attacks refer to a large class of attacks based on implementations of an algorithm instead of a weakness in the algorithm itself. Usually, side channel attacks leak information about the private inputs to some algorithm, based on features such as (but not limited to)

- the amount of time it takes to execute,
- the data it leaves in hardware or software caches,
- the power consumed during execution, or
- the electromagnetic radiation emitted.

Although there are a number of ways to measure and quantify resistance to side channel attacks, most widely-used protocol models do not consider them, restricting the adversary to "legitimate" communication channels with participants. However, many of the most significant reported vulnerabilities of the past few years fall into this category. Padding oracles are a particularly fruitful class, whereby either explicit error messages or a timing side channel reveal whether a ciphertext corresponds to a validly-padded plaintext.

may in fact require users to perform fingerprint verification [Jon Millican, personal communication].

2.2.3 Game-Based Models

Our computational models and reductions in this thesis are in the frameworks initially described by Bellare and Rogaway [23]; this type of model is often called game-based, B-R, or indistinguishability-based. Following Cremers [64], we define three core aspects: the honest execution model, the adversary model and the security property.

Execution Model The execution model describes how an honest participant executes the protocol with an intended peer, deriving a session key which they believe to be known only to the two of them. As part of this model, we must specify for example how to initialise a participant's state with long-term keys and identifiers, how to respond to unexpected or invalid messages, how to create a session and initialise its session memory, when to delete particular ephemeral keys, and so on.

We usually consider each agent as a set of *oracles*, one for each session in which they participate. That is, when Alex starts a session an oracle Π_1^{A} is created and initialised with Alex's long-term key, some random bits, and various other pieces of information. If Alex starts a second session a second oracle Π_2^{A} is created, and so on. In some models, the state of Π_2^{A} may depend on that of Π_1^{A}, though there is usually no direct communication between the two.

Adversary Model The adversary model describes the abilities of an active adversary, modelled as a Probabilistic Polynomial-time Turing Machine (pptm). The adversary is almost always permitted complete control over the network, able to intercept, drop, modify, replay or generate new messages at any point. In addition, we may grant additional powers to the adversary, often to learn the secret keys, random numbers or session state of a particular honest participant.

The adversary's abilities over and above local computation are encoded as *queries*, oracle invocations which return certain results depending on the state of the system. For example, to send a message to a given agent the adversary can issue a Send query, with return value given by the agent's response. Other queries may allow for creating new sessions at a given agent (Create), requesting public information (PublicInfo), creating sessions with poor randomness (CR-Create, [80]) and more.

A second class of queries reveals secret values from honest participants. These queries vary substantially between models, but usually include RevSessKey (revealing the session key derived in some session) and perhaps RevLTK (revealing the long-term key of an agent).

Security Property Finally, the security property states what should be achieved by the honest participants when interacting with the adversary. In B-R models this is encoded as two sub-properties:

(i) that honest participants communicating without an active attack should derive the same session key, and

(ii) if two honest participants derive a session key even under active attack, the adversary should not be able to distinguish that session key from a random string of bits.

The former property is generally a functional but not a security requirement: without it, the protocol does not perform its job as an AKE. Thus, while it is important, many analyses of existing protocols omit or only sketch its proof.

The latter property captures the core security requirement, and is encoded as a pair of queries Test and Guess, in the style of many cryptographic games. The Test query allows the adversary to select a particular session to attack, and depending on the value of a uniformly random bit b returns either a random bitstring or, if it exists, the session key of that session. The adversary must then query Guess(b'), winning if and only iff $b' = b$.

Remark 2 (On multiple Test queries). B-R models usually only allow the adversary to make a single Test query. One could imagine a model which instead allowed the adversary to make multiple Test queries and then choose any one of them to Guess. Brzuska et al. [44] argue by a hybrid argument that such a model is no stronger.

Although the queries vary between different models they are generally powerful enough that an adversary with unrestricted access can easily break any protocol. For example, no nontrivial protocol can be secure in a model containing unrestricted RevSessKey queries, since the adversary could pick a session, query RevSessKey against it to learn the session key, then Test it and compare the result to the revealed session key.

The security property is therefore not absolute but relative to a set of restrictions on the adversary: it encodes that *if* the adversary has not performed a "trivial" attack *then* it cannot Guess better than randomly. (Defining triviality here is not quite trivial [118].)

In addition to assumptions on the adversary's actions, we will also need assumptions on the cryptographic primitives used in the protocol, perhaps that the Decisional Diffie–Hellman (DDH) problem is hard in a particular group, or that some hash function is instantiated as a random oracle. The resulting theorems generally take the form of a probability bound on the advantage of an adversary in terms of advantages of other adversaries against the security games of the primitives. For example, we reproduce here a recent security theorem by Dowling et al. for TLS 1.3 draft 5:

Theorem. *[73, Theorem 5.2] The draft-05 full handshake is Multi-Stage-secure in a key-independent and stage-1-forward-secure manner with [certain authentication properties]. Formally, for any efficient adversary $\mathcal{A}$ against the Multi-Stage security there exist efficient algorithms $\mathcal{B}_1, \ldots, \mathcal{B}_7$ such that*

$$\begin{aligned}\mathsf{Adv}^{\textsf{\textit{multi-stage}},\mathrm{d}}_{\texttt{\textit{draft-05}},\mathcal{A}}(n) \leq 3n_s \cdot \Big(&\mathsf{Adv}^{\mathrm{coll}}_{H,\mathcal{B}_1}(n) + n_u \cdot \mathsf{Adv}^{\mathrm{euf-cma}}_{Sig,\mathcal{B}_2}(n)\\ &+ \mathsf{Adv}^{\mathrm{coll}}_{H,\mathcal{B}_3}(n) + n_u \cdot \mathsf{Adv}^{\mathrm{euf-cma}}_{Sig,\mathcal{B}_4}(n)\\ &n_s \cdot \Big(\mathsf{Adv}^{\mathrm{prf-odh}}_{PRF,\mathbb{G},\mathcal{B}_5}(n) + \mathsf{Adv}^{\mathrm{prf-sec}}_{PRF,\mathcal{B}_6}(n) + \mathsf{Adv}^{\mathrm{prf-sec}}_{PRF,\mathcal{B}_7}(n)\Big)\Big),\end{aligned}$$

where n_s is the maximum number of sessions and n_u is the maximum number of users.

We see that the theorem expresses the advantage $\mathsf{Adv}^{\mathsf{multi\text{-}stage,d}}_{\mathtt{draft\text{-}05},\mathcal{A}}(n)$ of an adversary $\mathcal{A}$ against their Multi-Stage game in terms of advantages against signature forgery, Pseudorandom Function Oracle Diffie–Hellman (PRF-ODH), PRF and hash collision games.

2.2.3.1 Partnering

One particular definition that has caused a fair amount of trouble is that of partnering, also known as matching. Partnering definitions attempt to capture when two agents' oracles are "intended communication partners" and should derive the same key. They are used in two ways in key exchange models. First, the correctness requirement is often phrased as requiring partnered sessions to derive the same key.

Second, and crucially, many restrictions against the Tested oracle must also apply to the partner of the Tested oracle, if it exists. For example, suppose that Alex has a oracle s whose peer is Blake, that Blake has a partnered oracle s' with Alex, and that s and s' have both completed and (by correctness) derived the same session key. An adversary which issues $\mathsf{RevSessKey}(s)$ and $\mathsf{Test}(s)$, comparing the results to answer Guess, will win the security game. We thus rule out querying Test and RevSessKey on the same oracle. But this is not enough: instead of issuing the query $\mathsf{RevSessKey}(s)$ an adversary can instead do $\mathsf{RevSessKey}(s')$, since by correctness that query returns the same value. We therefore must rule out issuing $\mathsf{RevSessKey}(s')$ for any oracle s' which is a partner to the Tested oracle s.

There are many subtleties to consider when choosing a definition of partnering, especially because choosing an overly-restrictive definition can rule out adversary actions which might correspond to true attacks. The traditional approach is through "matching conversations", where we say that two oracles are partners if all messages sent by each one have been received by the other. This can have some counterintuitive subtleties—for example, adding an irrelevant random string to the start of a message leads to insecurity in matching-conversations models[3]. Matching conversations are used in the B-R and Extended Canetti-Krawczyk (eCK) models as well as authenticated and confidential channel establishment (ACCE) (§2.2.4.1).

Some more recent models use "session identifiers" instead, introduced by Bellare, Pointcheval and Rogaway [22]. Here, each oracle is assigned an identifier or *sid* based on its actions, and two oracles are partners if and only if they have the same sid. If the sid is defined to be the sequence of sent and received messages then this definition can be made to reduce to matching conversations, but it can also include other public data such as identifiers and roles, or even secret data from the oracle's internal state.

[3] This is because an active adversary can modify the irrelevant data without detection, resulting in two oracles whose conversations differ only in the irrelevant section. These oracles are not considered partnered because of the different conversations, and hence the adversary can issue a RevSessKey query against the partner to the Tested session.

While session identifiers are expressive and flexible, they are often closely tied to the protocol under consideration, and thus can lead to protocol-specific models.

A handful of other partnering definitions have been proposed.

- Bellare and Rogaway [24] assume an external partnering function which reports whether two oracles are partnered (which can allow for "some unintuitive partnering functions" [22, Remark 1]).
- Kobara, Shin and Strefler [104] consider two oracles to be partnered if they have derived the same key and no other oracle has also derived it.
- Li and Schäge [118] consider two oracles to be partnered just if they have derived the same keys that they would have derived in the presence of a passive adversary.

This is an area of active research, and we expect more definitions to be presented over the coming years.

Our models in Chapters 3 and 5 use session identifiers for partnering, and thus are rather specific to the protocols under consideration (since the session identifiers, a feature of the *model*, are carefully constructed to include the relevant *protocol* data).

2.2.3.2 Freshness

Armed with a definition of partnering, it remains to encode the adversary restrictions into the model. Bellare and Rogaway [23] introduced the notion of a freshness (or cleanness) predicate for this purpose: a Boolean predicate on oracles whose return value determines whether the oracle is a valid target for a Test query. Their predicate is simple by modern standards: the adversary is allowed to issue a Test query against some oracle s just if

(i) it has accepted i.e., completed execution and derived a session key,
(ii) no RevSessKey query has been issued against it, and
(iii) if it has a partner, no RevSessKey query has been issued against its partner.

This allows them to include a RevSessKey query in their model without rendering it unsatisfiable.

For historical reasons, models' freshness predicates are usually where their security properties are encoded. This has led to something of an freshness complexity explosion as more and more cases are added. We depict some of the evolution of these predicates in Figure 2.8 on page 28, reproduced from [57].

Monolithic Properties Many properties which we would like to study are not of the form "some key k is secret". For example, we might want to prove that both parties agree on the holders of a session key i.e., if Alex derives a key for use with Blake, then Blake believes that the key is shared with Alex. This is really an authentication property, but because security is expressed in the B-R framework as indistinguishability of session keys from random we cannot directly express it as such.

Instead, it is encoded via the freshness predicate as a lack of restrictions on the adversary: if Blake derives a key k for use with Alex, but Alex believes k is for use with Charlie, then we will allow the adversary to

(i) query RevSessKey against Alex's session to learn k,

(ii) query Test against Blake's (fresh) session and compare the result to k, and

(iii) query Guess(b) where b is 1 iff k was the result of the Test query.

If on the other hand Alex believed that k was for use with Blake, then the freshness predicate would define Blake's session to be no longer fresh once RevSessKey has been queried on Alex's session. In this way, we encode authentication properties as part of the monolithic key indistinguishability property.

The downside of this approach is that the resulting security property tends to be very complex, and it is not always clear how the (lack of) existence of attacks corresponds to the clauses of the freshness predication. This is a weakness of current game-based models.

Recent models by Brzuska et al. [44] have begun to break apart monolithic properties into individual components. However, there is much more work to do in this direction.

2.2.3.3 Reductions: Game Hopping

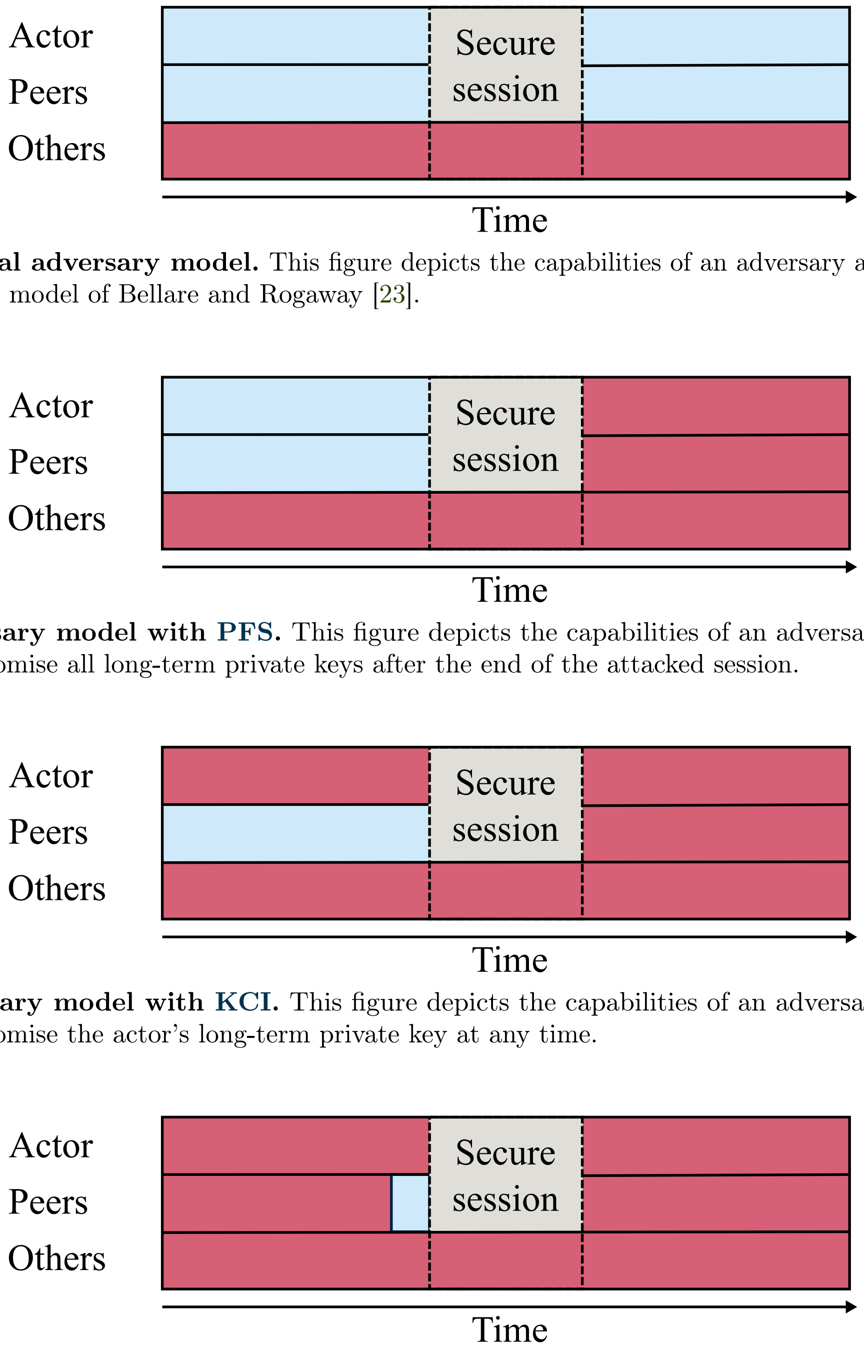
The B-R framework gives us a security experiment, but it remains up to us to show that the probability that an adversary succeeds in this experiment is negligibly better than random guessing. In order to prove this sort of bound, in this work we use the "sequence of games" technique described by Shoup [157], also called "game hopping".

Reductions by game hopping comprise a sequence of games $G_0, G_1, \ldots, G_n$ where G_0 is the security game on which we would like to establish security bounds. If S is some event in G_0 then we define events S_1 in G_1, S_2 in G_2 and so on, prove that $|\Pr[S_i] - \Pr[S_{i+1}]|$ is negligible for each $0 \leq i \leq n-1$, and deduce that $|\Pr[S] - \Pr[S_n]|$ is negligible. We then use this relationship to bound the advantage of the adversary in G_0.

Intuitively, we can think of each transition $G_i \to G_{i+1}$ as a modification of G_i that preserves the probability of some events we care about. We call each of these transitions a "game hop", and hop from game to game in turn until the probability of the event we care about is easy to compute. Shoup [157] identifies three main classes of these transitions: indistinguishability-based, failure-based and bridging.

Bridging transitions are simply administrative, and have $\Pr[S_i] = \Pr[S_{i+1}]$. Failure-based transitions rule out some "bad" event F: $\Pr[S_i] \wedge \neg F = \Pr[S_{i+1}] \wedge \neg F$. For example, game G_{i+1} might be defined to proceed as game G_i does except that it aborts if some set of values contains a hash function collision. It is easy to show from elementary probability arithmetic that in this case $|\Pr[S_{i+1}] - \Pr[S_i]| \leq \Pr[F]$. Thus, it is only required to show that F occurs with negligible probability.

Finally, the most interesting transitions are those based on indistiguishability. In this class of transition, we show that there exists a distinguisher $\mathcal{D}$ between two



(a) **Classical adversary model.** This figure depicts the capabilities of an adversary as defined in the original model of Bellare and Rogaway [23].

(b) **Adversary model with PFS.** This figure depicts the capabilities of an adversary that can also compromise all long-term private keys after the end of the attacked session.

(c) **Adversary model with KCI.** This figure depicts the capabilities of an adversary that can also compromise the actor's long-term private key at any time.

(d) **Adversary model with PCS through state** This figure depicts the capabilities of an adversary that can also compromise the peer's long-term private key before the completion of the Tested session.

Figure 2.8: Existing adversary models for security protocols. We depict a session in which an actor Alex is communicating with a peer Blake, and mark in red (dark) areas when it is permitted for the adversary to compromise a long-term key. Note that in the later models, there is no guarantee that a secure session will be able to take place, only that *if* the actor derives a key then it is not known to the adversary. Figure 2.8a represents a relatively weak model, and Figures 2.8b to 2.8d extend it with PFS, KCI resistance and PCS respectively. We remark that these figures are only intuitive: since sessions can run concurrently the dashed vertical lines do not really exist. Moreover, we have not depicted ephemeral or session-state reveals. These figures are from [57].

probability distributions P_1 and P_2 such that

$$\Pr[\mathcal{D}(x) = 1 | x \overset{\$}{\leftarrow} P_1] = \Pr[S_i] \quad \text{and} \quad \Pr[\mathcal{D}(x) = 1 | x \overset{\$}{\leftarrow} P_2] = \Pr[S_{i+1}]$$

We then show that P_1 and P_2 are computationally indistinguishable probability distributions, from which it follows that $|\Pr[S_{i+1}] - \Pr[S_i]|$ must be negligible. The easiest way to set up this sort of transition is to arrange that games G_i and G_{i+1} are in fact both based on a "hybrid game" $G_{i \to i+1}$ which takes an additional input, in the sense that $G_{i \to i+1} = G_i$ when the input is drawn from P_i. We can think of the hybrid game as sitting halfway in between G_1 and G_2.

A common use of an indistinguishability-based transition is to perform a game hop based on the hardness of a DH problem such as DDH. To do this, if G_i contains some DH shared secret g^{xy} we define G_{i+1} to proceed as in G_i except that g^{xy} is replaced by a random value g^z sampled from the DH group. Here, the hybrid game $G_{i \to i+1}$ takes a tuple (g^x, g^y, g^z) as its additional input, P_1 is the uniform distribution over all DDH tuples, and P_2 is the uniform distribution over all triples of group elements. When the additional input is sampled from P_1 we have that $G_{i \to i+1} = G_i$ and when it is sampled from P_2 then $G_{i \to i+1} = G_{i+1}$. Hence, if we assume that P_1 and P_2 are computationally indistinguishable i.e. that it is hard to solve DDH, then by the reasoning above we can conclude that $|\Pr[S_{i+1}] - \Pr[S_i]|$ is negligible.

Game hopping reductions tend to be complex and there are many edge cases to consider. Barthe et al. [14] describe a tool called EasyCrypt which is designed to mechanise the process of game hopping, formally proving the probability bounds between the different games. EasyCrypt allows for mechanised proofs and thereby higher confidence; however, it is generally much harder to produce an EasyCrypt proof than a corresponding pen-and-paper one, particularly when performing types of game hop which are not already understood by the tool. CryptoVerif [31, 32] is another tool producing computational protocol proofs. It is fully automated, requiring no user input to generate its proofs but meaning that it is hard to guide the tool in its search. Both EasyCrypt and CryptoVerif have been used to produce compututational reductions for practical protocols.

2.2.3.4 Whistlestop Tour of Models

We give a *very* brief overview of some key game-based models from the past few decades, beginning with the seminal paper of Bellare and Rogaway [23]. As with all subsequent models, in their definition an adversary creates oracles and sends messages to them, potentially asking RevSessKey queries and eventually choosing a fresh oracle to Test. An oracle is fresh here if it has accepted and if no RevSessKey query has been directed at it or any other oracle with which it has a matching conversation.

Shoup [156] makes an early distinction between adaptive corruption (revealing the long-term key of an honest agent) and strong adaptive corruption (revealing all secrets of an honest agent).

Bellare, Pointcheval and Rogaway [22] have two freshness conditions, the latter encoding forward security as rendering an oracle s unfresh if *any* RevLTK query was issued before the Test query and any message was sent to s. The authors note that this is overly restrictive, since a single RevLTK query renders all oracles unfresh even if they are unrelated to the corrupted agent. Separately, they also note a flaw in Bellare and Rogaway's original definition, namely that their model requires the adversary to Guess immediately after issuing a Test query, instead of permitting other actions to take place in between the two (which can correspond to real attacks).

Canetti and Krawczyk [50] include the freshness condition directly in their definition of security, forbidding adversarial queries against the Tested session or its partner until the session has completed, and subsequently allowing only RevLTK queries. Their model is widely-used and referred to as "CK"; it also includes a session-state query which reveals the "session state" of its target.

Canetti and Krawczyk [51] introduce the post-specified peer model, in which oracles do not know the identity of their peer when they are created but instead learn it at some point during their execution. They modify the CK model to allow for this technical change. Menezes and Ustaoglu [131] subsequently show that these changes are not quite trivial, giving examples of protocols which are secure in the pre-specified peer model but insecure with a post-specified peer and vice versa.

Krawczyk [106] gives a model ("CK-HMQV") tailored to their HMQV protocol. They include KCI attacks by allowing the adversary to issue a RevLTK query to the same oracle which it Tests, and UKS attacks by including agent identities in session identifiers. The same model also introduces "weak PFS", in which the adversary is permitted to reveal long-term keys only if it did not modify the messages in the Tested session.

LaMacchia, Lauter and Mityagin [112] further generalise to what is now known as "eCK", allowing corruption of any subset of ephemeral and long-term keys as long as not all of the actor or peer's keys are revealed. (For example, they consider an adversary which reveals the ephemeral keys of the actor to the Test session, and the long-term keys of its peer.)

Cremers [64] shows that CK, CK-HMQV and eCK are in fact formally incomparable, due in part to complex subtleties with respect to their adversary restrictions. One particular challenge is the use of ephemeral key reveal queries, which require a protocol to define precisely which parts of its memory are revealed by the ephemeral key reveal query. This motivates our later use of RevRand queries, which do not require protocol-specific definitions.

Feltz and Cremers [81] extend eCK with further queries modelling random number generator failures, and give stateful protocols which can be secure even if the adversary chooses some of their random numbers. Their adversary restrictions encode the intuition that a previous session of the same agent must have some good property.

Cohn-Gordon, Cremers and Garratt [57] capture PCS by allowing the adversary to issue a RevLTK query to the partner of the Tested oracle under certain conditions, building on the restrictions of Feltz and Cremers [81] for stateful protocols. Our Signal models are based on these.

Multi-stage models Google's Quick UDP Internet Connections (QUIC) (and later TLS 1.3) have adopted a design which first negotiates a weak key and later upgrades it. This allows for data to be sent alongside the first protocol message, thus reducing initial connection latency. In order to capture these multiple stages, Fischlin and Günther [82] introduce the concept of *multi-stage* key exchange.

Multi-stage models differ from those above in that sessions no longer derive a single session key. Instead, they proceed through multiple *stages*, with each stage deriving a separate stage key. The adversary need only distinguish a single stage key from random in order to win. These models were originally used to capture only two stages, but they allow for arbitrary numbers of stages. We will adopt them later in order to capture Signal's long-lived sessions.

2.2.4 Alternative Approaches

The models we described above all encode key *indistinguishability*: they guarantee that an adversary cannot tell the difference between a random value and the actual key produced by a protocol run. This is a technical property, and not quite the same as what we might actually want for a messaging protocol.

In particular, it does not prove that messages which are received under the key derived from a secure AKE protocol are themselves secret or authentic. Indeed, it is not in fact hard to construct an AKE protocol which is secure on its own, but insecure when additionally messages are sent encrypted under the session key it derives.

Remark 3 (AKE protocols with insecure composition)*.* A simple example is a protocol which includes a confirmation message such as $\{A, B\}_k$ encrypted under the session key k. (Such a message might for example serve to confirm that A's intended peer is B, preventing UKS attacks.) Suppose that the session key is subsequently used to encrypt data, and that data could take the form of a pair (A, B). An attacker could then transmit the encrypted data $\{A, B\}_k$ as a confirmation message to an honest peer B, enabling a UKS attack.

Here, the error is that the protocol uses its session key directly for identity confirmation. Protocols which wish to use confirmation messages should instead derive a separate handshake key which is only ever used for these messages; this prevents it from being confused with their session key.

Symbolically, this property generally follows fairly easily from secrecy and authentication of the session key itself. In the game-based world, there are two main approaches to prove message secrecy: monolithic models and composition theorems.

2.2.4.1 ACCE-style models

Key indistinguishability is a very strong condition, and in particular it forbids protocols from directly using the session key to encrypt a key confirmation message. For example, consider the simple protocol depicted in 2.9 for key confirmation after

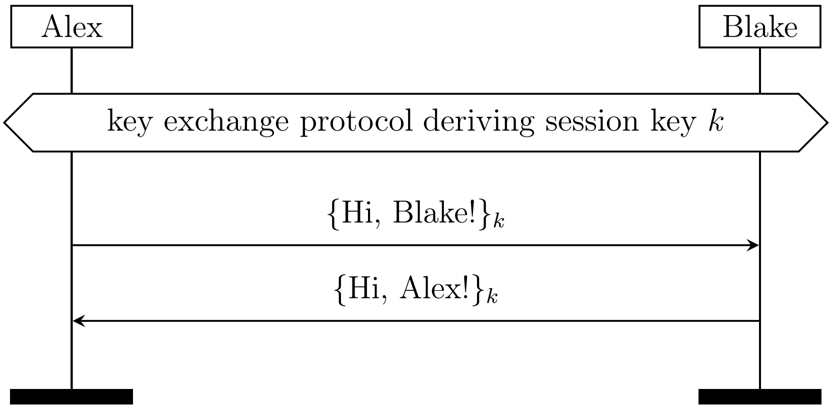

Alex
Blake
key exchange protocol deriving session key k
$\{\text{Hi, Blake!}\}_k$
$\{\text{Hi, Alex!}\}_k$

Figure 2.9: Simple protocol demonstrating the problem with indistinguishability of keys as a security notion when they are also used in protocol messages. $\{\cdot\}$ denotes authenticated encryption.

a key exchange. Alex and Blake perform some key exchange to derive a secret key, and then exchange authenticated messages to confirm that they both know it.

Regardless of the key exchange protocol in the first step, this protocol is insecure in (most) B-R models: there is a simple adversary which can break key indistinguishability in all cases. To do so, it runs a single session between any two parties and issues a Test query against that session at either party, receiving a key k_t in response. It then decrypts the final message $\{\text{Hi, Blake!}\}_k$ under k_t. If the result is a decryption failure it knows that k_t was random, and if not it knows that k_t was the actual session key. It thus can distinguish the session key from random, breaking indistinguishability.

This is clearly not a particularly interesting attack: it is an artefact of the model, specifically the interaction between

- the encoding of key secrecy as indistinguishability from random, and
- the use of session keys in a way visible to the adversary.

However, since many protocols (including both of those discussed in this thesis) wish to *use* instead of just export the keys they derive, we would like some way to ignore or work around this class of attacks. There are two main approaches.

The most common approach is to modify the protocol by removing usages of the session key, and update the security goals to match. For example, in the scenario above we might remove the key confirmation messages and prove implicit authentication (only Blake could know k) instead of explicit (Blake and only Blake knows k). In protocols which use the key for authentication, say by using it to compute a Message Authentication Code (MAC), we might enforce authentication of the resulting message through the model, proving a statement such as “if the authenticated message is received as it was transmitted then the security goal holds,” instead of based on the security of the MAC. This approach is simple, clean and allows us to use standard B-R definitions of security. However, it is of course rather undesirable in the sense that we are no longer directly proving the security of the protocol under consideration but instead of a "related" protocol which we heuristically argue is good enough. This is the approach we take throughout this thesis.

An alternative and relatively recent idea is to explicitly capture the use of the key in a monolithic *secure channel* notion. If the protocol has a clean separation between the key derivation and usage, it may be possible to prove security of the channel through a composition theorem. Alternatively, one can use a model such as ACCE [96], which has two phases in each session:

(i) first, agents communicate and derive a key as in a traditional B-R model; and
(ii) second, agents use the key to encrypt messages to each other.

In ACCE the adversary is allowed to choose either of these phases to attack; that is, there are two ways it may win the security game. It can win either by breaking key indistinguishability in the B-R sense (but without access to the encrypted messages), or it can win by breaking the properties of the secure channel in the second phase.

Quoting from the designers of the first ACCE model, used for an analysis of TLS 1.2:

> We do not want to propose ACCE as a new security notion for key exchange protocols, since it is very complex and the modular two-step approach approach seems more useful in general. (Jager et al. [96])

Moreover, Signal uses the session key to authenticate not just data but also handshake messages of later stages, meaning that even ACCE models would face challenges in defining the boundary between the first and second stages.

In general we view ACCE models as a necessary evil, but hope that future work will provide composition theorems (as described below) which are general enough to apply to all the protocols which we wish to analyse.

2.2.4.2 Composition Theorems

Various authors have proved composition theorems, showing that a combined system which

(i) runs a key exchange protocol to derive a shared symmetric key, and then
(ii) uses that key to exchange encrypted data

enjoys some security property.

Bellare and Namprempre [21] and Krawczyk [108] study the composition of generic AKE protocols with specific notions of encryption, in the former case Encrypt-then-MAC and in the latter case authenticated encryption. This allows them to prove security of specific compositions; however, they do not study *generic* symmetric key protocols.

Brzuska et al. [43] show a powerful composition theorem: if a protocol is secure in their B-R-style model, and moreover meets a certain condition which they call "public session matching", then it is still secure when composed with an arbitrary symmetric key protocol.

This is a very useful result. However, it does not solve the problem above, that protocols such as TLS 1.2 are not in fact secure in a B-R-style model due to their use of session keys as handshake messages. Brzuska et al. [44] tackle this problem by aiming for a weaker pre-condition, requiring instead of B-R security a property related to key usability. However, their result is also inapplicable to our model since we consider a stronger adversary (who can e.g., learn random values generated by certain agents).

Although we do not use composition theorems for the results in this thesis, we foresee significant future work in their application to messaging protocols.

2.2.4.3 Metatheorems for Computational Reductions

Various strands of work have attempted to make protocol constructions more modular, versus the usual monolithic definitions. Bellare, Canetti and Krawczyk [20] give an early example of this approach, defining compilers which convert protocols secure in a weak model to protocols automatically secure in a stronger model.

This approach can only give results about the protocols which are output by one of these compilers. Kudla and Paterson [109] aim to prove security of a wider class of protocols by using modularity in the reductions instead of in the constructions: they show that under certain conditions it suffices to prove security of a protocol in a much weaker than realistic model.

In general, such metatheorems can give simple proofs where they are applicable, but often fail to apply to new designs such as those of Signal.

2.2.4.4 Computational Soundness

Abadi and Rogaway [1] pointed out that there is a gap between symbolic and computational approaches, and began the study of "computational soundness". Their goal was to give a symbolic formulation with a soundness property, namely that if there exists a symbolic proof then there exists a corresponding computational one. In other words, if their result applies, in order for a protocol to enjoy computational security it is enough to prove it secure symbolically.

Computational soundness theorems have many powerful features: it can often be much easier to give a symbolic proof of a complex protocol than a computational one, and the resulting proofs are generally much more comprehensible. Moreover, they centralise effort: instead of requiring a separate complex proof for every protocol, it is enough to prove a single computational soundness theorem which can then be applied in many different domains.

However, these theorems also often have a number of downsides. The strongest results today often still omit primitives (e.g., they might only apply to protocols using just symmetric encryption and DH), or require that all messages be strictly tagged and do not include key cycles. The most widely-used tools for mechanised symbolic verification also do not enjoy computational soundness. These restrictions, and the level of complexity in applying such a theorem, mean that such proofs are often not a good choice for more complex protocols.

Computationally Complete Symbolic Attacker The "computationally complete symbolic attacker" is a recent line of work initiated by Bana and Comon-Lundh [10] and continued in [8, 9, 11, 12]. They give an alternative way to prove computational soundness in which, very roughly, they provide only *restrictions* on the adversary, and then allow all other actions. Their main theorem is a (bounded-session) computational soundness result.

2.2.4.5 Ideal-Real Approaches

Another popular formulation for computational security definitions is the ideal-real flavour of definition as initially presented by Shoup [156]. Here, we define security for a system with an "ideal functionality" describing how it might behave in an idealised world not limited by the constraints of practicality. For example, an ideal functionality for a signature scheme might just remember the list of all messages ever signed by any key, so that to verify a signature one need merely consult this list.

To prove security of a real system in this paradigm, we give a *simulation* argument showing that there is no way to distinguish between interacting with the real system and interacting with the ideal one. It follows that any attack on the real system must also lead to an attack on the ideal one, since otherwise the success of the attack would lead to a distinguisher. If formulated correctly, attacks should be obviously impossible against the ideal system, and so security of the real one follows.

There are various advantages of ideal-real formulations. In particular, they can allow for composition results with much less pain than game-hopping definitions. However, in our context there are a number of drawbacks.

Choosing the right ideal functionality To define simulation security for some system we must choose an ideal functionality which captures how it "should" work. For some primitives this is fairly straightforward, although not quite trivial. However, for more complex systems the ideal functionality must encode more and more, and it can often be hard to choose one that works as required.

For a complex protocol like Signal, defining the ideal functionality alone is a significant challenge. Indeed, a functionality for DH was only defined in 2017 [111], after our initial analysis of Signal.

Complex threat modelling Signal is designed to be secure against a complex threat model: for example, it is meant to provide key secrecy if an adversary learns a current chain key and tries to derive earlier message keys, but not if the adversary can compromise a medium-term signed prekey [105].

Expressing properties like this in game-hopping models is fairly straightforward if complex: they are encoded into the different cases of a freshness predicate. It is not so easy to encode them in ideal-real models, because the latter do not directly refer to execution traces and thus do not allow us to easily express time-based trace properties such as the above.

Strong requirements Simulation security is in general a very strong property, and many practical protocols do not in fact meet it, even though they do not have any practical attacks [43]. For example, Canetti and Fischlin [48] show that no commitment protocol can be secure in the Universal Composability (UC) framework.

Shared state Until recently no security models captured long-lasting sessions like Signal's, instead modelling separate key derivations as coming from separate handshakes. Since Signal key derivations share secret values between different protocol runs, we would need a simulation-based model that allows shared state, such as that of Canetti and Rabin [52].

Using such a model requires using functionalities that work in the joint-state model, which are not always easy to find. In contrast, in the game-based setting the multi-stage framework of Fischlin and Günther [82] allows for easy modelling of long-lived sessions deriving multiple keys.

Operational interpretation Finally, the operational interpretation of security models in the game hopping context is generally clearer, which helps us to understand the intricate execution traces of certain attacks.

The Universal Composability (UC) models [110, 138, 145, 152, 167] are a family of ideal-real models which have a particularly powerful security property not shared with game-based definitions: their security holds even when composed with other systems. This is an excellent property to have, and one which is in general false for game-based security reductions.

There are a wide range of other ideal-real security definitions: "GNUC" [92], "reactive simulatability" [5, 143], "abstract cryptography" [127], "constructive cryptography" [126] and others. A form of UC has even been considered in the symbolic model [33].

Ideal-real approaches are the main alternative to game-based security models. For the reasons described above, our reductions in this thesis are in the latter.

CHAPTER 3 SIGNAL: FORMAL MODELLING

This chapter is based on the formal model from

> *Katriel Cohn-Gordon, Cas J. F. 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 2017, Paris, France, April 26-28, 2017. *IEEE, 2017, pp. 451–466.* DOI*:* *10.1109/EuroSP.2017.27.* URL*:* https://doi.org/10.1109/EuroSP.2017.27

The paper was joint work throughout, but the majority of my contributions are described in this section: I deduced the precise definition of Signal from reading the Java implementation, giving it as a mathematical specification, and (with Douglas) built the model to capture it.

In this chapter we aim to provide a formal framework for the Signal Protocol, with (i) a careful definition of the protocol itself, and (ii) a formal, game-based model which defines the security guarantees we believe that it should provide. Since at the time of analysis Signal did not claim any security properties, we begin this chapter with an informal threat model (§3.1), which we will use to motivate our formal definitions. We then state these properties formally in a multi-stage key exchange model (§3.3), and redescribe the Signal protocol itself in the language of the model (§3.4). By the conclusion of the chapter, we can formally define the Signal protocol and a security experiment for it.

In this and subsequent chapters, we take standard cryptographic definitions as given. We refer the reader to Appendix A on page 129 for an overview of our notation for e.g., probabilities, negligibility, signatures, symmetric and asymmetric encryption, groups, DH assumptions and so on.

3.1 Threat Model

A formal security model gives a precise statement: some class of attacks is not possible against protocols which meet the model. However, there is no obvious candidate for which class of attacks we should rule out: it depends on context.

For example, consider a client C accessing a web server S using the TLS protocol. A security model for this situation might wish to cover malicious web servers attempting to impersonate S to C, aiming to prove that this is not possible. However, it will generally *not* cover malicious clients connecting to S: in its most common deployment scenario, TLS only authenticates the server to the client. Other attacks to exclude might for example be those in which a malicious certification authority colludes with a web server.

Similar considerations apply to the Signal protocol: there are some attacks which it should prevent, and some which by design it does not, but there is a "gray zone" in between. In order to pin down this class more precisely we take inspiration from the documentation but in the end must make a judgment.

The README accompanying Signal's Java source code [119] states that Signal

> is a ratcheting forward secrecy protocol that works in synchronous and asynchronous messaging environments.

A separate GitHub wiki page [142] provides some more goals (forward and future secrecy, metadata encryption and detection of message replay/reorder/deletion) but to the best of our knowledge no mention of message integrity or authentication is made other than the use of authenticated encryption. We remark here that this wiki page defines "future secrecy" to mean that "a leak of keys to a *passive* eavesdropper will be healed by introducing new DH ratchet keys", emphasis ours.

Initial Assumptions We begin by stating some initial assumptions.

First, we will assume a Dolev-Yao [72] style network, in the sense that "the adversary carries the messages". That is, we make no assumptions on the confidentiality, integrity or reliability of the network channels between participants. Our participants will instead send messages directly to, and receive responses from, the adversary. This is a very standard overapproximation of a network adversary.

Second, we will assume an honest public key infrastructure, in the sense that each participant receives the true public keys of all other agents in the system, honest and malicious. These public keys are used in the protocol for authentication. In a real deployment this assumption is not easy to satisfy, and there are models which make weaker assumptions e.g., [39]. However, for now we will take it as a given and discuss later the ways in which it may be implemented. We remark in particular that as part of the public key infrastructure we will assume a trusted distribution mechanism for Signal's signed prekeys, and that while Signal specifies a mandatory out-of-band mechanism for participants to check the validity of public keys they receive, most implementations do not require such verification to take place before messaging can occur.

Third, we will not consider side channel attacks or implementation errors in our model. In particular, deletion of data from flash-based storage media is known to be a hard problem [148], but we will assume that all agents have the ability to delete old secrets.

Finally, we will assume that all parties generate ephemeral keys honestly but that these keys may be corrupted. In particular, we assume that all honestly-generated random values are sampled uniformly at random from their input space. (There are alternative techniques to model imperfect random number generation; for example, Feltz and Cremers [81] allow the adversary to choose honest participants' random values.) Note that Signal sessions are insecure if both parties' ephemeral keys are corrupted.

Standard Security Properties The TLS 1.3 standard [149, Appendix E] discusses the following properties.

Establishing the same session keys The handshake needs to output the same set of session keys on both sides of the handshake, provided that it completes successfully on each endpoint.

Uniqueness of the session keys Any two distinct handshakes should produce distinct, unrelated session keys. Individual session keys produced by a handshake should also be distinct and unrelated.

Secrecy of the session keys The shared session keys should be known only to the communicating parties and not to the attacker.

Peer Authentication The client's view of the peer identity should reflect the server's identity, and the server's view of the peer identity should match the client's identity.

Downgrade protection The cryptographic parameters should be the same on both sides and should be the same as if the peers had been communicating in the absence of an attack.

Forward secret with respect to long-term keys If the long-term keying material is compromised after the handshake is complete, this does not compromise the security of the session key, as long as the session key itself has been erased.

KCI resistance Peer authentication should hold even if the local long-term secret was compromised before the connection was established. (That is, if Alex's secret key is compromised, it should not be possible to impersonate Blake to Alex.)

Protection of endpoint identities The server's identity (certificate) should be protected against passive attackers. The client's identity should be protected against both passive and active attackers.

We will interpret most of these properties in the context of Signal, aiming at a high level for secrecy and authentication of message keys. Correctness (establishing the same session keys) is simple to prove, and uniqueness of session keys will be implicitly required in our model, following from session identifiers in the proof.

Instead of considering forward secrecy and KCI resistance as explicit goals we will require secrecy and authentication to hold under a variety of compromise scenarios, including if a long-term secret has been compromised but a medium or ephemeral secret has not (forward secrecy). These scenarios will also capture PCS.

Signal, unlike TLS, has only a single version, and clients must speak the same version to communicate. In particular, there is no way to negotiate the particular primitives or protocol version used in a Signal connection, and thus downgrade protection is not a threat.

Some claims have been made about privacy and deniability [162] in Signal, but these are relatively abstract. In general, signatures are used but only for the signed pre-key in the initial handshake, meaning that an observer can prove that Alex sent a message [69, full deniability] to *someone* but perhaps not to *Blake* [66, peer deniability]. We do not aim to prove such properties, and thus do not consider endpoint identity protection as a goal.

3.2 Notation

Stages Within a session, Signal admits a tree of various different stages (Figure 2.4 on page 16). We refer to all stages by identifiers in [square brackets], defined below. Stages are local to each agent, but correspond in the sense that stages at intended communication partners will derive the same key. The initial stage contains the Signal handshake and subsequent stages capture the ratcheting key derivations.

Agents have different interpretations of stages which derive the same keys. In particular, the initiator of a session may consider some stage s as generating a key for encrypting messages to send. The responder to that session may have a stage s' deriving the same key as s, but will use that key to decrypt received messages. To avoid making repeated case distinctions between initiators and responders, we adopt a *role-agnostic* naming scheme, describing stages as "**-ir**" if they are used for the initiator to send to the responder, and as "**-ri**" if they are used for the responder to send to the initiator. We will thus maintain the invariant that stages with the same name generate the same key.

The specific types of stage which we define are

- for the initial handshake: `triple` or `triple`+`DHE` depending on whether the optional ephemeral-ephemeral DH exchange was included in the key derivation
- for asymmetric ratcheting: `asym-ir` or `asym-ri`
  The former uses a received ratchet key and begins a receiving chain, and the second generates a new ratchet key and begins a sending chain. At a given party, we count the number of asymmetric updates in a variable x; thus, we can refer to the x^{th} update of the first type in a session as stage [**asym-ri**:x], and of the second type as [**asym-ir**:x]. Note that the x^{th} "**-ri**" stage precedes the x^{th} "**-ir**" stage. This is because the first asymmetric stage is of type "**-ri**", in turn because the responder performs the first asymmetric ratchet update.
- for symmetric ratcheting: `sym-ir` or `sym-ri`
  The type of a symmetric stage depends on whether its chain was created by a stage of type [**asym-ri**:x] or [**asym-ir**:x]. At a given party, we count the

number of symmetric updates in the x^{th} symmetric chain in a variable y; thus, we can refer to the y^{th} update in the x^{th} symmetric chain as stage [**sym-ri**:x,y] or [**sym-ir**:x,y].

The full definitions of each stage type are in §3.4.2.

In our model, there are no stages [**sym-ir**:x,0] or [**sym-ri**:x,0], but there are keys with these indexes, since the first entry in each sending and receiving chain is created by the asymmetric update starting that chain (see Figure 2.4). We could equivalently think of Signal only deriving message keys in symmetric stages and allowing $y = 0$, in which case asymmetric stages would not derive message keys. Our formulation simply renumbers keys, so that every stage derives a message key.

Session identifiers For partnering of Signal sessions we use session identifiers as described in §2.2.3.1 on page 25, generalising to "sids" since each stage has a distinct identifier in our multi-stage model. Sids are defined recursively in Table 3.1, with tags distinguishing between the different types of stage. For example, if the first stage was conducted without the optional one-time prekey then we define its sid to be $(\texttt{triple} : \mathit{ipk}_i, \mathit{ipk}_r, \mathit{prepk}_r, \mathit{epk}_i)$, where i and r are the identities of the initiator and responder respectively. It is important to note that these session identifiers only exist in our model, not in the protocol specification itself. If two stages have equal session identifiers we say that they "match."

The precise components of the session identifiers are crucial to our definition of security: including more information in the identifier restricts which stages are considered to match a given target stage. Since the adversary is forbidden from certain actions against such matching stages, including more information in sids weakens the restrictions on the adversary and therefore captures more attacks, leading to a stronger security model.

In particular, we do *not* include identities in Signal's sids, because they are not included in Signal's key derivation or associated data of encrypted messages. This means that stages can be partners even if they disagree on each other's identities. Concretely, the unknown key share attack against TextSecure [83] is not considered an attack in our model: if an adversary performs that attack then Alex's session with Eli will have the same session identifier as Blake's session with Alex, and thus Alex's and Blake's sessions will match.

State Each Signal session and stage stores some public and secret values in its local memory. We call the collection of these values the state and denote by π_u^i the state of the i^{th} session of agent u.

Definition 1 (State)**.** The *state* π is a collection of variables stored by a protocol agent. We write π_u^i for the state of the i^{th} session of agent u. For variables v which change per-stage, we write $\pi_u^i.v[s]$ for the value of v in stage s of u's ith session. Note that s is not a natural number but a sid.

- $\pi.\mathit{role} \in \{\texttt{init}, \texttt{resp}\}$: the instance's role
- $\pi.\mathit{peeripk}$: the peer's long-term public key

name	sid	
$sid[0]$	$(\texttt{triple} : ipk_i, ipk_r, prepk_r, epk_i)$	if $type[0] = \texttt{triple}$
	$(\texttt{triple+DHE} : ipk_i, ipk_r, prepk_r, epk_i, eprepk_r)$	if $type[0] = \texttt{triple+DHE}$
$sid[\textbf{asym-ri}{:}x]$	$sid[0] \parallel (\texttt{asym-ri} : rchpk_i^0, rchpk_r^0)$	if $x = 1$
	$sid[\textbf{asym-ir}{:}x-1] \parallel (\texttt{asym-ri} : rchpk_r^{x-1})$	if $x > 1$
$sid[\textbf{asym-ir}{:}x]$	$sid[\textbf{asym-ri}{:}x] \parallel (\texttt{asym-ir} : rchpk_i^x)$	if $x > 0$
$sid[\textbf{sym-ri}{:}x{,}y]$	does not exist	if $x = 0$
	$sid[\textbf{asym-ri}{:}x] \parallel (\texttt{sym-ri} : y)$	if $x > 0$
$sid[\textbf{sym-ir}{:}x{,}y]$	$sid[0] \parallel (\texttt{sym} : y)$	if $x = 0$
	$sid[\textbf{asym-ir}{:}x] \parallel (\texttt{sym-ir} : y)$	if $x > 0$

Table 3.1: Definition of session identifiers $sid[s]$ for an arbitrary stage s. Since our stages are named role-agnostically, the definitions for initiator and responder stages coincide; we use i to refer to the identity of the initiator and r for that of the responder. For example, if Alex believes Blake is the peer, then ipk_i denotes Blake's identity public key and ipk_r denotes Alex's. The initial asymmetric stage sid contains two ratchet keys (instead of one) since they are not used in the initial session key derivation and thus are not contained in $sid[0]$. We note that $sid[\textbf{sym-ir}{:}x{,}y]$ for $x = 0$ does not exist because the receiver never starts a symmetric chain immediately after the handshake, always first performing a DH ratchet. This figure is from [59].

- $\pi.peerprepk$: the peer's medium-term public key
- $\pi.status[s] \in \{\varepsilon, \texttt{active}, \texttt{accept}, \texttt{reject}\}$: execution status for stage s, set to `active` upon start of a new stage, and set to `accept` or `reject` when the stage computes its key
- $\pi.k[s] \in \mathcal{K}$: the key computed in stage s
- $\pi.sid[s]$: the identifier of the current session s
- $\pi.type[s]$: a protocol-defined value describing the type of the current stage
- $\pi.st[s]$: additional protocol-defined state values computed in a previous stage and given as input to stage s

The state π of an instance models "real" protocol state that might be found in the memory of a protocol implementation. We will need additional state for administrative reasons in the security experiment, for example to record the set of honestly-generated DH keys; this state will be given in Definition 4.

3.3 Formal Security Model

We now capture our threat model for Signal in a formal game-based model. We use a multi-stage model as defined in §2.2.3.4, but with a tree of stages instead of a linear sequence, as seen in Figure 2.4. As with all game-based models, ours allows multiple parties to execute multiple, concurrent *sessions*; since it is multi-stage, each session has multiple *stages*. For Signal, sessions will represent a long-lasting conversation between two agents, and stages will represent a single ratchet application, either symmetric or asymmetric.

Generality It is possible to write a game-based model at various levels of generality. A very general model allows many similar protocols to be described in the same framework and thus allows for comparisons between them. A specific model, on the other hand, allows for security guarantees carefully tailored to the protocol under consideration.

Signal includes various features which are not normally captured in general models for key exchange security (in particular: medium-term keys and trees of stages). We thus cannot hope for a fully general model. However, we keep the overall structure of our definitions relatively independent of Signal. Our cleanness predicates will then capture the fine-grained assertions we prove about Signal specifically.

Model notation Instead of writing our model in prose, we use the cryptographic notation of a tuple of pseudocode algorithms with which an adversary may interact. Since our games are relatively complex and include many details (such as the precise definitions of session identifiers), we believe that the pseudocode notation makes it easier to follow subtleties in the games.

We use the following typographical conventions:

- Monotype text denotes constants.
- Serif text denotes algorithms associated with the *protocol*.
- *Italicized* text denotes variables associated with the protocol.
- Sans-serif text denotes algorithms, oracles, and variables associated with the *experiment*.
- Algorithms and Oracles start with upper-case letters; variables start with lower-case letters.
- Object-oriented notation represents collections of variables: $x.y$ represents the value of the variable called y in the collection x.

DH protocols conventionally use both ephemeral keys (unique to a session) and long-term keys (in all sessions of an agent). Signal's prekeys do not fit cleanly into this separation, and in order to follow the conventions of the field we refer to the reused DH keys as "medium-term keys".

Remark 4 (Match-security). Following Brzuska et al. [43], Fischlin and Günther [82] define a security requirement for session matching called "Match security" which can be used when composing a key exchange protocol with a symmetric key protocol.

Signal can admit more than two sessions with the same session identifier (in the case that Blake reuses the same medium-term key without a one-time key), so their definition is not immediately met. A weaker result, that the sid uniquely determines the resulting session key, is straightforward to prove for Signal: the key is a deterministic function of values included in the sid.

3.3.1 Multi-Stage models, formally

We revisit the multi-stage models of Fischlin and Günther [82], giving now a formal definition.

Definition 2 (Multi-stage key exchange protocol)**.** A *multi-stage key exchange protocol* Π is a tuple of algorithms, along with a keyspace $\mathcal{K}$ and a security parameter λ. The algorithms are:

- $\mathrm{KeyGen}() \overset{\$}{\mapsto} (\mathit{ipk}, \mathit{ik})$: A Probabilistic Polynomial-time (ppt) *long-term key generation algorithm* that outputs a long-term public key / secret key pair $(\mathit{ipk}, \mathit{ik})$. In Signal, these are called "identity keys".
- $\mathrm{MedTermKeyGen}(\mathit{ik}) \overset{\$}{\mapsto} (\mathit{prepk}, \mathit{prek})$: A ppt *medium-term key generation algorithm* that takes as input a long-term secret key ik and outputs a medium-term public key / secret key pair $(\mathit{prepk}, \mathit{prek})$. In Signal, these are called "signed prekeys"; in the key exchange literature, they are sometimes called "semi-static keys".
- $\mathrm{Activate}(\mathit{ik}, \mathit{prek}, \mathit{role}) \overset{\$}{\mapsto} (\pi', m')$: A ppt *protocol activation algorithm* that takes as input a long-term secret key ik, a medium-term secret key prek, and a role $\mathit{role} \in \{\mathtt{init}, \mathtt{resp}\}$, and outputs a state π' and (possibly empty) outgoing message m'.
- $\mathrm{Run}(\mathit{ik}, \mathit{prek}, \pi, m) \overset{\$}{\mapsto} (\pi', m')$: A ppt *protocol execution algorithm* that takes as input a long-term secret key ik, a medium-term secret key prek, a state π, and an incoming message m, and outputs an updated state π' and (possibly empty) outgoing message m'.

3.3.1.1 Key Indistinguishability Experiment

We now set up an experiment defining the security of a multi-stage key exchange protocol. As is typical, the experiment establishes long-term keys and then allows the adversary to interact with the protocol algorithms via queries issued to a challenger. These queries allow the adversary to

- start sessions with particular peers and medium-term keys,
- carry any message to and from any sessions (including modifying, dropping, delaying, and inserting messages), and
- reveal long-term or per-session secret information from any party.

At some point, the adversary may choose a single stage of a single session to "test". They are then given either the real session key from this stage, or a random key from the same keyspace. They may now continue executing, issuing more queries and perhaps revealing more secrets. Finally, they issue a guess as to whether they received the true session key of the Tested session. If they decide correctly, we say they win the experiment.

Definition 3 (Security experiment)**.** We define the *security experiment* $\mathsf{Exp}^{\text{ms-ind}} = \mathsf{Exp}^{\text{ms-ind}}_{\mathsf{fresh},\Pi,\mathsf{n_P},\mathsf{n_M},\mathsf{n_S},\mathsf{n_s}}$ in Figure 3.1. $\mathsf{Exp}^{\text{ms-ind}}$ is parameterised by a freshness predicate fresh which assigns Boolean values to tuples (u, i, s) representing sessions and stages, as well as by a protocol Π and integer bounds $\mathsf{n_P}, \mathsf{n_M}, \mathsf{n_S}, \mathsf{n_s}$ on the number of parties, medium-term keys, sessions and stages it allows. It includes the following global variables:

- b: a challenge bit
- $\mathsf{tested} = (u, i, s)$ or $\perp$: recording the inputs to the query $\mathsf{Test}(u, i, s)$, or $\perp$ if no Test query has happened

and extends the per-session state π_u^i with the following experiment variables. (Recall that experiment-only variables are in sans-serif.)

- $\pi_u^i.\mathsf{rand}[s] \in \{0,1\}^\lambda$: random coins for π_u^i's stage s
- $\pi_u^i.\mathsf{peerid} \in \{1, \ldots, \mathsf{n_P}\}$: the identifier of the alleged peer, set by the experiment when $\pi_u^i.\mathit{peeripk}$ is assigned
- $\pi_u^i.\mathsf{peerpreid} \in \{1, \ldots, \mathsf{n_M}\}$: the index of the alleged peer's medium-term key, set by the experiment when $\pi_u^i.\mathit{peerprepk}$ is assigned
- $\pi_u^i.\mathsf{rev\_session}[s] \in \{\text{true}, \text{false}\}$: whether $\mathsf{RevSessKey}(u, i, s)$ was called or not; default false
- $\pi_u^i.\mathsf{rev\_random}[s] \in \{\text{true}, \text{false}\}$: whether $\mathsf{RevRand}(u, i, s)$ was called or not; default false
- $\pi_u^i.\mathsf{rev\_state}[s] \in \{\text{true}, \text{false}\}$: whether $\mathsf{RevState}(u, i, s)$ was called or not; default false

Given a protocol Π and freshness predicate fresh, we are finally able to define security: Π is secure if no ppt algorithm can win the security game with non-negligible probability.

Definition 4 (Multi-stage key indistinguishability)**.** Let Π be a key exchange protocol and fresh a freshness predicate. Let $\mathsf{n_P}, \mathsf{n_M}, \mathsf{n_S}, \mathsf{n_s} \in \mathbb{N}$ denote upper bounds on the number of parties, medium-term keys, sessions per party and stages per session. Let $\mathcal{A}$ be an efficient probabilistic algorithm (i.e., one whose running time is a polynomial function of the security parameter). Then the *advantage* of $\mathcal{A}$ is defined by

$$\mathsf{Adv}^{\text{ms-ind}}_{\Pi,\mathsf{n_P},\mathsf{n_M},\mathsf{n_S},\mathsf{n_s}}(\mathcal{A}) = \left|2 \Pr\left[\mathsf{Exp}^{\text{ms-ind}}_{\mathsf{fresh},\Pi,\mathsf{n_P},\mathsf{n_M},\mathsf{n_S},\mathsf{n_s}}(\mathcal{A}) = 1\right] - 1\right|$$

We say that Π is secure if $\mathsf{Adv}^{\text{ms-ind}}_{\Pi,\mathsf{n_P},\mathsf{n_M},\mathsf{n_S},\mathsf{n_s}}(\mathcal{A})$ is a negligible function of the security parameter.

```
$\mathsf{Send}(u, i, m)$:
if $\pi_u^i = \bot$ then
    // start new session and record intended peer
    parse $m$ as $(\pi_u^i.\mathsf{preid}, \mathit{role})$
    $\pi_u^i.\vec{\mathsf{rand}} \xleftarrow{\$} \{0,1\}^{\mathsf{n_s} \times \lambda}$
    $(\pi_u^i, m') \leftarrow \mathrm{Activate}(\mathit{ik}_u, \mathit{prek}_u^{\pi_u^i.\mathsf{preid}}, \mathit{role}; \pi_u^i.\mathsf{rand}[0])$
    return $m'$
else
    $s \leftarrow \pi_u^i.\mathit{stage}$
    $(\pi_u^i, m') \leftarrow \mathrm{Run}(\mathit{ik}_u, \mathit{prek}_u^{\pi_u^i.\mathsf{preid}}, \pi_u^i, m; \pi_u^i.\mathsf{rand}[s])$
    return $m'$
```

```
$\mathsf{Test}(u, i, s)$:
// can only call Test once, and only on accepted stages
if $(\mathsf{tested} \neq \bot)$ or $(\pi_u^i.\mathit{status}[s] \neq \mathtt{accept})$ then
    return $\bot$
$\mathsf{tested} \leftarrow (u, i, s)$
// return real or random key depending on b
if $\mathsf{b} = 0$ then
    return $\pi_u^i.k[s]$
else
    $k' \xleftarrow{\$} \mathcal{K}$
    return $k'$
```

```
$\mathsf{RevSessKey}(u, i, s)$:
$\pi_u^i.\mathsf{rev\_session}[s] \leftarrow \mathrm{true}$
return $\pi_u^i.k[s]$
```

```
$\mathsf{RevLongTermKey}(u)$:
$\mathsf{rev\_ltk}_u \leftarrow \mathrm{true}$
return $\mathit{ik}_u$
```

```
$\mathsf{RevMedTermKey}(u, \mathsf{preid})$:
$\mathsf{rev\_mtk}_u^{\mathsf{preid}} \leftarrow \mathrm{true}$
return $\mathit{prek}_u^{\mathsf{preid}}$
```

```
$\mathsf{RevRand}(u, i, s)$:
$\pi_u^i.\mathsf{rev\_random}[s] \leftarrow \mathrm{true}$
return $\pi_u^i.\mathsf{rand}[s]$
```

```
$\mathsf{RevState}(u, i, s)$:
$\pi_u^i.\mathsf{rev\_state}[s] \leftarrow \mathrm{true}$
return $\pi_u^i.\mathit{st}[s]$
```

(a) Challenger responses to oracle queries made in the multi-stage security game.

```
$\mathsf{Exp}^{\mathsf{ms\text{-}ind}}_{\mathsf{fresh},\Pi,\mathsf{n_P},\mathsf{n_M},\mathsf{n_S},\mathsf{n_s}}(\mathcal{A})$:
$\mathsf{b} \xleftarrow{\$} \{0,1\}$
$\mathsf{tested} \leftarrow \bot$
// generate long-term and semi-static keys
for $u = 1 \ldots \mathsf{n_P}$ do
    $(\mathit{ipk}_u, \mathit{ik}_u) \xleftarrow{\$} \mathrm{KeyGen}()$
    for $\mathsf{preid} = 1 \ldots \mathsf{n_M}$ do
        $(\mathit{prepk}_u^{\mathsf{preid}}, \mathit{prek}_u^{\mathsf{preid}}) \xleftarrow{\$} \mathrm{MedTermKeyGen}(\mathit{ik}_u)$
$\mathit{pubinfo} \leftarrow (\mathit{ipk}_1, \ldots, \mathit{ipk}_{\mathsf{n_P}}, \mathit{prepk}_1^1, \ldots, \mathit{prepk}_{\mathsf{n_P}}^{\mathsf{n_M}})$
$\mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\mathsf{Send},\mathsf{Rev}*,\mathsf{Test}}(\mathit{pubinfo})$
if $(\mathsf{tested} \neq \bot) \wedge \mathsf{fresh}(\mathsf{tested}) \wedge \mathsf{b} = \mathsf{b}'$ then
    return $1$ // the adversary wins
else
    return $0$ // the adversary loses
```

(b) Security game for an adversary $\mathcal{A}$ given oracle access to the queries in Figure 3.1a.

Figure 3.1: Security experiment for adversary $\mathcal{A}$ against the multi-stage key indistinguishability security of protocol Π. Figure 3.1a defines how the challenger responds to oracle queries by the adversary $\mathcal{A}$, and Figure 3.1b defines the security game itself. This figure is based on [59, Figure 6].

3.4 Signal as a MSKE protocol

We now have a framework in which we can define multi-stage protocols like Signal and give a formal statement of their security properties. The next steps are

- to instantiate Signal as a multi-stage AKE protocol (§3.4.2)
- to give a freshness predicate which captures the precise security guarantees which we believe it was intended to provide (§4.1)

Finally, we state our security theorem at the end of this chapter.

3.4.1 Changes: what we don't model

In addition to the general assumptions we described in §2.2.2, we highlight some features of Signal which we explicitly do not model in our analysis, and differences between our model and the protocol as implemented.

Reordering We reorder various messages to make the boundaries between stages clear. Signal handshake data is never sent on its own; instead, it is either pre-registered at a server or attached to a user's encrypted message. This means that various operations, such as processing a received ratchet public key and generating a new one, are collapsed into a single event in implementations. We separate these operations out into their own stages, marking the changes in Dark red in Figure 3.2.

We consider the initial registration and fetch of one-time prekeys $eprepk_B$ as the first message of the protocol, instead of relaying it through a server. This means that the first message of the protocol is in the direction $B \rightarrow A$ and contains registration information for B.

We consider the first message keys for each chain to be generated when the chain is initialised, instead of when a new message is received under that chain. This allows us to define those message keys as the session keys of their asymmetric stages. Similarly, we consider ratchet public keys to be sent out by asymmetric stages, instead of attached to subsequent messages.

Signature key reuse Signal uses the same key for DH agreement and for signing the medium-term prekeys. (This is done in practise by reinterpreting Curve25519 points as Ed25519 keys, and signing with EdDSA.) This form of key reuse breaks key separation and is challenging to analyse, because one needs to show that the signature does not leak information about the DH operations and vice versa. Moreover, in threat models where the adversary can compromise certain keys, the security property must now take into account cases where the adversary is able to forge these signatures.

In our computational model, this key reuse poses a particular challenge: when replacing with random a reused DH key, the simulator must now be able to produce valid-looking signatures as well. Degabriele et al. [68] and Paterson et al. [141] use the hashing random oracle under Gap Diffie–Hellman (GDH) to define such a simulator under the GDH assumption, but this would add a fair amount of complexity to our model. We instead consider the key reuse as out of scope, enforcing authentication of the prekeys in our security model. In the game, this is implemented as an extra argument when the adversary creates a new session, dictating which of the honestly-generated medium-term keys is to be used for that session. This enforced authentication means we do not capture the class of attacks in which the adversary corrupts an identity key and then inserts a malicious signed pre-key.

One generalisation of our model would be to allow an adversary the ability to register malicious prekeys instead of just corrupting honest ones. This would capture for example attacks in which certain asymmetric keys are weak but these weak keys are unlikely to be generated by the honest key generation algorithm. (This generalisation is similar to that of ASICS [39] but for prekeys instead of identity keys.)

Authentication of ratchet keys Signal uses the associated data of message encryptions to authenticate, among other things, public ratchet keys. This means that, as discussed in §2.2.4.1 on page 31, it cannot achieve a standard definition of key indistinguishability: an adversary can always tell if a Signal message key is real or random, by using it to decrypt a message.

We phrase our security property so as not to require the authentication of ratchet keys: if an adversary injects a compromised ratchet key into a previously-secure session then PCS requires that subsequent message keys are secret, and otherwise we consider the attack invalid. This means that even though Signal might resist attacks in which an adversary attempts to inject malicious ratchet keys, we did not prove that it does so.

Out-of-order messages Signal supports decrypting messages which arrive out of order. To do so, if a message arrives at index j in a receive chain but the recipient has only advanced to index $i < j$, then they advance the chain $j - i$ times. Each of these steps produces a new message key, and the recipient stores these messages keys in their local memory until the corresponding message arrives.

This feature weakens forward secrecy, since corrupting an agent who has stored one of these keys now allows decrypting messages sent under the stored key. We do not model this feature, in the sense that we do not include previous message keys in the state passed between stages.

Simultaneous initiations Signal has a mechanism to deal silently with the case that Alex and Blake simultaneously initiate a session with each other. Roughly, when an agent detects that this has happened they deterministically choose one party as the initiator (e.g. by sorting identity public keys and choosing the smaller), and then complete the session as if the other party had not acted. This requires a certain amount of trial and error: agents maintain multiple states for each peer, and attempt decryption of incoming messages in all of them. We do not consider this mechanism.

3.4.2 Algorithms for Signal

Recall that to define a multi-stage key exchange protocol we must define the algorithms KeyGen, MedTermKeyGen, Activate and Run. We do so now for Signal. KeyGen and MedTermKeyGen are uniform sampling from the DH group. Activate depends on the invoked role. Our prekey reorganization described above means that the roles of initiator and responder are technically reversed: although intuitively Alex initiates a session in our presentation, in fact Blake sends the first message, namely the set of prekeys. This is the first right-to-left flow of Figure 3.2(b). Thus, the activation algorithm for the responder (Blake) outputs a single one-time prekey and awaits a response. The activation algorithm for the initiator (Alex) outputs nothing and awaits incoming prekeys. Both algorithms also perform the administrative overhead of the game, creating oracles and initialising session states.

Run is the core protocol algorithm. It admits various cases, which we briefly describe. If the incoming message is the first, Run builds a session as described previously: for Alex, it operates as in the left side of Figure 3.2(b) and outputs a message containing epk_A; for Blake, it operates as in the right side of Figure 3.2(b) and outputs nothing.

After that, there are two cases: Run is either invoked to process an incoming message, or to encrypt an outgoing one. We distinguish between incoming ratchet public keys (causing asymmetric updates) and incoming messages (causing symmetric updates).

(i) *Outgoing message.* Perform a symmetric sending update, modifying the current sending chain key and using the resulting message key as the session key (left side of Figure 3.2(c)).

(ii) *Incoming ratchet public key.* If this ratchet public key has not been processed before, perform an asymmetric update using it to derive new sending and receiving chain keys as in Figure 3.2(d). Advance both chains by one step, and output the message keys as the session key for the two asymmetric sub-stages as indicated in the figure.

(iii) *Incoming message.* Use the message metadata to determine which receiving chain should be used for decryption, and which position the message takes in the chain. Advance that chain according to the right side of Figure 3.2(c) as many stages as necessary (possibly zero), storing for future use any message keys that were thus generated. Return as the session key the next receiving message key.

We now describe the algorithms in Figure 3.2 in more detail. Recall that there are three stage types in Signal: initial ([0]), symmetric ratcheting ([**sym-ir**:x,y] or [**sym-ri**:x,y]), and asymmetric ratcheting ([**asym-ri**:x] or [**asym-ir**:x]). Ratcheting stages differ based on whether they are used for generating keys for the initiator to send to the responder (denoted **-ir**) or vice versa (denoted **-ri**). For our purposes, every stage generates a session key; depending on the stage, this will be either the sending or the receiving message key.

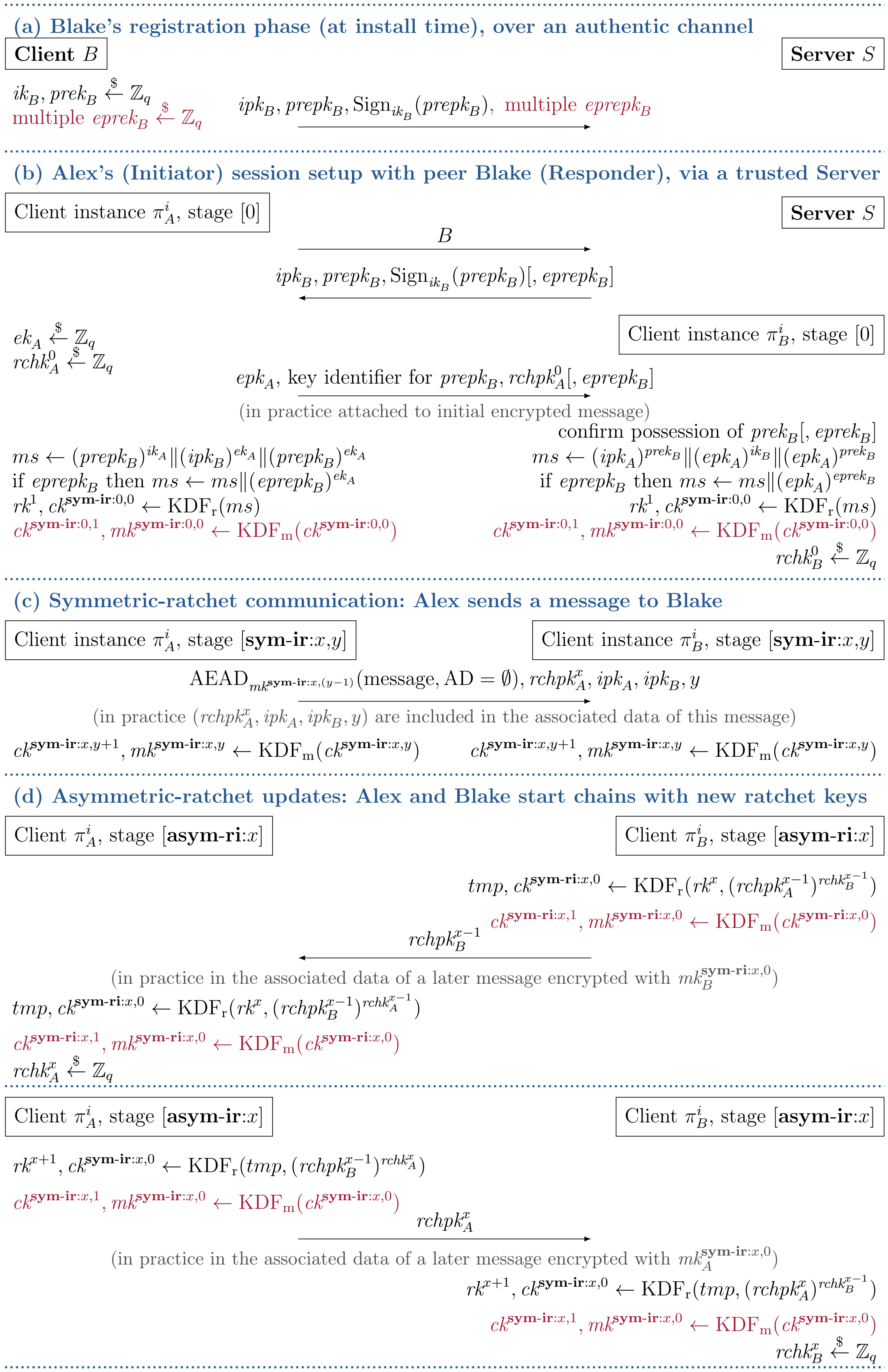

(a) Blake's registration phase (at install time), over an authentic channel
Client B
Server S
$ik_B, prek_B \xleftarrow{\$} \mathbb{Z}_q$
multiple $eprek_B \xleftarrow{\$} \mathbb{Z}_q$
$ipk_B, prepk_B, \mathrm{Sign}_{ik_B}(prepk_B)$, multiple $eprepk_B$
(b) Alex's (Initiator) session setup with peer Blake (Responder), via a trusted Server
Client instance π_A^i, stage [0]
Server S
B
$ipk_B, prepk_B, \mathrm{Sign}_{ik_B}(prepk_B)[, eprepk_B]$
Client instance π_B^i, stage [0]
$ek_A \xleftarrow{\$} \mathbb{Z}_q$
$rchk_A^0 \xleftarrow{\$} \mathbb{Z}_q$
epk_A, key identifier for $prepk_B, rchpk_A^0[, eprepk_B]$
(in practice attached to initial encrypted message)
confirm possession of $prek_B[, eprek_B]$
$ms \leftarrow (prepk_B)^{ik_A} \| (ipk_B)^{ek_A} \| (prepk_B)^{ek_A}$
if $eprepk_B$ then $ms \leftarrow ms \| (eprepk_B)^{ek_A}$
$rk^1, ck^{\textbf{sym-ir}:0,0} \leftarrow \mathrm{KDF_r}(ms)$
$ck^{\textbf{sym-ir}:0,1}, mk^{\textbf{sym-ir}:0,0} \leftarrow \mathrm{KDF_m}(ck^{\textbf{sym-ir}:0,0})$
$ms \leftarrow (ipk_A)^{prek_B} \| (epk_A)^{ik_B} \| (epk_A)^{prek_B}$
if $eprepk_B$ then $ms \leftarrow ms \| (epk_A)^{eprek_B}$
$rk^1, ck^{\textbf{sym-ir}:0,0} \leftarrow \mathrm{KDF_r}(ms)$
$ck^{\textbf{sym-ir}:0,1}, mk^{\textbf{sym-ir}:0,0} \leftarrow \mathrm{KDF_m}(ck^{\textbf{sym-ir}:0,0})$
$rchk_B^0 \xleftarrow{\$} \mathbb{Z}_q$
(c) Symmetric-ratchet communication: Alex sends a message to Blake
Client instance π_A^i, stage [**sym-ir**:x,y]
Client instance π_B^i, stage [**sym-ir**:x,y]
$\mathrm{AEAD}_{mk^{\textbf{sym-ir}:x,(y-1)}}(\text{message}, \text{AD} = \emptyset), rchpk_A^x, ipk_A, ipk_B, y$
(in practice $(rchpk_A^x, ipk_A, ipk_B, y)$ are included in the associated data of this message)
$ck^{\textbf{sym-ir}:x,y+1}, mk^{\textbf{sym-ir}:x,y} \leftarrow \mathrm{KDF_m}(ck^{\textbf{sym-ir}:x,y})$
$ck^{\textbf{sym-ir}:x,y+1}, mk^{\textbf{sym-ir}:x,y} \leftarrow \mathrm{KDF_m}(ck^{\textbf{sym-ir}:x,y})$
(d) Asymmetric-ratchet updates: Alex and Blake start chains with new ratchet keys
Client π_A^i, stage [**asym-ri**:x]
Client π_B^i, stage [**asym-ri**:x]
$tmp, ck^{\textbf{sym-ri}:x,0} \leftarrow \mathrm{KDF_r}(rk^x, (rchpk_A^{x-1})^{rchk_B^{x-1}})$
$ck^{\textbf{sym-ri}:x,1}, mk^{\textbf{sym-ri}:x,0} \leftarrow \mathrm{KDF_m}(ck^{\textbf{sym-ri}:x,0})$
$rchpk_B^{x-1}$
(in practice in the associated data of a later message encrypted with $mk_B^{\textbf{sym-ri}:x,0}$)
$tmp, ck^{\textbf{sym-ri}:x,0} \leftarrow \mathrm{KDF_r}(rk^x, (rchpk_B^{x-1})^{rchk_A^{x-1}})$
$ck^{\textbf{sym-ri}:x,1}, mk^{\textbf{sym-ri}:x,0} \leftarrow \mathrm{KDF_m}(ck^{\textbf{sym-ri}:x,0})$
$rchk_A^x \xleftarrow{\$} \mathbb{Z}_q$
Client π_A^i, stage [**asym-ir**:x]
Client π_B^i, stage [**asym-ir**:x]
$rk^{x+1}, ck^{\textbf{sym-ir}:x,0} \leftarrow \mathrm{KDF_r}(tmp, (rchpk_B^{x-1})^{rchk_A^x})$
$ck^{\textbf{sym-ir}:x,1}, mk^{\textbf{sym-ir}:x,0} \leftarrow \mathrm{KDF_m}(ck^{\textbf{sym-ir}:x,0})$
$rchpk_A^x$
(in practice in the associated data of a later message encrypted with $mk_A^{\textbf{sym-ir}:x,0}$)
$rk^{x+1}, ck^{\textbf{sym-ir}:x,0} \leftarrow \mathrm{KDF_r}(tmp, (rchpk_A^x)^{rchk_B^{x-1}})$
$ck^{\textbf{sym-ir}:x,1}, mk^{\textbf{sym-ir}:x,0} \leftarrow \mathrm{KDF_m}(ck^{\textbf{sym-ir}:x,0})$
$rchk_B^x \xleftarrow{\$} \mathbb{Z}_q$

Figure 3.2: A single session of the Signal protocol (caption on the next page).

Figure 3.2: (on the preceding page) A single session of the Signal protocol, including preregistration of keys. Local actions are depicted in the left and right columns, and messages flow between them. We show only one step of the symmetric and asymmetric ratchets; they can be iterated arbitrarily. Variables storing keys are defined in Table 2.1, $\mathrm{KDF_r}$ and $\mathrm{KDF_m}$ in Figure 2.5, and session identifiers in Table 3.1. Dark red text indicates reordered actions in our model. Each stage derives message keys with the same index as the stage number, and chaining/root keys with the index for the next stage; the latter is passed as state from one stage to the next. State info *st* in asymmetric stages is defined as the root key used in the key derivation, and for symmetric stages *st* is defined as the chain key used in key derivation. Symmetric stages always start at $y = 1$ and increment. When an actor sends consecutive messages, the first message is a DH ratchet and then subsequent messages use the symmetric ratchet. When an actor replies, they always DH ratchet first; they never carry on the symmetric ratchet. This figure is from [59].

Session setup The key exchange occurs in the initial stage of a Signal session, which we call stage [0]. After retrieving data from the key distribution server, Alex (the initiator) computes a master secret

$$ms \leftarrow (prepk_B)^{ik_A} \| (ipk_B)^{ek_A} \| (prepk_B)^{ek_A} \left[\| (eprepk_B)^{ek_A} \right]$$

where the final concatenand $(eprepk_B)^{ek_A}$ is present only if A received $eprepk_B$ from the server. Alex then applies the root key derivation function $\mathrm{KDF_r}$ to ms with an empty DH value to derive two initial shared secrets: an initial root key rk^0 and an initial sending chain key $ck^{\textbf{sym-ir}:0,0}$.

Recall that Signal intermingles its key exchange and message transport: Alex's ephemeral public key is transmitted as associated data of the first encrypted message. Since our key exchange model requires a clean separation between key derivations and uses, we will say instead that Alex sends the ephemeral key as an extra handshake message, and as part of stage [0] also generates an initial sending message key $mk^{\textbf{sym-ir}:0,0}$ (which is this stage's session key output) and the next sending chain key $ck^{\textbf{sym-ri}:0,0}$ as part of this stage. Finally, Alex also generates a new ephemeral DH ratchet key $rchk^0$ and stores it in session memory.

Upon receiving Alex's initial message, Blake performs the responder analogue of this computation, deriving the same shared root key rk^0 and chain key $ck^{\textbf{sym-ir}:0,0}$. We also have Blake generate an initial receiving message key $mk^{\textbf{sym-ir}:0,0}$ (which is this stage's session key output) and the next receiving chain key $ck^{\textbf{sym-ir}:0,1}$.

Symmetric Ratchet To perform a symmetric ratchet stage [**sym-ir**:x,y], Alex takes the current sending chain key $ck^{\textbf{sym-ir}:x,y}$ and applies the message key derivation function $\mathrm{KDF_m}$ to derive two new keys: an updated sending chain key $ck^{\textbf{sym-ir}:x,(y+1)}$ and a sending message key $mk^{\textbf{sym-ir}:x,y}$. The sending chain key is stored in Alex's session state, and the message key becomes the stage key of the stage

Symmetric ratchet stages computed by the responder are analogous, but are called [**sym-ri**:x,y].

Asymmetric Ratchet To perform an asymmetric ratchet stage [**asym-ri**:x], Alex receives a new (previously unseen) ratchet public key $rchpk_B^{x-1}$ from Blake; note that for this to occur Alex must already have stored a previously generated private ratchet key $rchk_A^{x-1}$.

Alex computes a DH shared secret between $rchpk_B^{x-1}$ and $rchk_A^{x-1}$, and uses this secret as input to $\mathrm{KDF_r}$ together with the previous root chain key. The output of $\mathrm{KDF_r}$ is then used to compute new receiving chain and message keys, as well as an intermediate temporary value stored in local memory.

To perform an asymmetric ratchet stage **asym-ir**:x, Alex generates a new ratchet private key $rchk_A^x$ and computes a DH shared secret between $rchpk_B^{x-1}$ and $rchk_A^x$. As before, this shared secret is then used as input to $\mathrm{KDF_r}$ together with the intermediate temporary value from above. The output of $\mathrm{KDF_r}$ is used to compute new sending chain and message keys, as well as a replacement root chain key, and the ratchet public key sent out to the network.

Note that Signal implementations generally consider a *single* asymmetric update to comprise both of the above two stages. We separate them out because the message keys they derive have slightly different security properties. Intuitively, message keys derived in the second stage depend on message keys derived in the first stage, so they achieve a slightly stronger security guarantee.

Blake's computations are analogous. While symmetric updates can be triggered either by Alex (the session initiator) or Blake (the session responder) and thus could be as in Figure 3.2(c) or its horizontal flip, asymmetric updates can only be triggered by Alex (the session initiator) receiving a new (previously unseen) ratchet key from Blake (the session responder) and not the other way around, so Figure 3.2(d) will never be horizontally flipped.

3.5 Conclusion and Related Work

Let us examine what we have achieved so far. In §3.1 we gave a high-level threat model for Signal, and in Sections 3.2 and 3.3 a low-level model in the B-R framework which can express the security properties we would like to prove about Signal. We then looked at the protocol definition itself, and gave in §3.4 a detailed definition of the Signal algorithms in our notation. Instantiating Definition 4 in this framework will thus express our security property; to do so we must give a freshness predicate, which we shall do in Chapter 4.

3.5.1 Related Work

Symmetric ratcheting and DH updates (asymmetric ratcheting) are not the only way of updating state to ensure forward secrecy—i.e., that compromise of current state cannot be used to decrypt past communications. Forward-secure public key encryption [49] allows users to publish a short unchanging public key; messages are encrypted with knowledge of a time period, and after receiving a message, a user can update their secret key to prevent decryption of messages from earlier time periods.

Signal's asymmetric ratcheting, which it inherits from the design of OTR [37], have been claimed to offer properties such as "future secrecy". This property has been discussed in depth by Cohn-Gordon, Cremers and Garratt [57], whose key observation is that Signal's future secrecy is (informally) specified with respect to a passive adversary and therefore turns out to be implied by the formal notion of forward secrecy[1]. Instead, they observe that mechanisms such as asymmetric ratcheting can be used to achieve a substantially stronger property against an active adversary. They formally define this property as "post-compromise security", and show how this substantially raises the bar for resourceful network attackers to attack specific sessions.

Remark 5 ("Future Secrecy"). We see the term "future secrecy" as somewhat confusing and misleading, for two reasons. First, while in some sense it is dual to forward secrecy, the use of the word "future" implies that the two properties in fact deal with the same scenario. ("Backward secrecy" would not suffer from this problem, and indeed that term has also been used.) Second, because Open Whisper Systems's definition of future secrecy is against a passive adversary, it is often unclear exactly which threat model is intended. While "post-compromise security" is perhaps less elegant a term, we see it as much clearer: the security guarantee is provided *after* a compromise. For these reasons, we exclusively use the term PCS in this thesis.

Frosch et al. [83] performed a security analysis of TextSecure v3 (the protocol and app which later became Signal), showing that in their model the computation of the long-term symmetric key which seeds the ratchet comes from a secure one-round key exchange protocol, and that the key derivation function and authenticated encryption scheme used in TextSecure are secure. However, they did not cover any of the security properties of the ratcheting mechanisms.

Bellare et al. [27] develop generic security definitions for ratcheted key exchange in a different context, also based on a computational model with key indistinguishability. They view ratcheting as a cryptographic primitive in itself, and define a security game at a lower level than we do. They also give a DH based protocol that is somewhat similar to the Signal protocol in that it uses a ratcheting mechanism and updates state, prove that it is secure in their model under an oracle DH assumption, and show how to combine symmetric encryption schemes with ratcheted key exchange. Their model captures variations of "backward" (post-compromise) and forward secrecy, but only allows for one-way communication between Alex and Blake. Their security notions are therefore one-sided: if the receiver's long-term key is compromised then all security is lost. They also only capture the asymmetric type of ratcheting in this sense, and do not consider symmetric ratcheting. The authors explicitly identify modelling Signal as future work.

Poettering and Rösler [144] build on [27], extending the latter's work to bidirectional communications and allowing corruptions of either party's state. Again, they work at a lower level than our models, considering ratcheting as a standalone

[1] Roughly, if a *passive* adversary can learn some secret and then decrypt a later conversation, it could equally well wait for the conversation to occur and only then learn the secret. This latter property is precisely an attack on forward secrecy.

primitive and giving a generic construction for unidirectional ratcheting based on a Key Encapsulation Mechanism (KEM). They also give generic constrictions for bidirectional ratcheting, though somewhat surprisingly it turns out that they require stronger primitives, building a "key-updatable" KEM from Hierarchical Identity-Based Encryption (HIBE).

Kobeissi, Bhargavan and Blanchet [105] use ProVerif and CryptoVerif to analyze a simplified version (omitting e.g., symmetric ratcheting) of Signal specified in a JavaScript variant called ProScript. Their main focus is to present a methodology for automated verification for secure messaging protocols and implementations. They identify a possible KCI attack, which we rule out in our freshness predicate by requiring that the medium-term key not be compromised in the initial stage of a session. From the ProScript code, they automatically extract ProVerif models that consider a finite number of sessions without loops. The CryptoVerif models are created manually. In both cases, the analysis involves the systematic manual exploration of several combinations of compromised keys. In contrast, we set out to manually construct and prove the strongest possible property that holds for Signal. For the core protocol design, this allows us to prove a stronger and more-fine grained security property.

Green and Miers [90] use puncturable encryption to achieve fine-grained forward security with unchanging public keys. The intuition behind puncturable encryption is that it is possible to remove the ability to decrypt certain ciphertexts from a private key—to *puncture* it—without changing the corresponding public key. This means that after a handshake is conducted with a fixed public key, it is possible to puncture the corresponding private key at the relevant value so that it cannot be used to reproduce the computation of the secret key. This enables non-interactive forward secrecy for protocols such as TLS. While this is an interesting approach (especially for its relative conceptual simplicity), we focus on Signal due to its widespread adoption.

Lehmann and Tackmann [117] show PCS in a different context, giving a construction whereby a ciphertext can be transformed into a valid ciphertext under a new key such that the new ciphertext remains secret even if the previous key is compromised. This builds on the definition of PCS from [57].

CHAPTER 4 SIGNAL: ANALYSIS

This chapter is based on the analysis and reduction from

> *Katriel Cohn-Gordon, Cas J. F. 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 2017, Paris, France, April 26-28, 2017. *IEEE, 2017, pp. 451–466.* DOI*:* 10.1109/EuroSP.2017.27*.* URL*:* https://doi.org/10.1109/EuroSP.2017.27

We give a relatively high-level treatment of the analysis and reduction, which was jointly developed with Luke Garratt. We refer the reader to Luke's doctoral thesis or to the full version of [59] for a fuller treatment of the reduction.

In this section we prove that Signal is a secure multi-stage AKE protocol in the language of §3.3.1.1, under standard assumptions on the cryptographic building blocks. We have already defined a security experiment $\mathsf{Exp}^{\mathsf{ms\text{-}ind}}$ which captures a generic definition of security. It is parameterised by a freshness predicate, which depends on the protocol under consideration and expresses the strongest security guarantees that can be achieved by that protocol.

4.1 Freshness

We now give a freshness predicate for Signal, based on the attacks which we believe it was designed to resist. This allows us to capture the different security goals of each stage's derived keys, as we described when constructing our threat model.

Our goal when defining fresh is to describe the strongest security condition that might be provable for each of Signal's message keys based on the protocol's design; here, "strongest" is with respect to the maximal combinations of secrets learned by the adversary. That is, we use the structure of the protocol to infer which attacks cannot possibly be prevented by it, and rule them out by restricting the adversary. Our goal is to prove that, working from the design choices made, Signal indeed achieves the best it can. We remark that it is certainly possible to modify Signal in order to prevent further attacks, for example by introducing more elements into its key derivation function. Our goal, however, is to analyse the protocol with as few modifications as possible.

We must define fresh separately for the initial stages and for subsequent ones, since adversary queries can be targeted against any stage and additional secrets are introduced in asymmetric stages. In the initial stages, our choices are based on Figure 2.3 on page 13. In that graph, the edges can be seen as the individual secrets established between initiator and responder, on which the secrecy of the session keys is based. If the adversary cannot learn the secret corresponding to one of these edges, it cannot compute the session key. The adversary can learn the secret corresponding to an edge if it can compromise one of the two endpoints; thus, if an adversary can learn, e.g., the initiator A's ik_A and ek_A, it can derive the secrets corresponding to all edges. A similar observation can be made for the responder.

A vertex cover of a graph is a set of nodes incident to every edge. Each vertex cover of Figure 2.3 gives a way for the adversary to compute the relevant session key directly, since by construction it is exactly enough to compute the DH values attached to each edge. We can think of our freshness predicate as excluding all such vertex covers; if all DH pairs were included in the KDF, this would yield the standard eCK freshness predicate.

In stages after the initial ones, we define modified freshness conditions to capture Signal's PCS properties. These conditions are recursive: either the stage was fresh already, or it becomes fresh through the introduction of new secrets.

The freshness predicate fresh for our experiment works hand-in-hand with a variety of sub-predicates ($\mathsf{clean}_{\texttt{triple}}$, $\mathsf{clean}_{\texttt{triple+DHE}}$, $\mathsf{clean}_{\texttt{asym-ir}}$, $\mathsf{clean}_{\texttt{asym-ri}}$, $\mathsf{clean}_{\texttt{sym-ir}}$ and $\mathsf{clean}_{\texttt{sym-ri}}$) which are highly specialized to Signal to capture the exact type of security achieved in Signal's different types of stages.

Definition 5 (Validity and freshness)**.** Let s be the i^{th} session at agent u, and let $\tau = \pi_u^i.type[s]$ be its type (e.g. `triple`, `triple+DHE`, …). It is is valid if it has accepted and the adversary has not revealed either its session key or the session key of any session with the same identifier, and fresh if it additionally satisfies cleanness:

$$\begin{aligned} \mathsf{clean}_\tau(u,i,s) &\quad \text{is defined in the following sections} \\ \mathsf{valid}(u,i,s) &= (\pi_u^i.status[s] = \texttt{accept}) \wedge \neg \pi_u^i.\mathsf{rev\_session}[s] \\ &\quad \wedge \left(\forall\, j : \pi_u^i.sid[s] = \pi^j_{\pi_u^i.\mathsf{peerid}}.sid[s] \implies \neg \pi^j_{\pi_u^i.\mathsf{peerid}}.\mathsf{rev\_session}[s]\right) \\ \mathsf{fresh}(u,i,s) &= \mathsf{valid}(u,i,s) \wedge \mathsf{clean}_\tau(u,i,s) \end{aligned}$$

fresh and its sub-clauses have access to all variables in the experiment (global, user, session, and stage).

4.1.1 Session Setup Stage [0]

The session key derived from a `triple` (resp. `triple+DHE`) key exchange is derived from the concatenation of three (resp. four) DH shared secrets; thus, it will be secret as long as at least one of the input secrets is. The cleanness predicate in this stage is thus the disjunction of three (resp. four) predicates, each encoding the secrecy of one DH pair. Note that $\mathsf{clean}_{\mathtt{triple}}$ and $\mathsf{clean}_{\mathtt{triple+DHE}}$ only need to be defined for the initial key exchange, i.e., stage [0].

Definition 6 ($\mathsf{clean}_{\mathtt{triple}}$)**.** Within the same context as Definition 5, let

$$\begin{aligned}\mathsf{clean}_{\mathtt{triple}}(u,i,[0]) &= \mathsf{clean}_{\mathtt{LM}}(u,i) \vee \mathsf{clean}_{\mathtt{EL}}(u,i,0) \vee \mathsf{clean}_{\mathtt{EM}}(u,i,0)\\ \mathsf{clean}_{\mathtt{triple+DHE}}(u,i,[0]) &= \mathsf{clean}_{\mathtt{triple}}(u,i,[0]) \vee \mathsf{clean}_{\mathtt{EE}}(u,i,0,0)\end{aligned}$$

where the definitions of the sub-clauses $\mathsf{clean}_{\mathtt{XY}}$ will follow. Our naming convention is that initiator's key is of type `X` and the responder's key of type `Y`, where the possible types are `L`, `M`, and `E` for long-term (*ik*), medium-term (*prek*), and ephemeral (*ek*) keys respectively, as in Figure 2.3. (This is why we need the case distinctions below for when the tested session is the initiator or responder.)

The $\mathsf{clean}_{\mathtt{XY}}$ definitions are straightforward for initiator sessions. For responder sessions, the difficult part is that the ephemeral key is now the peer's, not the actor's: to ensure that it is not known by the adversary, we have to ensure first that it was actually generated by the intended peer (meaning that the peer session must exist), and second that it was not subsequently revealed (identifying the peer session using session identifiers). The following clause will help identify that precise situation.

$$\begin{aligned}\mathsf{clean}_{\mathtt{peerE}}(u,i,s) = \exists\, j : \pi_u^i.sid[s] = \pi^j_{\pi_u^i.\mathsf{peerid}}.sid[s]&\\ \wedge \Big(\forall\, j : \pi_u^i.sid[s] = \pi^j_{\pi_u^i.\mathsf{peerid}}.sid[s] \implies \neg\pi^j_{\pi_u^i.\mathsf{peerid}}.\mathsf{rev\_random}[s]\Big)&\end{aligned}$$

We can now define our various clean predicates, capturing nontrivial restrictions on the adversary. In particular, note that if the medium-term key is corrupted then we do not permit an attack impersonating Alex to Blake: since the only randomness in Signal's handshake is from the initiator (and there is no static-static DH secret), such an attack will succeed.

$$\mathsf{clean}_{\mathtt{LM}}(u,i) = \begin{cases}\neg\mathsf{rev\_ltk}_u \wedge \neg\mathsf{rev\_mtk}^{\pi_u^i.\mathsf{peerpreid}}_{\pi_u^i.\mathsf{peerid}} & \pi_u^i.role = \mathtt{init}\\ \neg\mathsf{rev\_ltk}_{\pi_u^i.\mathsf{peerid}} \wedge \neg\mathsf{rev\_mtk}^{\pi_u^i.\mathsf{preid}}_u & \pi_u^i.role = \mathtt{resp}\end{cases}$$

$$\mathsf{clean}_{\mathtt{EL}}(u,i,[0]) = \begin{cases}\neg\pi_u^i.\mathsf{rev\_random}[0] \wedge \neg\mathsf{rev\_ltk}_{\pi_u^i.\mathsf{peerid}} & \pi_u^i.role = \mathtt{init}\\ \mathsf{clean}_{\mathtt{peerE}}(u,i,[0]) \wedge \neg\mathsf{rev\_ltk}_u & \pi_u^i.role = \mathtt{resp}\end{cases}$$

$$\mathsf{clean}_{\mathtt{EM}}(u,i,[0]) = \begin{cases}\neg\pi_u^i.\mathsf{rev\_random}[0] \wedge \neg\mathsf{rev\_mtk}^{\pi_u^i.\mathsf{peerpreid}}_{\pi_u^i.\mathsf{peerid}} & \pi_u^i.role = \mathtt{init}\\ \mathsf{clean}_{\mathtt{peerE}}(u,i,[0]) \wedge \neg\mathsf{rev\_mtk}^{\pi_u^i.\mathsf{preid}}_u & \pi_u^i.role = \mathtt{resp}\end{cases}$$

$$\mathsf{clean}_{\mathtt{EE}}(u,i,s,s') = \begin{cases}\neg\pi_u^i.\mathsf{rev\_random}[s] \wedge \mathsf{clean}_{\mathtt{peerE}}(u,i,s') & \pi_u^i.role = \mathtt{init}\\ \mathsf{clean}_{\mathtt{peerE}}(u,i,s) \wedge \neg\pi_u^i.\mathsf{rev\_random}[s'] & \pi_u^i.role = \mathtt{resp}\end{cases}$$

Since we reveal randomness instead of specific keys, this final predicate applies to both the ephemeral keys and the ratchet keys, a fact which we shall use later when defining cleanness of asymmetric stages.

Excluded attacks. Recall that a vertex cover of Figure 2.3 on page 13 gives an attack which we rule out. Any cover must include one of $prek_B$ and ek_A to meet the edge between them, so the only (minimal) vertex covers for a `triple` handshake are the full state of B ($prek_B$, ik_B), the full state of A (ek_A, ik_A), or the pair ($prek_B$, ek_A). The former two are trivial: an agent must be able to compute its own session key, so learning all their secrets also allows the adversary to compute it. The final pair exists because of the lack of an edge in Signal between ik_A and ik_B, and means that an adversary who learns $prek_B$ and ek_A can learn the session key. In particular, since the ephemeral key is not authenticated, the adversary can generate their own ek_A and successfully impersonate A. This is the KCI attack of Kobeissi, Bhargavan and Blanchet [105]; it is ruled out in our model because $\mathsf{clean}_{\mathtt{triple}}$ is false if both $prek_B$ and ek_A have been revealed.

Since a vertex cover for a `triple`+`DHE` handshake must be a superset of the above, the only non-trivial one is again ($prek_B$, ek_A); this means that the KCI attack succeeds even against a `triple`+`DHE` handshake.

4.1.2 Asymmetric Stages [asym-ir:x]/[asym-ri:x]

In asymmetric stages, new ratchet keys are exchanged and used for DH to derive shared secrets, used in turn to start new symmetric chains. Roughly, we wish to encode that the keys derived from an asymmetric stage should be secret either if the stage is using unrevealed DH values, or if it follows a stage whose keys we expect to be secure.

The following predicate captures whether the state of a stage has been revealed directly via a query, targeting either it or a partner.

Definition 7 ($\mathsf{clean}_{\mathsf{state}}$). Within the same context as Definition 5, define

$$\begin{aligned}\mathsf{clean}_{\mathsf{state}}(u,i,s,s') = {} & \neg\pi_u^i.\mathsf{rev\_state}[s] \\ & \wedge \left(\forall\, j : \pi_u^i.sid[s] = \pi^j_{\pi_u^i.\mathsf{peerid}}.sid[s'] \implies \neg\pi^j_{\pi_u^i.\mathsf{peerid}}.\mathsf{rev\_state}[s']\right)\end{aligned}$$

We write $\mathsf{clean}_{\mathsf{state}}(u,i,s)$ as shorthand for $\mathsf{clean}_{\mathsf{state}}(u,i,s,s)$.

The state reveal query reveals additional state information that a previous stage gives as input to stage s. For Signal, we defined it as follows. For asymmetric stages, state reveal gives the root key used in the session key computation that was derived in the previous stage; for symmetric stages, state reveal gives the chain key derived in the previous stage.

During asymmetric ratcheting, there are actually two substages, in which keys with slightly different properties are derived. In the first substage, the parties apply a KDF to two pieces of keying material: the root key derived at the end of the previous asymmetric stage, and a DH shared secret derived from both party's previous ratcheting public keys. Keys from this substage are marked with $\mathit{sid}[\textbf{asym-ri}{:}x]$; they should be secure if either of the two pieces is unrevealed, which is what type `asym-ri` captures. In the second substage, the parties apply a KDF to three pieces of keying material: the root key, a DH shared secret from the first substage, and a DH shared secret derived from one party's previous ratcheting public key and the other's new ratcheting public key. Keys from this substage are marked with $\mathit{sid}[\textbf{asym-ir}{:}x]$ and should be secure if at least one of the three pieces is unrevealed, which is what `asym-ir` captures.

Definition 8 ($\mathsf{clean}_{\texttt{asym-ir}}, \mathsf{clean}_{\texttt{asym-ri}}$)**.** Within the same context as Definition 5, let $s_{ir} = [\textbf{asym-ir}{:}x]$, $s_{ri} = [\textbf{asym-ri}{:}x]$, $s'_{ir} = [\textbf{asym-ir}{:}x-1]$ and $s'_{ri} = [\textbf{asym-ri}{:}x-1]$ and define

$$\mathsf{clean}_{\texttt{asym-ri}}(u, i, s_{ri}) = \begin{cases} \mathsf{clean}_{\texttt{EE}}(u, i, [0], [0]) \ \vee \Big(\mathsf{clean}_{\mathsf{state}}(u, i, s_{ri}) \wedge \mathsf{clean}_{\pi_u^i.\mathit{type}[0]}(u, i, [0])\Big) & x = 1 \\ \mathsf{clean}_{\texttt{EE}}(u, i, s'_{ri}, s'_{ir}) \vee \Big(\mathsf{clean}_{\mathsf{state}}(u, i, s_{ri}) \wedge \mathsf{clean}_{\texttt{asym-ir}}(u, i, s'_{ir})\Big) & x > 1 \end{cases}$$

and

$$\mathsf{clean}_{\texttt{asym-ir}}(u, i, s_{ir}) = \begin{cases} \mathsf{clean}_{\texttt{EE}}(u, i, s_{ri}, [0]) \ \vee \Big(\mathsf{clean}_{\mathsf{state}}(u, i, s_{ir}) \wedge \mathsf{clean}_{\texttt{asym-ri}}(u, i, s_{ri})\Big) & x = 1 \\ \mathsf{clean}_{\texttt{EE}}(u, i, s_{ri}, s'_{ir}) \vee \Big(\mathsf{clean}_{\mathsf{state}}(u, i, s_{ir}) \wedge \mathsf{clean}_{\texttt{asym-ri}}(u, i, s_{ri})\Big) & x > 1 \end{cases}$$

These clauses capture the PCS goal of Signal: a stage is clean and thus should derive secret keys if its agent had been compromised at some prior time (i.e., their long-term key, past states and keys are compromised and thus the second disjuncts are not satisfied) but the current ephemeral keys of both parties are uncompromised and honest ($\mathsf{clean}_{\texttt{EE}}(u, i, s_{ir}, s_{ri})$ *is* satisfied). Dually, a stage is clean if its latest ratchet keys are compromised but the previous stage is clean. This latter case captures PCS.

Note that $\mathsf{clean}_{\texttt{EE}}$ is used twice (because cleanness of ephemerals is defined as cleanness of the random numbers): once to show that the randomness is clean when generating ephemerals for the initial key exchange, and once to show that it is clean when generating the first ratchet key pair.

4.1.3 Symmetric Stages [sym-ir:x,y] and [sym-ri:x,y]

For stages with only symmetric ratcheting, new session keys should be secure only if the state is unknown to the adversary: this demands that all previous states in this symmetric chain are uncompromised, since later keys in the chain are computable from earlier states in the chain. Thinking recursively, this means that the previous stage's key derivation should have been secure, and that the adversary has not revealed the state linking the previous stage with the current one.

While the symmetric sending and receiving chains derive independent keys and are triggered differently during Signal protocol execution, their security properties are identical and captured by the following predicate; the different forms of the predicate are due to needing to properly name the "preceding" stage. There are different freshness conditions depending on whether the symmetric stage is used for a message from initiator to responder or vice versa. Moreover, the symmetric stages arising from the initial handshake ($x = 0$) and from subsequent asymmetric stages ($x > 0$) are subtly different.

Definition 9 ($\mathsf{clean}_{\mathsf{sym}}$)**.** Writing $s = [\textbf{sym-ir}{:}x{,}y]$,

$$\begin{aligned}\mathsf{clean}_{\texttt{sym-ir}}(u,i,s) = & \mathsf{clean}_{\mathsf{state}}(u,i,s,s)\\ & \wedge \begin{cases}\mathsf{clean}_{\pi_u^i.type[0]}(u,i,[0]) & x = 0, y = 1\\ \mathsf{clean}_{\texttt{asym-ir}}(u,i,[\textbf{asym-ir}{:}x]) & x > 0, y = 1\\ \mathsf{clean}_{\texttt{sym-ir}}(u,i,[\textbf{sym-ir}{:}x{,}y-1]) & x \geq 0, y > 1\end{cases}\end{aligned}$$

There is no stage of type `sym-ri` with $x = 0$, so (writing now $s = [\textbf{sym-ri}{:}x{,}y]$)

$$\begin{aligned}\mathsf{clean}_{\texttt{sym-ri}}(u,i,s) = & \mathsf{clean}_{\mathsf{state}}(u,i,s,s)\\ & \wedge \begin{cases}\mathsf{clean}_{\texttt{asym-ri}}(u,i,[\textbf{asym-ri}{:}x]) & x > 0, y = 1\\ \mathsf{clean}_{\texttt{sym-ri}}(u,i,[\textbf{sym-ri}{:}x{,}y-1]) & x > 0, y > 1\end{cases}\end{aligned}$$

We write $\mathsf{clean}_{\mathsf{sym}}$ to denote $\mathsf{clean}_{\texttt{sym-ir}}$ or $\mathsf{clean}_{\texttt{sym-ri}}$ where it is clear which we mean.

Excluded attacks. Since no additional secrets are included in message keys derived from symmetric ratchet stages, these predicates simply require that the adversary has not compromised any previous state along the chain: neither the asymmetric stage which created the chain, nor any of the intermediate symmetric stages, are permitted targets for queries. In other words, we exclude the attack in which the adversary corrupts a chain key and computes subsequent message keys from it.

4.2 Security Theorem

At the conclusion of the previous chapter we stated

(i) a formal security model which can capture multi-stage AKE protocols, and

(ii) a freshness predicate for the Signal as instantiated in our model.

We can put these two parts together into a definition of security for Signal, by considering a ppt adversary attempting to distinguish any of its message keys from random. Specifically, recall that a multi-stage AKE protocol is secure just if for all ppt adversaries $\mathcal{A}$ which do not violate the freshness predicate, the probablity that $\mathcal{A}$ can distinguish a stage key of its choice from a random value is negligibly better than chance.

We prove this form of security for Signal under the PRF-ODH assumption and assuming that all key derivations are random oracles. This assumption is encoded by expressing a probability bound on adversary success in terms of the success probability of any adversary against the PRF-ODH game; if the latter is negligible, the former will be as well.

Thus, our main theorem:

Theorem 1. *The Signal protocol is a secure multi-stage key exchange protocol under the PRF-ODH assumption and assuming all KDFs are random oracles. That is, if no adversary achieves non-negligible advantage against the (mm)-PRF-ODH game, then the Signal advantage is a negligible function of the security parameter:*

$$\mathsf{Adv}^{\mathsf{ms\text{-}ind}}_{\Pi,\mathsf{n_P},\mathsf{n_M},\mathsf{n_S},\mathsf{n_s}}(\mathcal{A}) = \mathrm{negl}(n)$$

Remark 6 (On Monolithic Theorems). Most computational reductions are "mega-theorems" asserting that some protocol is secure, and ours is no exception. This is because of the security model, which encodes many different properties and attacks into a single indistinguishability experiment.

Some recent work in computational protocol models has begun to factor these theorems into smaller pieces. For example, Brzuska et al. [43] use two games, one for key secrecy and one to show that their partnering function is not too permissive. However, our models remain monolithic for now.

Proof (sketch). We give here a proof sketch. The full proof considers of each stage type and sub-clause exhaustively, and is structured using the sequence-of-games technique; it can be found at [58].

As with most game-hopping protocol proofs, we begin with a number of administrative game hops:

(i) to prevent collisions of DH keys
In the target of this hop, we have the challenger maintain a list of all honestly-generated DH values, and abort if a duplicate value ever appears in this list. Since our DH key generation is random sampling from a group of size q, we bound the advantage difference over this hop by a constant multiple of $^1/_q$.

(ii) to guess the Tested session
In the target of this hop, we have the challenger guess in advance the agent u and index i of the session against which the adversary chooses to issue its Test query, and abort if its guess is wrong. This guess is independent of the adversary's random coins, and thus is correct with probability 1 in the number of possible agents and indices. This is the main source of looseness in our proof.

(iii) to guess the peer of the Tested session
In the target of this hop, we have the challenge guess in advance an agent v, and abort if v is not the target of the Tested session. Again, this guess is correct with probability 1 in the number of agents. Note that we do not guess

the session index j at agent v, since it is not necessarily the case that a unique partner session exists.

Next, we break into cases depending on the types of the Tested stage. We give a probability bound for each case in turn; the maximum of each of these bounds is an overall bound for the success of an adversary against any stage.

Initial Stage In this case, we consider an adversary which issues a $\mathsf{Test}(u, i, [0])$ query. The initial stage can be either of type `triple` or `triple`+`DHE`.

Suppose first it is of type `triple`. We know from our security definition that the adversary can only win if the Tested stage is clean, so we know that $\mathsf{clean}_{\mathtt{triple}}(u, i, [0])$ holds. Looking back at the definition of $\mathsf{clean}_{\mathtt{triple}}$, we see that in turn one of $\mathsf{clean}_{\mathtt{LM}}(u, i)$, $\mathsf{clean}_{\mathtt{EL}}(u, i, 0)$ and $\mathsf{clean}_{\mathtt{EM}}(u, i, 0)$ must hold.

Suppose first that $\mathsf{clean}_{\mathtt{LM}}(u, i)$ holds. That is, we know that the long-term secret key of the initiator is unrevealed, as is the medium-term secret key of the responder, and we want to show that the first message key (which is a deterministic function of the first chain key) is secret. To do so, we'll replace the first chain key with a random value, and bound the probability difference over this hop using the PRF-ODH assumption.

Specifically, we assumed that all KDFs are instantiated by random oracles, so in particular the first chain key is the output of a call to the random oracle. The adversary cannot compute the random oracle's internal function itself, so it must either learn this output directly from a query (which will be ruled out by the cleanness predicate) or by querying the random oracle with the same input. We will therefore embed a PRF-ODH challenge into the input of the random oracle, so that in order to query the random oracle with the correct input the adversary must solve the PRF-ODH challenge.

Since we are in case `LM`, we do this by replacing the long-term key of agent u with the first PRF-ODH challenge value, and the correct medium-term key of agent v with the other. In order to do this, we first guess which medium-term key was used, which requires two more game hops. After doing so, we arrive at a game in which the adversary must solve the PRF-ODH challenge in one of its queries to the random oracle. We must show that the simulation of the game with the replaced values is indistinguishable from the original game, which requires a fair amount of work which we do not describe here. In particular, the simulator uses the PRF oracle in order to simulate certain adversary behaviours. We depict in Figure 4.1 the Signal key schedule, and mark

(i) the keys which are assumed honest,
(ii) the keys which may be corrupted or malicious, and
(iii) the keys which are replaced with random values.

Now, we can show that the challenger can simulate the adversary actions by using the PRF oracle to which the PRF-ODH game grants access. We can therefore bound the probability difference over this game hop by the success probability of any adversary against the PRF-ODH game. The conclusion is a probability bound in terms of the success against a game in which the first chaining key has been replaced

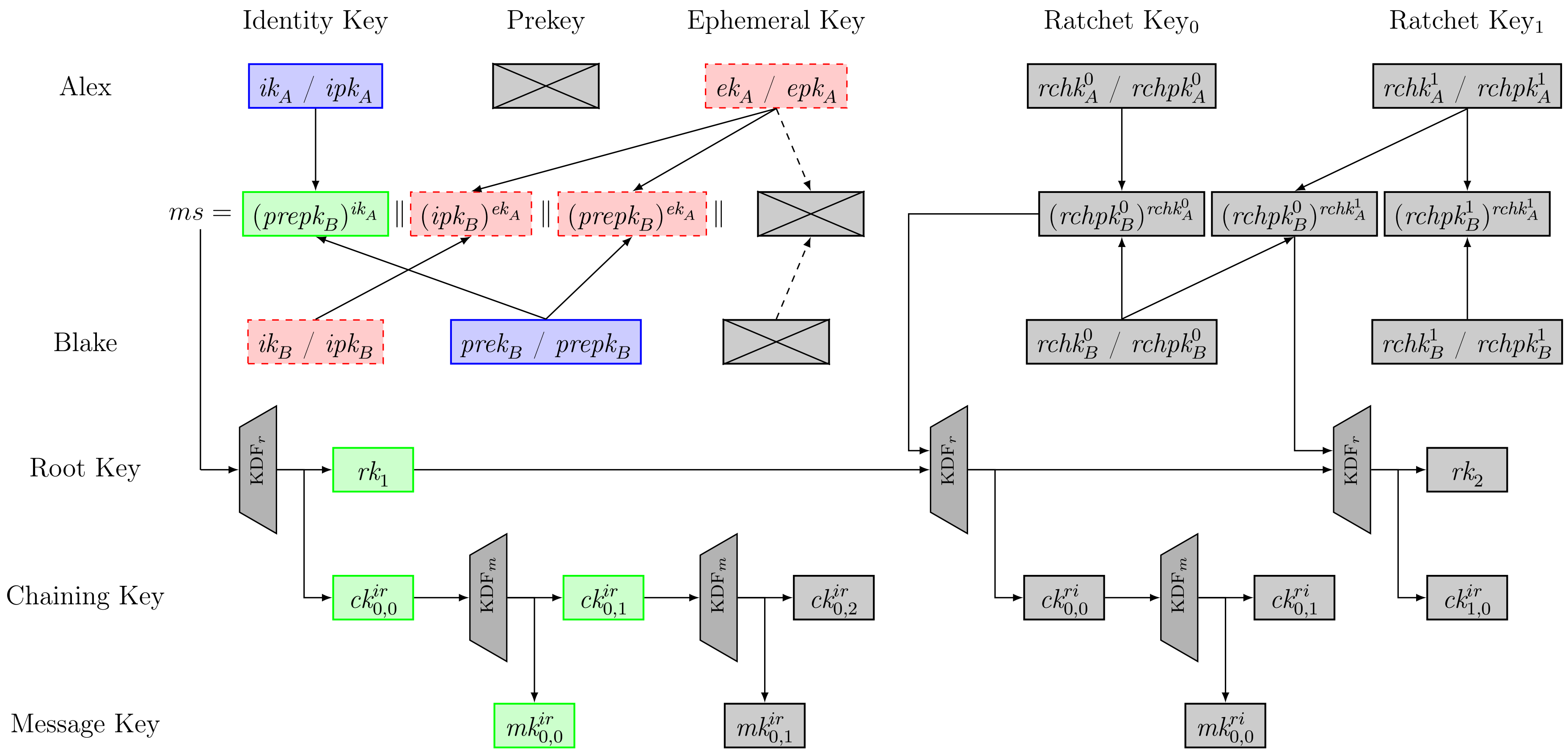


Figure 4.1: Depiction of the corrupted and honest keys in the first replacement game hop in our sketch proof. In particular, 🟥 denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); 🟩 denotes secrets that are assumed honest; 🟦 denotes secrets that the challenger is able to replace with random, thus ensuring security; and ⬜ denotes secrets that are not relevant to this case. This figure is from [58].

by a random value, which no adversary can win with better than even odds.

Almost all the remaining cases in the proof follow the same lines. However, in each case different keys are replaced with the PRF-ODH challenge values, and therefore the simulation algorithm is different. If $\mathsf{clean}_{\mathtt{EL}}(u, i, [0])$ holds then the challenge values are inserted in place of the initiator's ephemeral key and the responder's long-term key, and if $\mathsf{clean}_{\mathtt{EM}}(u, i, [0])$ holds then they are inserted in place of the initiator's ephemeral key and the responder's medium-term key. If the initial stage is of type triple+DHE then the cases are the same but there is an additional possibility ($\mathsf{clean}_{\mathtt{EE}}(u, i, [0])$), in which case the PRF-ODH challenge values are inserted in place of the initiator's and responder's ephemeral keys.

In each case, we proceed via a sequence of game hops to an unwinnable game, accumulating a negligible probability bound along the way.

Asymmetric stages. We describe the reductions here at a much higher level. In this case, we consider an adversary which issues a Test query against a stage s of type asym-ir or asym-ri. (There is a subtlety for the first asymmetric stage which we skip over here.)

Again, we take cases over the different ways to satisfy the cleanness predicate, depending on the stage type. For example, for stages $[\textbf{asym-ir}{:}x]$, $\mathsf{clean}_{\mathtt{asym\text{-}ir}}$ requires that either $\mathsf{clean}_{\mathtt{asym\text{-}ri}}(u, i, [\textbf{asym-ri}{:}x-1]) \wedge \mathsf{clean}_{\mathsf{state}}(u, i, [\textbf{asym-ir}{:}x])$ or $\mathsf{clean}_{\mathtt{EE}}(u, i, x-1, x-1)$. This case is where we see PCS appear: for stage x to be clean, either its predecessor must be clean or it must use a pair of honest ephemeral keys.

In the latter case, we perform a PRF-ODH reduction similar to the one already described. The PRF-ODH challenge values are inserted in place of the two honest ephemeral keys, and the subsequent stage keys replaced with random values.

In the former case, we replace the root key of the *previous* stage with a random value. Detecting this change is precisely differentiating a real root key from random, and thus if an adversary can detect it then there exists an adversary which breaks the security of the previous stage. By induction, therefore, we conclude that the change is not detectable. Once the previous root key is replaced with random, the remaining game hops are fairly straightforward.

Symmetric stages. Finally, we consider the security of stages of type sym. Here there is no disjunction in the cleanness predicate and hence only one case to consider. The argument is similar to the above one for asymmetric ratcheting: we replace the previous chain key with a random value, inductively showing that detecting this change would violate the security of the previous stage.

Conclusion. The theorem follows by summing probabilities. □

4.3 Conclusion

We have now achieved our original goal: starting from a high-level threat model for messaging protocols in their modern context, we have

(i) defined a formal B-R model for general multi-stage key exchange protocols,
(ii) instantiated the Signal protocol in our model,
(iii) given a freshness predicate encoding the properties which—according to our threat model—we believe Signal was designed to provide, and
(iv) stated and proved hardness of the security experiment for Signal with our freshness predicate.

Intuitively, we can now say that we have proved the strong security properties of Signal in a formal computational model.

CHAPTER 5
ART: GROUP AND MULTI-DEVICE MESSAGING

This chapter is based on the paper

> *Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican and Kevin Milner.* On Ends-to-Ends Encryption: Asynchronous Group Messaging with Strong Security Guarantees. *Cryptology ePrint Archive, Report 2017/666. http://eprint.iacr.org/2017/666. 2017*

My core contributions were to the Asynchronous Ratcheting Trees (ART) design and the computational security model. Luke Garratt made significant contributions to the security proof, and Kevin Milner to the algorithm pseudocode.

The Signal messaging protocol is fundamentally a two-party protocol: it specifies how two agents can exchange DH public keys, derive shared secret state, and encrypt messages for each other. While this gives a powerful primitive on which to build, modern messaging applications almost always intend to support conversations with more than two endpoints, for two main reasons:

Groups Many conversations are with more than two users: a message might be intended for all attendees of an event, or all participants in a workshop, or just an ad-hoc list of people. This mirrors the functionality of email, where multiple recipients can be included on a single mail.

Multiple devices Even in the case of two-user conversations, users may wish to send and receive messages on a number of different mobile phones, tablets, desktops and even web browsers. Unless all of these devices share their secret keys—a design which provides a number of significant security challenges—the messaging protocol itself must allow all of these devices to communicate.

In order to support these use cases, most implementers of end-to-end-encrypted group messaging at significant scale have invented their own protocols in order to support group and multidevice messaging. While they provide some strong security guarantees, many of these custom n-party protocols have significant drawbacks, either in security or in efficiency.

In this chapter, we'll present a design for a messaging protocol which natively supports groups. Our design, ART, provides strong security properties analogous to those of the two-party Signal protocol, while allowing for groups of any size to derive a shared symmetric key. It scales logarithmically in the number of participants, and we suggest extensions for group additions and removals. It is also the basis of a new IETF working group called MLS, for which we are working closely with industry collaborators in order to standardise and deploy a secure group messaging protocol at scale.

Messaging Layer Security (MLS)

We designed ART to be practically usable at the scale of WhatsApp, and with the goal of industry relevance. Like Signal, it is intended to work for deployments of hundreds of millions of users and to be efficient enough for low-end commodity devices to perform its cryptographic computations.

As part of this process, we worked with industry in the design and analysis process as much as possible. To ensure that we agree on a common standard and that it is usable as widely as possible, we intend to turn it into an Internet standard through the IETF. We have already convened two interim meetings (including academics and representatives from Facebook, Google, Twitter, Cisco and Wire, among others). After a successful "Birds of a Feather" session in London, the IETF has officially chartered a working group called Messaging Layer Security (MLS) to specify a protocol for secure group messaging building on our ART design.

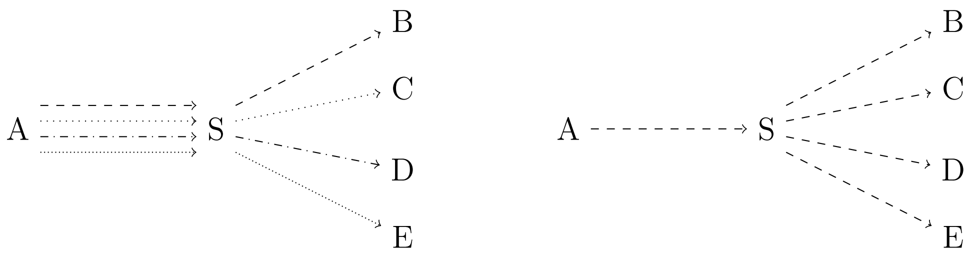
The standardisation process gathers input from any interested academic or industrial partners, and targets engineering challenges as well as the abstract protocol design and security analysis which we focus on here. For example, in our context it is enough to give an abstract specification of trees as mathematical objects, while in an IETF Request For Comments (RFC) we would define a precise canonical representation of the group state as a bit string. Since MLS aims to specify a deployable protocol it can also not make many of the abstractions which we did: it must specify primitives and primitive negotiation, wire formats, and so on.

5.1 Background

We briefly look at the existing ways Signal is used with n parties.

One natural way is to send messages separately to each recipient across their own Signal channel. In practice all messages are relayed by a transport server since peer-to-peer communication over the Internet is generally challenging, so this leads to the message flows depicted in Figure 5.1a. Sending a *single* message to n recipients using this scheme in fact requires A to send n copies of the message to the transport server S, with each copy encrypted under a different key.

This design is not inherently impossible to deploy, and indeed iMessage, Wire and the Signal app itself all use it for group messaging. (They also all implement multidevice messaging by considering each device as a distinct group member, with



(a) Sending pairwise messages from A to B, C, D and E via a server S.

(b) Sending a fanned-out message from A to B, C, D and E via a server S.

Figure 5.1: Comparing messages sent over pairwise channels with server-side fanout. Line type represents the message content. In all cases the server must send a single message to each recipient, but without server-side fanout the sender must send n different messages to the server as opposed to just a single one.

its own keys.) We remark that iMessage's deployment in particular indicates that the n-times per message overhead was considered practical on a 2009 iPhone 3GS.

However, there are significant limitations to using pairwise channels once group sizes start to increase. First, the linear scaling in computation becomes painful: while a 2x or 3x overhead can be relatively tolerable, it is hard to justify sending 1000 encryptions of the same message in order to implement a large group. Second, the extra bandwidth requirements can pose a challenge even for small groups: a short encrypted message of say 32 bytes becomes 1.6kB when sent to a group of size 50. This is particularly problematic for telephone networks in developing countries: the 2015 State of Connectivity report by internet.org [94] listed affordability of mobile data as one of the four major barriers to global connectivity.

We remark that large groups are common in many group messaging scenarios, though rarely supported by end-to-end encrypted messaging systems. For example, Slack supports channels with thousands of members, as does Cisco's enterprise messenger Spark.

Sender Keys The Signal developers implemented a different design for group messaging which they they refer to as the Sender Keys variant [166]. Here, instead of using pairwise secure channels for every message agents use them only to send a symmetric encryption key to the group. They can then send a transport server a single copy of messages encrypted under this "sender key", and the server can "fan out" the message to each recipient as depicted in Figure 5.1b.

This design has many advantages: it is efficient, simple, and scales well even to large groups. Indeed, it is used by WhatsApp[1], Facebook Messenger Secret Conversations and Allo for all groups of size three or greater, using their existing

[1] WhatsApp implements sender keys for group conversations but uses a different design to support multiple devices in a two-party conversation. There, it has a single "primary" mobile phone, and allows secondary devices to connect by scanning a Quick Response (QR) code. When Alex sends a message from a secondary device, WhatsApp first sends the message to Alex's mobile phone, and then over the pairwise Signal channel to the intended peer. While this method does allow for multiple device functionality, it suffers from the downside that Alex cannot use WhatsApp without an online mobile phone, even if another device is connected.

support for Signal in pairwise channels. In both implementations, new sender keys are broadcast whenever a participant is removed from a group but otherwise are only ever passed through a symmetric ratchet.

However, using sender keys sacrifices some of the strong security properties achieved by the Double Ratchet: if an adversary ever learns a sender key, it can subsequently eavesdrop on all messages and impersonate the key's owner in the group, even though it could not do so over the pairwise channels (whose keys are continuously updated). Thus, sender keys do not provide PCS.

Regularly broadcasting new sender keys over pairwise Signal channels prevents this type of attack. However, since a new sender key message must be sent separately to each group member, this returns to linear scaling in the size of the group for a given key rotation frequency, with all the same problems as above.

n-party DH One might wonder whether it is possible to generalise Signal's two-party ratchet by using an n-way DH-like primitive: one which given all of $g^{x_0}, \dots, g^{x_n}$ and a single x_i $(i \leq n)$, derives a value *grk* which is hard to compute without knowing one of the x_i.

With $n = 3$ Joux [97] gives a pairing-based construction of exactly such a primitive. However, for general n it is a known open problem to do so from standard assumptions. Boneh and Silverberg [35] essentially generalise the Joux protocol with a construction from the nonstandard assumption of a $(n-1)$-non-degenerate linear map on the integers. Boneh and Zhandry [36] present a construction from indistinguishability obfuscation (iO), and recent work by Ma and Zhandry [121] formalises the concept as an "encryptor combiner" and gives constructions from iO or from certain lattice assumptions.

5.2 Group Key Exchange

While n-way DH-like primitives are out of reach, the idea remains to use a group key exchange protocol as a building block for a Signal-like ratcheting scheme, with new keys generated periodically by group members and the group key exchange repeatedly applied to derive a new shared secret for the whole group. In this chapter we will show that this idea leads to a practical design for an efficient group messaging protocol with PCS.

We will use *tree-based DH* protocols as one of our core building blocks.

5.2.1 Tree-based group DH

There is a very large body of literature on tree-based group key agreement schemes. An early example is the "audio teleconference system" of Steer et al. [158], and the seminal academic work is perhaps Wallner, Harder and Agee [165] or Wong, Gouda and Lam [168]. Later examples include [40, 54, 101, 102, 103, 116, 169], among many others.

These protocols share the same observation: while an n-way DH shared secret cannot be computed *directly* from n public keys, it can be done *iteratively*, by repeatedly computing the DH shared secret of a pair of keys and reinterpreting the result as itself a DH secret key. We define the notation $\iota(\cdot)$ for this reinterpretation; $\iota(\cdot)$ maps group elements to integers, and we'll instantiate it later in the analysis with a random oracle.

This computation can be thought of as assigning private DH keys to leaves of a binary tree, defining

(i) g^{xy} as the secret key of a node whose two children have secret keys x and y, and

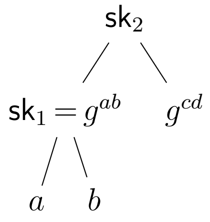
(ii) $g^{\iota(g^{xy})}$ as its public or 'blinded' key.

Recursively computing secret keys through the tree, starting from the leaves, yields a value at the root which we call the "tree key". This value has the DH property that we wanted, namely, that it can only be computed with knowledge of at least one secret leaf key. (A similar construction can be done with ternary trees for the three-party Joux protocol, using a bilinear pairing.)

For example, if A knows a secret key a and public keys g^b, g^c, g^d, they can compute

$$\begin{aligned} \mathsf{sk}_1 &= (g^b)^a = g^{ab} & \mathsf{pk}_1 &= g^{\mathsf{sk}_1} \\ \mathsf{sk}_2 &= (g^{\iota(g^{cd})})^{\mathsf{sk}_1} = g^{\iota(g^{ab})\iota(g^{cd})} & \mathsf{pk}_2 &= g^{\mathsf{sk}_2} \end{aligned}$$

or, drawn as a tree with nodes labeled by secret keys,



A derives $\mathsf{sk}_1 = g^{ab}$ using the public key g^b which was initially distributed. However, in order to compute $\mathsf{sk}_2 = (g^{g^{cd}})^{\mathsf{sk}_1}$ A must know not just the secret key g^{ab} but also the public key $g^{g^{cd}}$, which was *not* initially distributed. Moreover, $g^{g^{cd}}$ can only be computed with knowledge of either c or d; A cannot compute it alone.

More generally, in order to compute the secret key at the root of the tree—the tree key—an agent must know

(i) one secret leaf key λ_j, and

(ii) all public node keys pk_1 to pk_n along the *copath* of λ_j's node,

where the copath of a node is the list of sibling nodes along its path to the tree root. The group key is computed by alternately exponentiating the next public key with the current secret, and applying an injection from group elements to integers. The requirement for copath knowledge demonstrates that knowing the public leaf keys alone is not enough to compute the tree key.

How can A learn the values along their copath, g^{cd} in our example? Existing tree-based designs include a number of online rounds in which agents take turns broadcasting enough information for all group members to learn all public keys on their copath. For example, Kim, Perrig and Tsudik [103] describe a system where certain agents “sponsor” their subtrees, broadcasting the public keys which they know. In our simple example, A might broadcast $g^{g^{ab}}$, then C might broadcast $g^{g^{cd}}$, and then all group members have enough information to compute the tree key.

These designs are thus intrinsically synchronous: they require all parties to come online at the same time for the initial key exchange, in order to broadcast or exchange intermediate public keys in the tree. This is not a problem in their context of XMPP-style instant messaging, but poses a problem in our context of mobile and unreliable networks. In §5.4 we will construct a design for tree-based group key exchange which is asynchronous, allowing an agent to create a group and derive a group key without waiting for its peers to broadcast any information.

First, however, we quickly survey some other well-known protocols for group key exchange.

5.3 Threat Model

We discussed a context and threat model for modern messaging services in §3.1 on page 38, and we will not duplicate that discussion here. In the group context, we also aim for secrecy and authentication of group keys, where a authentication now means that only group members can derive the key. As in the two-party case, we have two novel requirements, one functional and one security:

(i) **asynchronicity**: no group operation should require any pair of group members to be online concurrently

(ii) **PCS**: if a group member is compromised but later performs a certain group operation, the adversary should not be able to derive the resulting group key

Unlike in the two-party case, however, there are many properties which are only meaningful in groups larger than two. In general many of these properties have been thoroughly discussed in the literature, and we do not aim to satisfy all of them with our work. Our goal here is to provide a provably-secure design for an asynchronous group messaging system, and to do so in a way that existing techniques for other features of group messaging should be applicable in a relatively straightforward manner.

In the interest of transparency, we explicitly name some properties which we do not set out to solve in this work, referring the reader to other research or designs. We do not mean to imply that these problems are unimportant or their solutions unnecessary—rather, merely that we are not setting out to solve them in this work. In many cases, a solution will indeed be necessary in a large-scale practical deployment.

As we will see later, our designs build on well-studied DH tree based systems, thereby enabling the reuse of existing solutions as components.

5.3.1 Properties Out of Scope

Sender-specific authentication With two parties, it is relatively straightforward to reason about authentication based on a shared key. Indeed, if Alex receives a message m they did not send, authenticated under a key whose only other holder is Blake, they can safely conclude that Blake must have sent m.

In a group, this conclusion is not quite so straightforward. Indeed, if Alex receives a message m they did not send, authenticated under a key whose only other holders are Blake and Charlie, they can only safely conclude that *either* Blake or Charlie sent m but not which one.

Depending on the context, this may not be a desirable property of a group messaging system. In Multi-Party Off-the-Record Messaging (mpOTR) it is considered a feature as a form of deniability: Alex can deny having sent any particular message, claiming that any other group member could also have produced it. In implementations of Signal's Sender Keys protocol this property is explicitly ruled out with signatures: each participant broadcasts a signature public key (over a deniable channel), and signs all of their messages under that key. We choose the simpler option and do not include signature keys, discussing this topic further in §5.6.

Centralised, unencrypted group messaging systems usually provide individual authentication via the service provider's accounts. For example, Facebook Messenger group chats do not allow Blake to impersonate Charlie, because Blake must log into a Facebook account to send a message. We do not assume such a trusted third party in our analyses. Of course, an encrypted messaging system can *also* include authentication from a third party, as with e.g. Facebook's Secret Conversations.

Malicious group members In the two-party case, security properties generally assume that the peer to a session is honest. Group properties may also assume that all peers to a session are honest, but with $n > 2$ parties there is also an intermediate assumption: $m < n$ parties may be honest. Different protocols may or may not provide guarantees in this latter case. For example, Abdalla et al. [2] give a group key exchange protocol which allows subsets of the group to derive keys known only to that subset. They aim for security in a subset even if a group member outside of that subset is malicious, resisting attacks such as Eli adding Charlie but claiming to have added Blake, or adding Charlie but trying to hide the fact.

Although these properties are useful, we consider them orthogonal to our core research question and we do not consider malicious group members. Moreover, because we use standard constructions from the (synchronous) literature, we anticipate that extending our design to handle e.g., subset communications should be relatively straightforward.

Malicious creators A particular example of a malicious insider is the group creator, who may be able to choose evil initial values. For example, a group creator might be able to secretly add an eavesdropper to a group without revealing their presence to the other (honest) group members. That is, Alex might create a group and claim it contains only Blake and Charlie, but secretly also include Eli. As for malicious insiders in general, we regard this context as out of scope. (We remark that

Alex Do you want to leak those government secrets?
Alex Oops, I mean *read those government speeches!
Blake Sure!

Alex Do you want to leak those government secrets?
Blake Sure!
Alex Oops, I mean *read those government speeches!

Figure 5.2: Two transcripts with different message ordering can have very different meanings!

a malicious insider could also publish received messages regardless of the underlying protocol.)

Executability Cohn-Gordon, Cremers and Garratt [57] point out a possible failure mode of stateful protocols: in some protocols, it is possible for an agent to enter a state in which they are unable to complete any sessions. This is not a problem for stateless protocols, for which if one session can be completed then a subsequent one can be completed as well.

A particular instantiation of this problem is desynchronisation of state in stateful group messaging protocols. If Blake attempts to update the state without realising that Alex has already performed an update, Blake may lose track of the current group key. In particular, if Alex and Blake both send a key update at the same time, only one can consistently be applied; this does not violate any secrecy properties, but may break availability if updating a key is necessary to send a message. In some cases, this may lead to a denial of service attack on the protocol.

We remark on two main techniques to avoid trivial denials of service, though a perfect solution is an open research question (studied e.g. by Chen and Tzeng [54]) and we consider it out of scope for our work.

The first technique is to separate state changes from message transport: once Blake has derived a valid message encryption key, the protocol may accept messages sent under that key for a short duration even if it expects Blake to have performed a state update. This allows Blake to send messages immediately while in the background performing a recovery process to return to the latest group state, at the cost of weakened security guarantees due to the extended key lifetime.

A second solution is at the transport layer, either by enforcing in-order message delivery or by refusing to accept out-of-order key updates and instead delivering the latest group state. That is, when the transport layer server receives a state update from Blake which was generated based on an out-of-date state, it can refuse to accept it and instead instruct Blake to process the latest updates and retry. Since this enforcement can operate based only on message metadata, a malicious transport server can then violate availability but not message confidentiality or integrity. This solution works fine for many group sizes, but in very large groups may cause a server performance bottleneck.

Transcript agreement In many scenarios it is valuable for all group participants to agree on the ordered list of messages that were sent and received in the group; Figure 5.2 is a silly example of two transcripts containing the same messages but with rather different meanings.

Various different messaging protocols have implemented their own solutions to transcript consistency. For example, some of the OTR family have a "tear down" operation when a chat finishes in which all participants compare a hash of the entire transcript. This will detect differences such as the above, though cannot identify where in the conversation the disagreement may have taken place.

Although transcript agreement is a useful property, it has many subtleties that are orthogonal to our key research questions and we do not cover it here.

5.4 Asynchronous Ratcheting Trees

We now give a design which we call Asynchronous Ratcheting Trees (ART). ART combines the structures from DH trees with some concepts from Signal and some novel ideas, in order to provide a group messaging protocol with PCS and which works fully asynchronously.

DH trees provide many of the features we will want, but recall two key problems: they require interactive communication to create, and they do not have any specified way to update a group key short of creating a new tree. We now give informal descriptions of two ART protocols for noninteractive setup and updates, showing how they solve these problems.

5.4.1 ART Setup

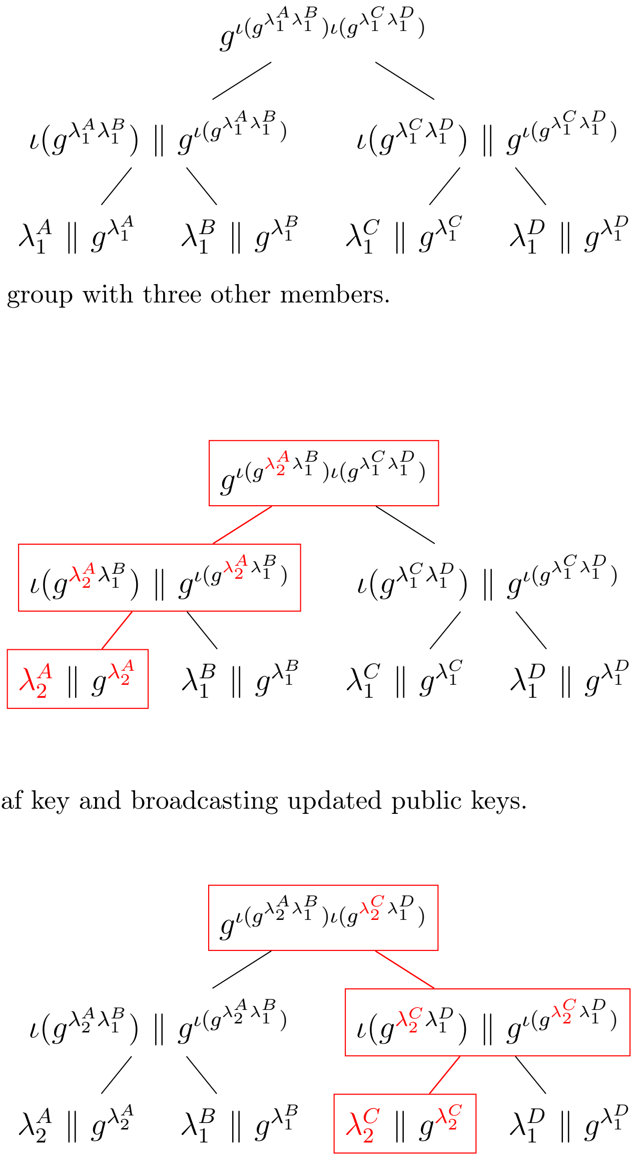
As discussed in §5.2.1, distributing the public keys on each agent's copath has normally led to a number of interactive rounds in previous tree DH protocols. Indeed, an agent who knows a leaf key λ can compute *grk* only if they know the public keys on their copath keys, but cannot compute those public keys even if they know all public keys at leaf nodes.

To avoid this interaction, we use asymmetric *prekeys* together with a one-time asymmetric *setup key*.

Prekeys were first introduced by Marlinspike [123] for asynchronicity in the TextSecure messaging app, and subsequently adopted by the Signal protocol (§2.1.3 on page 11). Recall that they are DH ephemeral public keys cached by an untrusted intermediate server, and fetched on demand by messaging clients. The prekeys are sent to clients through the public key infrastructure at the same time as long-term identity keys, act as initial messages for a one-round authenticated key exchange protocol, and allow for handshakes with offline peers.

For ART we introduce in addition a one-time DH *setup key*, generated locally by the creator of a group and used only during session creation. This key is used to perform an initial key exchange with the prekeys, and allows the initiator to generate secret leaf keys for the other group members while they are offline.

Asynchronous tree construction works roughly as follows. The initiator Alex generates a new leaf key λ_j for each group member, by fetching a prekey for them and performing a noninteractive key exchange. Knowing all the initial λ_j, Alex can compute all intermediate nodes in the tree, and thus can tell each group member the public keys which they need to compute the key at the root of the tree.


$g^{\iota(g^{\lambda_1^A\lambda_1^B})\iota(g^{\lambda_1^C\lambda_1^D})}$
$\iota(g^{\lambda_1^A\lambda_1^B}) \parallel g^{\iota(g^{\lambda_1^A\lambda_1^B})}$
$\iota(g^{\lambda_1^C\lambda_1^D}) \parallel g^{\iota(g^{\lambda_1^C\lambda_1^D})}$
$\lambda_1^A \parallel g^{\lambda_1^A}$
$\lambda_1^B \parallel g^{\lambda_1^B}$
$\lambda_1^C \parallel g^{\lambda_1^C}$
$\lambda_1^D \parallel g^{\lambda_1^D}$
$g^{\iota(g^{\lambda_2^A\lambda_1^B})\iota(g^{\lambda_1^C\lambda_1^D})}$
$\iota(g^{\lambda_2^A\lambda_1^B}) \parallel g^{\iota(g^{\lambda_2^A\lambda_1^B})}$
$\iota(g^{\lambda_1^C\lambda_1^D}) \parallel g^{\iota(g^{\lambda_1^C\lambda_1^D})}$
$\lambda_2^A \parallel g^{\lambda_2^A}$
$\lambda_1^B \parallel g^{\lambda_1^B}$
$\lambda_1^C \parallel g^{\lambda_1^C}$
$\lambda_1^D \parallel g^{\lambda_1^D}$
$g^{\iota(g^{\lambda_2^A\lambda_1^B})\iota(g^{\lambda_2^C\lambda_1^D})}$
$\iota(g^{\lambda_2^A\lambda_1^B}) \parallel g^{\iota(g^{\lambda_2^A\lambda_1^B})}$
$\iota(g^{\lambda_2^C\lambda_1^D}) \parallel g^{\iota(g^{\lambda_2^C\lambda_1^D})}$
$\lambda_2^A \parallel g^{\lambda_2^A}$
$\lambda_1^B \parallel g^{\lambda_1^B}$
$\lambda_2^C \parallel g^{\lambda_2^C}$
$\lambda_1^D \parallel g^{\lambda_1^D}$

Alex generates a new ART tree with

$$\lambda_1^B = \mathrm{AKE}(ik_a, suk, ik_b, ek_b)$$
$$\lambda_1^C = \mathrm{AKE}(ik_a, suk, ik_c, ek_c)$$
$$\lambda_1^D = \mathrm{AKE}(ik_a, suk, ik_d, ek_d)$$

and broadcasts all the public keys.

(a) Alex creates an ART group with three other members.

Alex updates their key by choosing a new leaf key λ_2^A, computing the updated nodes

$$g^{\lambda_2^A\lambda_1^B} = \left(g^{\lambda_1^B}\right)^{\lambda_2^A}$$
$$g^{\iota(g^{\lambda_2^A\lambda_1^B})\iota(g^{\lambda_1^C\lambda_1^D})} = \left(g^{\iota(g^{\lambda_1^C\lambda_1^D})}\right)^{\iota(g^{\lambda_2^A\lambda_1^B})}$$

on the path from λ_2^A to the tree root, and broadcasting the updated public keys to the group.

(b) Alex updates, choosing a fresh leaf key and broadcasting updated public keys.

Charlie updates their key in the same way: by choosing a new leaf key λ_2^C, computing the updated nodes on the path from λ_2^C to the tree root, and broadcasting the updated public keys to the group.

(c) Charlie updates their key in the same way.

Figure 5.3: Example ART tree creation and updates. We write secret keys and the corresponding public keys at each node, separated by $\parallel$. Leaf keys are denoted λ_i^u, where u is the corresponding identity and i a counter. $\iota(\cdot)$ denotes a mapping from group elements to integers. From any secret leaf key and the set of public keys on its copath, an agent can compute the tree key by repeated exponentiation.

In more detail, suppose Alex wishes to create a group of size n containing Alex and $n-1$ peers. Alex begins by generating a DH key $SUK = g^{suk}$ we call the setup key, and then requests from the public key infrastructure an identity key ik and an ephemeral prekey ek for each intended peer ("Blake", "Charlie", …), numbering them 1 through $n-1$. Using the secret identity key ik_a and the setup key suk together with the received keys for each peer, Alex executes a one-round authenticated key exchange protocol to derive leaf keys $\lambda_1^B, \lambda_1^C \ldots, \lambda_1^{u_n-1}$. Using these generated leaf keys together with a freshly-generated leaf key λ_1^A, Alex can build a DH tree whose root holds the initial group key.

We write $\mathrm{AKE}(ik_i, ek_i, ik_r, ek_r)$ for the session key derived by this one-round AKE protocol with identity keys ik_i/ik_r and ephemeral keys ek_i/ek_r for the initiator and responder respectively. Note that the initiator and the other group members compute $\mathrm{AKE}(\cdots)$ differently: the initiator uses the private keys corresponding to the first two arguments, and the responders use the private keys corresponding to the second two arguments. The resulting key is depicted in Figure 5.3a.

We will not yet force a particular instantiation of this one-round key exchange protocol. For example, it can be instantiated with an unauthenticated DH exchange between Alex's setup key and Blake's prekey (ignoring the identity keys), resulting in an unauthenticated tree structure. This is the design we analyse in §5.5.3. A more practical instantiation is with a strong authenticated key exchange protocol, which we briefly discuss there.

To share the initial group key, Alex sends

(i) the public prekeys (ek_i) and identities (ik_i) used to create the group,

(ii) the public setup key SUK,

(iii) the tree T of public keys, and

(iv) a signature of the previous data (i), (ii), (iii) under Alex's identity key.

When they receive this data and verify the signature, each group member can reproduce the computation of the tree key:

(i) they compute their leaf key λ_i using the secret ephemeral key corresponding to their public ephemeral key used by Alex in AKE,

(ii) they extract their copath of public keys from the tree T, and

(iii) they iteratively exponentiate with the public keys on the copath until they reach the final key, which by construction is the tree key tk.

At this stage tk is a shared symmetric secret key among all the members of the group, and can be used as input to a key schedule to derive message encryption and authentication keys.

We give pseudocode definitions in Figure 5.5 (Algorithms 1 to 3).

5.4.2 ART Updates

To achieve PCS, we must be able to *update* shared group keys in a way that depends both on state from previous stages and on newly exchanged messages. (Cohn-Gordon, Cremers and Garratt [57] prove necessity of this double dependency.) Since PCS is an explicit goal of ART, it must therefore support an efficient mechanism for any

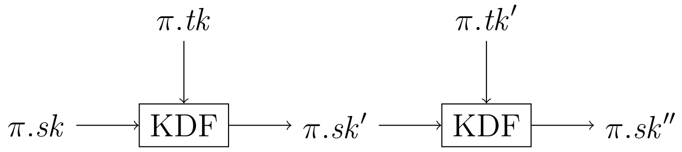


Figure 5.4: Derivation of stage keys $\pi.sk$. When a new tree key $\pi.tk$ is computed (as the root of a DH tree), it is combined with the current stage key to derive a new stage key $\pi.sk'$, etc. This "chaining" of keys is an important ingredient for achieving PCS. Note that the ART KDF also includes π.IDs and $\pi.T$, per Algorithm 2 on the facing page.

group member to update their key.

Key trees have the important property that changing a single leaf key only requires changing logarithmically many other values in the tree, namely, the ancestors of the changed leaf key. This means that if, say Alex generates and broadcasts a new leaf key, other group members can compute all the intermediate values in the resulting updated tree using only (i) their view of the tree before the change, and (ii) the list of updated public DH keys of nodes along the path from Alex's leaf node to the root of the tree. Since Alex can compute (ii) in logarithmic time and broadcast it to the group along with the new leaf key, this update is both efficient and asynchronous, allowing for fast, noninteractive group key updates.

Specifically, if at any point a participant wishes to change leaf key from λ_b to λ'_b, they can recompute the new public keys after the change, at all nodes along the path from their leaf to the tree root. They can then broadcast to the group their new public leaf key and all of the recomputed intermediate keys, authenticating the update with a MAC under the previous shared group key.

A group member who receives such a message can apply the update to their stored copath (at the node on the intersection of the two paths to the root). Computing the key induced by this new copath yields the updated group key, again without requiring any two group members to be online at the same time.

We give a pseudocode definition of these algorithms in Figure 5.5 and algorithms 4 and 5.

5.4.2.1 Stage key chaining

In order to achieve PCS, message encryption keys cannot be independent—instead, each key must depend both on fresh secrets and on previous stages. As long as one of these two sources of secret data is unknown to the adversary, the key derived by the stage will be as well. For ART we use the same hash chain design as Signal, depicted here in Figure 5.4. A master ("stage") key is derived at each stage from a KDF applied to two inputs: the previous stage key and the key at the root of the current tree.

5.4.3 Algorithms

We give pseudocode algorithms for all of the operations in our design in Figure 5.5. Suppose Alex wishes to create a group with four other agents, using Algorithm 1.

Algorithm 1 Asynchronous group setup

```
procedure SetupGroup($((ik_i, ek_i))_{i=1}^{n-1}$)
    // set up a group with $n-1$ identity and ephemeral keys of peers
    $\pi.\lambda \xleftarrow{\$}$ DHKeyGen()
    $suk \xleftarrow{\$}$ KeyExchangeKeyGen()
    for $i \leftarrow 1 \ldots n-1$ do // generate leaf keys for each agent
        $\lambda_i \leftarrow \iota(\text{KeyExchange}(\pi.ik, ik_i, suk, ek_i))$
        add $(\lambda_i, sid(\pi, \sigma))$ to $\sigma$.HKeys
    $T_{\text{secret}} \leftarrow \text{CreateTree}(\pi.\lambda, \lambda_1, \ldots, \lambda_{n_{\text{peers}}})$
    $\pi.T \leftarrow \text{PublicKeys}(T_{\text{secret}})$
    $\pi.\text{IDs} \leftarrow \pi.ik, ik_1, \ldots, ik_{n_{\text{peers}}}$
    $\pi.\text{EKs} \leftarrow \pi.ek, ek_1, \ldots, ek_{n_{\text{peers}}}$
    $x \leftarrow \pi.\text{IDs}, \pi.\text{EKs}, SUK, \pi.T$
    $m \leftarrow (x, \text{Sign}(x; \pi.ik))$
    $\pi.tk \leftarrow (T_{\text{secret}})_{0,0}$
    $\pi.\text{idx} \leftarrow 0$
    $\sigma.\ell[\pi.\text{idx}] \leftarrow 1$
    $\sigma.\ell[x] \leftarrow 0$ for $0 < x \leq n$
    $\pi.\bar{P} \leftarrow \text{Copath}(T, 0)$
    $\pi.sk = 0$
    DeriveStageKey()
    return $m$
```

Algorithm 2 Helper functions

```
function LeftSubtreeSize($x$)
    // height of the left subtree if there are $x$ elements
    return $2^{\lceil \log_2(x) \rceil - 1}$

function CreateTree($\lambda_0, \lambda_1, \ldots, \lambda_n$) // tree with $n$ leaves
    if $n = 0$ then return (leaf, $\lambda_0$)
    $h \leftarrow \text{LeftSubtreeSize}(n)$
    $(L, lk) \leftarrow \text{CreateTree}(\lambda_0, \ldots, \lambda_{(h-1)})$ // complete left subtree
    $(R, rk) \leftarrow \text{CreateTree}(\lambda_h, \ldots, \lambda_{n-1})$ // right subtree
    $k \leftarrow \iota(LK^{rk})$
    return (node($(L, lk), (R, rk)$), $k$)

function PublicKeys($T \leftarrow$ node($L, R$), $k$)
    if $T = \emptyset$ then return $\emptyset$
    return node(PublicKeys($L$), PublicKeys($R$)), $g^k$

function Copath($T, i$) // where $i$ is the index of the leaf and $|T| = \#$leaves
    if $i < \text{LeftSubtreeSize}(|T|)$ then // $i$ is in the complete left subtree
        return $g^{T_{1,1}}$, Copath($T_{1,0}, i$)
    else// $i$ is in the possibly incomplete right subtree
        return $g^{T_{1,0}}$, Copath($T_{1,1}, i - 2^l$)

function PathNodeKeys($\lambda, \bar{P}$) // leaf key and the copath of public keys
    $nks_{|\bar{P}|} \leftarrow \lambda$
    for $j \leftarrow (|\bar{P}| - 1) \ldots 1$ do
        $nks_j \leftarrow \iota((\bar{P}_j)^{nks_{j+1}})$
    return $(\bar{P}_0)^{nks_1}, nks_1, \ldots, nks_{|\bar{P}|}$

function DeriveStageKey
    $\pi.sk \leftarrow \text{KDF}(\pi.sk, \pi.tk, \pi.\text{IDs}, \pi.T)$
    return
```

Algorithm 3 Receiving a setup message as agent at index i

```
procedure ProcessSetupMessage($m$)
    $x, s \leftarrow m$
    assert SigVerify($x, s, (x.\text{IDs})_0$)
    assert $\pi.u \in x.\text{IDs}$
    $(\pi.\text{IDs}, \pi.T) \leftarrow (x.\text{IDs}, x.T)$ // store agent ids and copath in state
    $ek \leftarrow$ ephemeral prekey corresponding to $ek$ from $\pi$
    $\pi.\lambda \leftarrow \iota(\text{KeyExchange}(\pi.ik, (\pi.\text{IDs})_0, ek, x.SUK))$
    $nks \leftarrow \text{PathNodeKeys}(\pi.\lambda, \pi.\bar{P})$
    $\pi.tk \leftarrow nks_0$ // store initial tree key
    $\pi.\text{idx} = i$
    $\sigma.\ell[0] \leftarrow 1$
    $\sigma.\ell[x] \leftarrow 0$ for $0 < x \leq n$
    $\pi.sk = 0$
    DeriveStageKey()
    return
```

Algorithm 4 Agent updating their key

```
procedure UpdateKey
    $\pi.\lambda \xleftarrow{\$}$ DHKeyGen()
    $\sigma.\ell[\pi.\text{idx}] \leftarrow \sigma.\ell[\pi.\text{idx}] + 1$
    add $(\pi.\lambda, sid(\pi, \sigma))$ to $\sigma$.HKeys
    $nks \leftarrow \text{PathNodeKeys}(\lambda, \pi.\bar{P})$
    $x \leftarrow \pi.\text{idx}, NKS_1, \ldots, NKS_{|\bar{P}|}$
    $\pi.tk \leftarrow nks_0$
    $m \leftarrow x, \text{Mac}(x; \pi.sk)$
    $\sigma.t \leftarrow \sigma.t + 1$
    DeriveStageKey()
    return $m$
```

Algorithm 5 Processing another agent's key update

```
procedure ProcessUpdateMessage($m$)
    $x, \mu \leftarrow m$
    assert MACVerify($x, \mu, \pi.sk$)
    $j$, nks $\leftarrow x$
    $\nu \leftarrow \text{IndexToUpdate}(\lceil \log_2 n \rceil, 0, \pi.\text{idx}, j)$
    $\pi.\bar{P}_\nu \leftarrow U_\nu$ // index $\nu$ of the copath has been updated in this message
    $nks \leftarrow \text{PathNodeKeys}(\pi.\lambda, \pi.\bar{P})$
    $\pi.tk \leftarrow nks_0$
    $\sigma.\ell[j] \leftarrow \sigma.\ell[j] + 1$
    $\sigma.t \leftarrow \sigma.t + 1$
    DeriveStageKey()
    return

function IndexToUpdate($h, n, i, j$)
    if $(i < 2^{h-1}) \wedge (j < 2^{h-1})$ then // both are in the left subtree
        return IndexToUpdate($h-1, n+1, i, j$)
    else if $(i \geq 2^{h-1}) \wedge (j \geq 2^{h-1})$ then // both in the right subtree
        return IndexToUpdate($h-1, n+1, i-2^{h-1}, j-2^{h-1}$)
    return $n$ // otherwise return index where they differ
```

Figure 5.5: Pseudocode descriptions of the algorithms in our ART design. Informal explanations can be found in §5.4. We use **procedure** to denote subroutines which are used by the protocol algorithms PSend and PRecv, and **function** to denote ones which are not. Procedures operate on and mutate the agent's current state π and σ, receive an optional input message and return an optional output message, which are received from and returned to the adversary. When sending a tuple, we implicitly uniquely encode it as a bitstring (to avoid type confusion errors), and when receiving one we uniquely decode it.

Alex begins by generating a setup keypair with secret key suk, and a leaf keypair with secret key λ_1^A, then retrieves the public identity and ephemeral prekeys of each peer and creates the tree in Figure 5.3a.

Alex then sends each agent their respective copath and the prekey used to set them up in the tree, along with the identities of the other group members and the public setup key. Alex stores the leaf key, the ordered list of public identity keys, the tree key, and the copath, and finally derives the stage key used for messaging via DeriveStageKey in Algorithm 2. Parsing this message (Algorithm 3) allows a recipient to identify their position in the group tree, and to construct the group key using DeriveStageKey.

To update a key, any group member can run Algorithm 4, generating a new leaf key λ' and recomputing their path up to the root. This results in the new tree shown in Figure 5.3c. They then send a key update message containing an index into the tree as well as the path of updated public keys excluding the root, store the updated leaf key and tree key, and computes the new stage key with DeriveStageKey.

Upon receiving this key update message, agents determines their new copath, replacing the relevant existing copath entry with a public key from received message. This is done by executing Algorithm 5. From this, they compute the new tree key, and finally invoke DeriveStageKey to compute the new stage key.

5.4.3.1 Notation

Definition 10 (State)**.** For agent u, session counter i and stage counter t, the *session state* π comprises:

(i) $\pi.u$, the identity u of the current agent
(ii) $\pi.ik$, the identity key of the current agent
(iii) $\pi.ek$, the ephemeral prekeys of the current agent
(iv) $\pi.\lambda$, the leaf key of the current stage
(v) $\pi.T$, the current tree (with ordered nodes) with *public* keys stored at each node
(vi) $\pi.\mathrm{idx}$, the position of the current agent in the group
(vii) $\pi.\mathrm{IDs}$, an ordered list of agent identifiers and leaf keys for the group, where the index of each entry is the index of the corresponding leaf in the tree
(viii) $\pi.tk$, the tree key of the current stage
(ix) $\pi.\bar{P}$, the copath of the current agent

Where there are multiple distinct session states under consideration, we refer to $\pi = \pi(u, i, t)$ as the state of the t^{th} stage of agent u's i^{th} session.

Values in π roughly correspond to variables in a protocol implementation. However, for the security definitions we also keep track of some additional "bookkeeping" state σ. Values in σ are only used for the security game, and do not correspond to variables in a protocol implementation.

Definition 11 (Bookkeeping state)**.** For agent u, session counter i and stage counter t, the bookkeeping state σ of (u, i, t) is an ordered collection of the following variables.

(i) $\sigma.i$, the index of the current session among all sessions with the same agent

(ii) $\sigma.t$, the index of the current stage in the session (initialised to 0 and incremented after each new stage key is computed)

(iii) $\sigma.sk$, the agent's secret stage key to be used at the current stage

(iv) σ.status, the execution status for the current stage. Takes the value `active` at the start of a stage, and later set to either `accept` or `reject` when the stage key is computed

(v) $\sigma.sessk$, the key output by the current stage

(vi) σ.HKeys, the set of ephemeral keys honestly generated in the current stage

(vii) $\sigma.\ell[i']$, the number of leaf keys received so far from node i' in $\pi.T$ (when $i' = \pi.\mathrm{idx}$, this is the number of leaf keys that (u, i) has generated so far).

Definition 12 (sid)**.** By $sid(\pi, \sigma)$ we mean the triple $(\pi.u, \sigma.i, \sigma.t)$. Agents are unique, session counters monotonically increase and session state does not change without the stage changing. Therefore, such a tuple (u, i, t) uniquely identifies states π and σ if they exist.

5.5 Security Analysis

In this thesis, we give a computational security model and proof for an unauthenticated instantiation of ART. To do so, we build a computational security model for multi-stage group key exchange protocols similar to the one we used for Signal, but supporting more than two participants in a conversation. We instantiate this model with an unauthenticated version of ART in which the initial leaf keys are derived directly from the setup key and prekeys. This allows us to capture the core security properties of the key updates, including PCS, without focusing on the properties of the authenticated key exchange used for the initial construction. In this model, we sketch a reduction showing indistinguishability of group keys from random values, using a game-hopping argument.

5.5.1 Out of Scope: Authentication

As shown in the algorithms, authentication can be provided by deriving initial leaf keys from a non-interactive key exchange, whose security property also applies to the resulting tree key. We don't aim to give a computational analysis of such a property, because we believe that its complexity would be beyond the scope of current computational proof techniques. In particular, our freshness condition is already fairly complex, and its interaction with a modern AKE model with identity key corruptions would lead to a state space explosion in the number of proof cases.

We see analyses of complex protocols such as authenticated ART as a motivating example for symbolic verification tools. Indeed, a Tamarin analysis can be found in our paper [60]. There, the ratcheting tree structure is abstracted away and modelled as a black box oracle mapping n public leaf keys and one secret leaf key to a group key.

We remark that the authentication property does not follow trivially from using an AKE protocol at the leaves: TAMARIN found an attack against an earlier design of ART which did not provide integrity protection for the initial group-creation messages. In this attack Alex constructs a group containing Blake by fetching prekeys as usual, computing a group key using an authenticated key exchange at the leaf nodes, and sending an initial message to Blake. However, an active adversary modifies this initial message to add a malicious leaf key to the group. Blake, unable to detect the modification due to the lack of integrity protection, willingly constructs a group containing the adversary, who can subsequently read all messages sent by Blake. The attack is prevented by authenticating the initial message.

5.5.2 Computational Model

We build on the multi-stage definition of Fischlin and Günther [82], in which sessions admit multiple stages with distinct keys and the adversary can Test any stage. This is similar to the model described in §3.3, but we must extend to group messaging by allowing multiple peers for each session. As before, our model defines a *security experiment* as a game played between a challenger and a pptm adversary. The adversary is given a set of queries through which it can interact with the challenger, including the ability to relay or modify messages but also to compromise certain secrets, and eventually chooses a so-called Test session and stage, receiving uniformly at random either its true key or a random key sampled from the same distribution. It must then decide which it has received, winning the game if the guess is correct. Thus, a protocol which is secure in this model enjoys the property that an adversary cannot tell if the true keys are replaced with random values.

Our model for Signal, as with most similar key exchange models, uses Activate and Run algorithms to encode the adversary's interaction with the protocol algorithms. That is, if the adversary wishes Alex third session to send a message to Blake, say, it invokes a Send query with some arguments, which the challenger then uses to invoke the Run algorithm. In Signal, there are two possible actions that the Run algorithm must support: receiving a message (if the Send query was passed the message as an argument), or sending a message (otherwise). However, for ART there are multiple actions that an agent can perform: for example, whether or not to perform a key update when sending a message.

We would like a way for the adversary to instruct the target of its query as to which action to take. To do so, we split the traditional Run algorithm into PRecv ("protocol receive", to receive and process a message from $\mathcal{A}$) and PSend ("protocol send", to receive instructions from and then send a message to $\mathcal{A}$), and split the Send query into ASend ("adversary send", to instruct the challenger to send a message to a target session) and ARecv ("adversary receive", to request a message from a target session). The challenger will respond to ASend queries using the PRecv algorithm and ARecv using PSend.

Definition 13 (Multi-stage key exchange protocol)**.** A multi-stage key exchange protocol Π is defined by a keyspace $\mathcal{K}$, a security parameter λ (dictating the DH

Table 5.1: Adversary queries defined in our model. We use u to denote the agent targeted by a query, i to denote the index of a session at an agent, and t to denote the stage of a session—thus, for example, $(Alex, 3, 4)$ identifies the fourth stage of Alex's third session. We use m for messages and b, b' for bits.

$\mathsf{Create}(u, v_1, \dots v_{n-1})$	Given a set of intended peers $v_1, \dots, v_{n-1}$ $(n \leq \gamma)$, the challenger executes Activate to prepare a new state π, prepares a new bookkeeping state σ with $\sigma.i$ set to the number of times $\mathsf{Create}(u, \dots)$ has already been called, and initialises a new role oracle with states π and σ for agent u.
$\mathsf{ASend}(u, i, m)$	Given a message m and a session (u, i) with state π, execute $\pi' \leftarrow \text{PRecv}(\pi, m)$ and set the session state to π'. u must be a valid agent identifier and $\mathsf{Create}(u, \dots)$ must have been called at least $i - 1$ times. This query models sending a message to a session.
$\mathsf{ARecv}(u, i, \text{data})$	Given a session (u, i) with state π, execute $\pi', m \leftarrow \text{PSend}(\pi, \text{data})$, update the session state to π' and return the message m. This query models a role oracle performing of the protocol's actions.
$\mathsf{RevSessKey}(u, i, t)$	Given (u, i, t), return $\pi.sessk$ where π is the stage with $sid(\pi) = (u, i, t)$ if it exists. This query models keys being leaked to the adversary and is used to capture authentication properties.
$\mathsf{RevRand}(u, i, t)$	Given (u, i, t), reveal the random coins by u in stage t of session (u, i). This query models the corruption of an agent, either in their initial key generation (at $t = 0$) or afterwards $(t > 0)$.
$\mathsf{Test}(u, i, t)$	Given (u, i, t), let k_0 denote the key computed by user u at stage t of session (u, i), and let k_1 denote a uniformly randomly sampled key from the challenger. The challenger flips a coin $b \overset{\$}{\leftarrow} \text{Uniform}(\{0, 1\})$ and returns k_b.
$\mathsf{Guess}(b')$	The adversary immediately terminates its execution after this query.

group size q) and the following probabilistic algorithms:

(i) $(x, g^x) \leftarrow_\$ \text{KeyExchangeKeyGen}()$: generate DH keys

(ii) $\text{Activate}(x, role, peers) \rightarrow \pi$: the challenger initialises the protocol state of an agent u by accepting a long-term secret key x, a role *role* and a list *peers* of peers, creating states π and σ, assigning $\sigma.i$ to the smallest integer not yet used by u, and returning (π, σ)

(iii) $\text{PRecv}(\pi, m) \rightarrow \pi'$: an agent receives a message m, updating their protocol state from π to π'

(iv) $\text{PSend}(\pi, \text{data}) \rightarrow \pi', m$: an agent receives some instructions data and sends a message m, updating their protocol state from π to π'

Definition 14 (Adversary queries)**.** We allow the adversary access to the queries defined in Table 5.1.

We fix a maximum group size γ, which is the largest group that an agent is willing to create. This can be application-specific.

5.5.3 Analysis

We now have enough tools to analyse ART in the model of §5.5.2. In this analysis we do not consider the use of long-term keys, since we are working in an unauthenticated model. Our freshness criteria allow the adversary to corrupt the random values or key from any stage, but rule out trivial attacks created by such corruptions. We define

$$\textsc{KeyExchange}(\pi.ik, \mathrm{IDs}_0, ek, SUK) \leftarrow SUK^{ek}.$$

That is, our initial leaf nodes are constructed unauthenticated from initial ephemeral keys. In this setting we do not need the MACs which are defined in the protocol algorithms, and we do not make any assumptions here on their security properties.

We define $\mathrm{PRecv}(\pi, m)$ as follows. For a session with $\sigma.t = 0$, validate that m is of the expected format for ProcessSetupMessage, and if so then execute ProcessSetupMessage and return the result. For a session with $\sigma.t > 0$, validate that m is of the expected format for ProcessUpdateMessage, and if so then execute ProcessUpdateMessage, returning the result.

We define $\mathrm{PSend}(\pi, \mathrm{data})$ as follows. Validate that data is one of "create-group" or "update-key", or else abort, setting the session state to `reject`. Then, if data is "create-group", execute SetupGroup and return the result; if data is "update-key", execute UpdateKey and return the result.

Definition 15 (Matching)**.** We say that two stages with respective sids (u, i, t) and (v, j, s) *match* if they both have σ.status = `accept` and moreover have derived the same key.

Definition 16 (Freshness of a copath)**.** Let $\bar{P} = \bar{P}_0, \ldots, \bar{P}_{|\bar{P}|-1}$ be a list of group elements representing a copath and let $\Lambda = \lambda_0 \ldots \lambda_{n-1}$ be a list of group elements representing leaf keys. We say that $\bar{P}$ is the i^{th} *copath induced by* Λ precisely if, in the DH tree induced by Λ, each $\bar{P}_j$ is the sibling of a node on the path from λ_i to the tree root, and that $\bar{P}$ is *induced by* Λ if for some i it is the i^{th} copath induced by Λ.

We say that a copath $\bar{P}$ is *fresh* if both

(i) $\bar{P}$ is the i^{th} copath induced by some Λ, and

(ii) for each $g^{\lambda_j} \in \Lambda$, both

  (a) there exists some stage whose $sid(\pi, \sigma) = (u, i, t)$ such that $(\lambda_j, sid(\pi, \sigma)) \in \sigma.\mathrm{HKeys}$, and

  (b) no $\mathsf{RevRand}(u, i, t)$ query was issued.

Intuitively, a copath is fresh if it is built from honestly-generated and unrevealed leaf keys. In particular, the copath's owner's leaf key must also be unrevealed, since it is included in Λ.

Definition 17 (Freshness of a stage)**.** We say that a stage with sid (u, i, t) deriving key *sessk* is *fresh* if

(i) it has status `accept`,

(ii) the adversary has not issued a $\mathsf{RevSessKey}(u, i, t)$ query,

(iii) there does not exist a stage with sid (v, j, s) such that the adversary has issued a query $\mathsf{RevSessKey}(v, j, s)$ whose return value is *sessk*, and

(iv) one of the following criteria holds:

  (a) $t > 0$ and the stage with sid $(u, i, t-1)$ is fresh, or

  (b) the current copath is fresh.

Intuitively, a stage is fresh if *either* all of the leaves in the current tree are honestly generated and unrevealed *or* the previous stage was fresh. The latter disjunct captures a form of PCS: if an adversary allows a fresh stage to `accept`, subsequent stages will also be fresh.

Remark 7 (Session key reveals)*.* We use a minimal form of restriction on RevSessKey queries: the adversary is not permitted to issue any RevSessKey query which reveals the key of the Tested session. This allows us to simplify our freshness condition, in exchange for an additional proof obligation in Definition 19 that only sessions which "should" derive the right session key in fact do. (Otherwise, for example, a protocol which always derives the same random session key would be secure, since the adversary is not permitted to reveal it.)

Remark 8 (Freshness of the group creator's first stage)*.* Our freshness predicate encodes stronger trust assumptions on the initiator's first stage than it does on subsequent updates. Indeed, for the creator's first stage to be fresh we must have that their first copath is fresh; since their first copath depends on all the generated λ_j, which were all added to σ.HKeys during the creator's first stage, the adversary is not permitted to issue a RevRand query against that stage until all the λ_j from the initial stage have been revealed. This captures the additional requirements discussed in §5.3.1.

Capturing strong security properties Our notion of stage freshness captures the strong security properties discussed earlier, by allowing the adversary to Test stages under a number of compromise scenarios. Specifically:

authentication states that if the ephemeral keys used in a stage are from an uncorrupted stage then only the agents who generated them can derive the group key. Indeed, for a stage to be fresh either it or one of its ancestors must have had a fresh copath; that is, one that is built only from λ_j which were sent by other honest stages.

PFS is captured through clause (iv)b and the definition of the RevRand query: suppose Alex accepts a stage t and then updates a key in stage $t + 1$. An adversary who queries $\mathsf{RevRand}(\ldots, t+1)$ does not receive the randomness from stage t, which therefore remains fresh. Our model thus requires the key of stage t to be indistinguishable from random to such an adversary.

PCS is captured through clause (iv)a: suppose the adversary has issued $\mathsf{RevRand}$ queries against all of one of Alex's session's stages from 0 to t *except* some stage $0 \leq j < t$. Absent other queries, stage j is therefore considered fresh, and hence by clause (iv)a stages $j+1, j+2, \ldots, t$ are fresh as well. Our model thus requires their keys to be indistinguishable from random to such an adversary.

ART's asynchronicity constraint means that Alex must be able to send a message to a group which was just created, even if none of the other participants have yet been online. ART's design allows for this, but at a cost: if Alex is corrupted during this initial phase, the resulting stage keys are insecure until all group members have performed an update. We capture this increased requirement in our freshness predicates, and note that one can remove it if all participants are online, by having each one in turn perform a key update. Our approach here is related to that of the Zero Round-Trip (0-RT) mode of TLS 1.3, in which agents can achieve asynchronicity at the cost of a weaker security property for early messages.

Definition 18 (Security experiment)**.** We define the security experiment as follows.

At the start of the game, the challenger generates the public/private key pairs of all $\mathsf{n_P}$ parties and sends all public info including the identities and public keys to the adversary. The adversary then asks a series of queries from Table 5.1 before eventually issuing a $\mathsf{Test}(u, i, t)$ query, for the t^{th} stage of the i^{th} session of user u. We can equivalently think of the adversary as querying oracle machines π_u^i for the i^{th} session of user u.

Our notion of security is that the key of the Tested stage is indistinguishable from random. Thus, after the $\mathsf{Test}(u, i, t)$ query, the challenger samples $b \leftarrow_\$ \{0, 1\}$ and with probability $1/2$ (when $b = 0$) reveals the actual session key of user u's ith session at stage t to the adversary, and with probability $1/2$ (when $b = 1$) reveals a uniformly randomly chosen key instead. The adversary is allowed to continue asking queries. Eventually the adversary must guess the bit b with a $\mathsf{Guess}(b')$ query before terminating. If the Tested (u, i, t) satisfies fresh and the guess is correct $(b = b')$, the adversary wins the game. Otherwise, the adversary loses.

Definition 19 (Partnering experiment)**.** We define the partnering experiment as follows. At the start of the game, the challenger initialises all parties as in the security experiment. The adversary then asks a series of Create, ASend or ARecv queries, and eventually terminates. There is no additional model state and no other queries are permitted.

When the game ends, the adversary wins if and only if for any session (u, i, t) with $(u, i, t).\sigma.\text{status} = \texttt{accept}$, any of the following hold.

(i) disagreement on group members: there exists another stage (v, j, s) deriving the same key as (u, i, t) but with $(u, i, t).\text{IDs} \neq (v, j, s).\text{IDs}$

(ii) incorrect peer: there exists a stage (v, j, s) deriving the same key as (u, i, t) with $v \neq u$ and $v \notin (u, i, t).\text{IDs}$

(iii) repeated session key: there exists another session (u, i', t'), $i' \neq i$, deriving the same key as (u, i, t)

(iv) too many copies of a peer: for any peer identity v appearing $n > 0$ times in (u, i, t).IDs, $v \neq u$, there exist $n + 1$ stages $(v, \cdot, \cdot)$ deriving the same key as (u, i, t)

We say that a multi-stage key exchange protocol is *secure* if the probability that any probabilistic polynomial-time adversary wins the security experiment (resp. the partnering experiment) is bounded above by $1/2 + \text{negl}[\lambda]$ (resp. $\text{negl}[\lambda]$), where $\text{negl}[\lambda]$ tends to zero faster than any polynomial in the security parameter λ. We now give our theorem and sketch the proof.

Lemma 2. *The success probability of any ppt adversary against the partnering experiment of our protocol is negligible in the Random Oracle Model (ROM).*

Proof sketch. The result follows directly from the fact that π.IDs is an argument to the KDF when deriving stage keys. If the KDF is a random oracle its output values do not collide, and thus equal output values imply equal input values, which is enough to rule out the cases in the partnering security experiment.

Suppose there exists an adversary $\mathcal{A}$ which wins the partnering security game. By definition, it wins if one of the four cases occurs and we consider each one in turn.

First, suppose that it wins because there exist two stages (u, i, t) and (v, j, s) deriving the same key but with (u, i, t).IDs $\neq (v, j, s)$.IDs. The stage key is derived as $\pi.sk \leftarrow \text{KDF}(\pi.sk, \pi.tk, \pi.\text{IDs}, \pi.T)$. In particular, equality of stage keys implies equality of IDs, so this case is impossible.

Second, suppose that it wins because there exists a stage (v, j, s) deriving the same key as (u, i, t) but with $v \notin (u, i, t)$.IDs. As in the first case, we know that (u, i, t).IDs $= (v, j, s)$.IDs, and thus $v \notin (v, j, s)$.IDs. However, this contradicts the fact that agents always believe they are in their own groups, so this case is impossible.

Third, suppose that there exist two sessions (u, i, t) and (u, i', t') deriving the same key. Recall that each stage derives an ephemeral key, and each stage's own ephemeral key is included in its local key derivation. For the derived keys to be equal, therefore, the ephemeral keys generated by both agents would have to be equal as well, which would require a DH collision. This happens only with negligible probability (formally, we make a game hop to a game which aborts if there is a DH collision, and bound the difference between the games; this argument appears in the proof sketch below), and hence this case is impossible unless $i = i'$.

Fourth and finally, suppose that it wins because there exist $n + 1$ stages $(v, \cdot, \cdot)$ deriving the same key as (u, i, t) while there are only n copies of v in (u, i, t).IDs. Since there are only n copies of v in (u, i, t).IDs, either

(i) one of the $n + 1$ must have v.idx not equal to an index of v in (u, i, t).IDs, or

(ii) two of the stages $(v, \cdot, \cdot)$ must "collide", having the same v.idx.

In the first case, the disagreement implies that (u, i, t).IDs $\neq (v, j, s)$.IDs and hence that the derived keys are distinct, which is a contradiction. In the second case we again use uniqueness of ephemeral keys: the two colliding stages must have derived distinct ephemeral keys, at most one of which is the key appearing (u, i, t).IDs, and hence the stages must derive distinct keys.

We have ruled out all cases, and thus are done. □

Lemma 3. *Let $\mathsf{n_P}$, $\mathsf{n_S}$ and $\mathsf{n_s}$ denote bounds on the number of parties, sessions and stages in the security experiment respectively. Under the decisional DH assumption, where ι is instantiated as a random oracle, the success probability of any ppt adversary against the security experiment of our protocol is bounded above by*

$$\frac{1}{2} + \frac{\binom{\mathsf{n_P n_S n_s}}{2}}{q} + \gamma(\mathsf{n_P n_S n_s}^2)^\gamma \left(\epsilon_{DDH} + {}^1/_q\right) + \mathrm{negl}[\lambda]$$

where ϵ_{DDH} bounds the advantage of a ppt adversary against the decisional DH game.

Proof structure. Our proof uses the standard game hopping technique: we start at our original security game and consider ("hop to") similar games, bounding the success probability of the adversary in each hop, until we reach a game that the adversary clearly cannot win with a probability non-negligibly over ${}^1/_2$. As all the games' probabilities are related to one another, we are able to bound the original success probability of the adversary.

We make one modification to the protocol for technical reasons: as specified, ART has agents authenticate group creation messages with a signature under the identity key, and update messages with a MAC under the stage key. Because these keys are also used in the key exchange protocol, we cannot achieve key indistinguishability notions of security. In the computational proof, we will capture this authentication through the freshness condition instead of by a direct reduction to the security of the authenticator.

The overall structure of the proof is as follows. First, we perform some administrative game hops to avoid DH key collisions. Then, we guess the indices (u, i, t) of the sid of the Test session and stage. If it is not fresh then the adversary loses. If it is fresh, we perform a case distinction based on the condition of the freshness predicate which it satisfies: either the current copath is fresh or a previous stage was fresh.

In the latter case, indistinguishability will hold by induction. In the former case, by definition we know that all of the leaf keys used to generate the current stage are honestly-generated and unrevealed. The secret key at a node with child public keys g^x and g^y is defined to be g^{xy}, and thus by hardness of the DDH problem we will indistinguishably replace it with a random group element. We perform this replacement in turn for each non-leaf node in the tree, bounding the probability difference at each game hop with the DDH advantage. After all non-leaves have been replaced, the tree key (and hence the stage key) is replaced with a random group element. The success probability of the adversary against this final game is therefore no better than ${}^1/_2$. By summing probabilities throughout the various cases we derive our overall probability bound. □

Proof sketch. Security in this sense means that no efficient adversary can break the key indistinguishability game against ART. Suppose for contradiction that there exists an adversary $\mathcal{A}$ which can do so. By the definition of the security experiment, $\mathcal{A}$ wins only if with non-negligible probability

(i) it issues a $\mathsf{Test}(u, i, t)$ query against some stage t of a session i at agent u such that the session with sid (u, i, t) is fresh, and then

(ii) issues a correct $\mathsf{Guess}(b)$ query.

By Definition 17, the session with sid (u, i, t) is fresh precisely when all of the following hold:

(i) it has status `accept`

(ii) the adversary has not issued a $\mathsf{RevSessKey}(u, i, t)$ query

(iii) there does not exist (v, j, s) such that
   (a) the adversary has issued a query $\mathsf{RevSessKey}(v, j, s)$, and
   (b) the return value of that query is *sessk*

(iv) one of the following criteria holds:
   (a) $t > 0$ and session $(u, i, t - 1)$ is fresh, or
   (b) the current copath is fresh.

Our proof is a case distinction based on adversarial behaviour; for each case, we will construct a sequence of related games as per the game hopping proof technique [157]. Let Game 0 denote the game from the original security experiment, and let Adv_i denote the maximum over all adversaries $\mathcal{A}$ of the advantage of $\mathcal{A}$ in game i. Our goal is to bound Adv_0, the advantage of any adversary against the security experiment.

Recall that due to technical limitations of key indistinguishability models we are unable to faithfully model the explicit MACs which ART uses in group creation and key update messages. Instead, for the remainder of the proof we omit them from the protocol, and specify authentication "by fiat" through our freshness predicate—that is, we rule out attacks in which the authentication of these messages is violated. Specifically, a copath is only fresh if it comes from a tree whose leaves were all honestly generated, so that if the adversary inserts their own ephemeral key then any resulting copath will not be fresh.

At any point in a run of the game, recall that by construction such a tuple (u, i, t) uniquely identifies a corresponding pair of states π and σ if they exist. To simplify our notation, therefore, where is it more convenient we refer to session and bookkeeping states directly by their identifiers: by $(u, i, t).\pi.x$ we mean $\pi.x$ where π is the state of the unique session with sid (u, i, t) and by $(u, i, t).\sigma.y$ we mean $\sigma.y$ where σ is the bookkeeping state of the unique session with sid (u, i, t).

Game 0. This is the original AKE security game. We see that the success probability of the adversary is bounded above by

$$^1\!/_2 + \mathsf{Adv}_0$$

Game 1. This is the same as Game 0, except the challenger aborts and the adversary immediately loses if there is ever a collision of honestly generated DH keys in the game. That is, the challenger maintains a list of every DH value it generates, and aborts if any value appears more than once in that list.

There are a total number of $\mathsf{n_P}$ parties in the game, with a maximum of $\mathsf{n_S} \times \mathsf{n_s}$ ephemeral DH keys generated per party. There are thus at most $\mathsf{n_P} \times \mathsf{n_S} \times \mathsf{n_s}$ DH keys, each pair of which must not collide. All keys are generated in the same DH group of order q, so each of the $\binom{\mathsf{n_P n_S n_s}}{2}$ pairs has probability $1/q$ of colliding. Therefore, we have the following bound:

$$\mathsf{Adv}_0 \leq \frac{\binom{\mathsf{n_P n_S n_s}}{2}}{q} + \mathsf{Adv}_1$$

Game 2. This is the same as Game 1, except the challenger begins by guessing (uniformly at random, independently of other random samples) a user u', session i' and stage t'. If the adversary issues a $\mathsf{Test}(u, i, t)$ query with $(u, i, t) \neq (u', i', t')$, the challenger immediately aborts the game and the adversary loses.

Additionally, the challenger guesses a key counter value ℓ' and aborts if $\ell' \neq (u, i, t).\sigma.\ell[(u, i, t).\pi.\mathrm{idx}]$. In other words, the challenger also attempts to guess the number of DH keys sent by the Tested session before its Tested stage. At most $\mathsf{n_s}$ keys could have been sent, since $\mathsf{n_s}$ bounds the number of stages in a session.

Since the challenger's guess is independent of the adversary's choice of Test session, we derive the bound

$$\mathsf{Adv}_1 \leq \mathsf{n_P n_S n_s}^2\ \mathsf{Adv}_2$$

We remark that this "guessing" game leads to a loose probability bound: for practical values of $\mathsf{n_P}$ and $\mathsf{n_S}$, the constant factor in this game hop is very large. Bader et al. [6] give an impossibility result which we conjecture could be applied to ART to show that this looseness is necessary.

Game 3. This is another "guessing" game; this time, the challenger will guess the agent, session, stage and key counter for each peer to the Test session. If the guess is incorrect, or if there is more than one valid guess, then it aborts the game and the adversary loses. Note that if no such session exists for a particular peer then we consider the guess to be correct and do not abort the game.

More precisely, for each leaf index l in the tree $(u, i, t).\pi.T$ of the Test session, the challenger guesses a triple of indices $\omega' = (v'_l, j'_l, s'_l) \in [n_{\mathsf{n_P}}] \times [\mathsf{n_S}] \times [\mathsf{n_s}]$. It aborts the game if there ever exists a session $\omega = (v, j, s)$ with

(i) $(v, j, s).\pi.\mathrm{idx} = l$ (ω believes that it is at index l in the tree)

(ii) $(v, j, s).\pi.T = (u, i, t).\pi.T$ (ω has the same tree as (u, i, t))

(iii) $(v, j, s).\sigma.t > 0$ (ω has performed at least one key update, and)

(iv) $(v'_l, j'_l, s'_l) \neq (v, j, s)$ (ω was *not* the guessed session)

In other words, for each leaf index l in the tree of the Test session, the challenger guesses a peer, session, stage and received key counter corresponding to the Test session's state, in the sense that the peer's session state agrees with that of the Test session. These might not exist, but this game hop ensures that if they do exist, they are uniquely defined and known in advance by the challenger.

Uniqueness of the guessed tuples follows from Game 1, in which we ensured that honestly generated DH values are unique. Indeed, if the guessed tuple ω' were not unique then there would be two sessions with the same actor identity having the same leaf key. But we only abort the game if $\omega.\sigma.t > 0$, meaning that the leaf key was generated by ω, and we know from Game 1 that all generated DH keys are unique.

Recall that γ denotes the maximum group size. From $(u, i, t).\pi.T$ we can derive an ordered list of the peers associated with each leaf at stage t, and there are at most $\gamma - 1$ such leaves since γ bounds the size of a group. For each of these group members we guessed a peer (of which there at most $\mathsf{n_P}$), a session (of which there are at most $\mathsf{n_S}$), and a stage and received key counter (of which there are at most $\mathsf{n_s}$). Since guesses are uniformly random and independent of other random events, it follows that

$$\mathsf{Adv}_2 \leq (\mathsf{n_P}\mathsf{n_S}{\mathsf{n_s}}^2)^{\gamma-1}\ \mathsf{Adv}_3$$

Case distinction. At this point in the proof, we will make a case distinction based on adversary behaviour. Define the event E to be true if and only if the copath of u at $(u, i, t).\pi.T$ is fresh. We now perform a case distinction on E, considering first the case (i) where E is true, and then the case (ii) where E is false. Thus, our game hopping sequence splits: we either proceed from case (i) game 4, 5, 6..., or case (ii) game 4, 5, 6...

Case (i), Game 4**.** This is a game hop based on indistinguishability [157], in which we will replace the DH secret key at the parent of the first two leaf nodes in the Test session and stage with a random value. We will then show a probability bound by considering a hybrid game and giving a distinguisher $\mathcal{D}$ that interpolates between the two. The distinguisher $\mathcal{D}$ has input distributions P_1 (tuples (g, g^x, g^y, g^z) where x, y, z are random) and P_2 (tuples (g, g^x, g^y, g^{xy}) where x, y are random), and

(i) when given an element drawn from distribution P_1, outputs 1 with probability $\mathsf{Adv}_3 + {}^1\!/_2$

(ii) when given element drawn from distribution P_2, outputs 1 with probability $\mathsf{Adv}_{4(i).1} + {}^1\!/_2$

In this case we assume that E holds. By definition of copath freshness, it follows that the copath of u at $(u, i, t).\pi.T$ is the i^{th} copath induced by some Λ, where each $\lambda_j \in \Lambda$ was output by an honest stage against which no RevRand query was issued. Without loss of generality, let λ_0 be the leaf key of u in $(u, i, t).\pi.T$.

Recall that the parent of the first two leaf nodes λ_0 and λ_1 has private key $\iota(g^{\lambda_0\lambda_1})$. We will replace $g^{\lambda_0\lambda_1}$ with a random group element g^z in the Test session and all of the guessed ω' from Game 3. In this game, all computations depending on $\iota(g^{\lambda_0\lambda_1})$ use $\iota(g^z)$ instead.

In the new game 4(i), in the local stage key computation of the actor of the Test session and stage and in any matching sessions (which are unique by the previous game), $\iota(g^{\lambda_0\lambda_1})$ is replaced with $\iota(g^z)$ for uniformly randomly chosen DH group exponent z, and all subsequent computations upwards along the path of the tree use $\iota(g^z)$ instead of $\iota(g^{\lambda_0\lambda_1})$.

We now prove that game 4(i) is indistinguishable from game 3 under the PRF-

ODH assumption. More precisely, we will show that if there exists a distinguisher $\mathcal{D}$ which can efficiently distinguish between games 3 and 4(i), then the PRF-ODH assumption is false. This implies that $\mathsf{Adv}_4 \leq \mathsf{Adv}_3 + \max_{\mathcal{D}} \epsilon_{\mathcal{D}}$, where $\epsilon_{\mathcal{D}}$ is the probability that a pptm $\mathcal{D}$ correctly distinguishes between Games 3 and 4(i).1.

It remains to bound $\epsilon_{\mathcal{D}}$, which we do with an explicit reduction to PRF-ODH. Specifically, suppose $\mathcal{D}$ is such a distinguisher. We construct an adversary $\mathcal{A}(\mathcal{D})$ against the PRF-ODH game. Upon receiving g^x, g^y, g^z from the PRF-ODH challenger, $\mathcal{A}(\mathcal{D})$ faithfully simulates the hybrid game as its challenger, except using g^x and g^y as the first two leaf nodes and $\iota(g^z)$ as the secret key of their parent in the Test session and all of the guessed ω' from Game 3.

Since $\mathcal{A}(\mathcal{D})$ knows g^z, it can use it to compute all intermediate keys in the tree which depend on the first two leaf nodes, including the tree key at the root.

Our constructed PRF-ODH adversary is given $\iota(g^z)$, which by construction is the node key at the parent of Alex's and Blake's leaf nodes. It can therefore replace this node key with $\iota(g^z)$ and, using this secret, compute all public DH intermediate keys up the tree that depend on $\iota(g^z)$, including the tree key at the top of the tree. This game is a hybrid game between Game 3 and Game 4, with equal probability of either. The simulator answers all queries in the honest way, except in the send/create queries where it needs to insert these DH values.

Since this is case (i), the leaf keys are honestly sent. From game 3 the challenger knows which agent's session and stage's they are generated at in advance, as well as which generated DH this will be. In other words, the challenger knows (v, j, t) and the counter $(v, j, t).\sigma.t$ of how many DH keys have been generated. If

(i) $t = 0$ and the adversary issues a query $\mathsf{Create}(v, \dots)$, or

(ii) $t > 0$ and the adversary issues a query $\mathsf{ASend}(v, j, t)$

then the challenger performs a replacement instead of honestly simulating the protocol algorithm. Specifically, it inserts the PRF-ODH challenge values into the state of session (v, j, t) replacing the first two leaf nodes and their parent, and then continues with the simulation as normal to derive the resulting leaf key and output values.

Because of the earlier game hops the simulator knows where to inject the replaced values in the simulation, and because of the freshness predicate they are honest. Similarly, because of the freshness predicate it never has to answer a $\mathsf{RevRand}$ query against either of these two values, and it can honestly simulate any other reveal queries. Therefore the simulation is sound.

In Game 1 we ensured no DH keys collide, and with probability $1/q$ the PRF-ODH challenger may provide challenge values $g^x = g^y$, in which case the simulator must abort. Fortunately this happens with negligible probability. Thus, we have the bound:

$$\mathsf{Adv}_3 \leq \mathsf{Adv}_4 + \epsilon_{\mathsf{PRF\text{-}ODH}} + 1/q$$

We will now iteratively repeat this game hop for all other fresh DH values in the tree $(u, i, t).\pi.T$. Because we are in case (i) and know from the previous game hops where to insert the PRF-ODH challenge values, we will therefore conclude that each node key in turn is indistinguishable from random. Repeating this process, the eventual conclusion will be that the secret at the root of the tree is also indistinguishable from random.

Case (i), Game $4 + k$ where $1 \leq k \leq \gamma$. We iteratively perform the replacement in the initial game for all the leaves in the tree. As before, we know in which sessions to perform the replacement because of the guesses in the earlier game. In each case, distinguishing between the games before and after the hop leads to an explicit adversary against PRF-ODH.

Given a group size of n, we never need to do more than $n \leq \gamma$ such game hops due to our tree structure. Thus

$$\mathsf{Adv}_{\mathsf{np}} \leq \gamma\,(\epsilon_{\mathsf{PRF\text{-}ODH}} + {}^{1}\!/_{q}) + 0$$

At this point, the tree key of the Tested session is no longer a function of the leaf keys—instead, it depends on the keys at the nodes whose children are leaves, each of which has been replaced by a random value, unknown to the adversary. The adversary can therefore not do any better than random against the final game.

Case (ii), Game 4. We now proceed with case (ii), restarting our game hopping sequence from Game 3. Assume now that E does not hold, and thus the copath in the session state of the Tested stage is not fresh. Since the Tested stage must be fresh, the first disjunct of the final clause of the freshness predicate must hold: that $t > 0$ and the stage with sid $(u, i, t - 1)$ is fresh.

We proceed by induction on the stage number of the Test session. Our inductive hypothesis at step k is that no adversary can win with non-negligible advantage if the tested session has stage number less than or equal to k. The base case $k = 0$ holds by the above argument: case (ii) cannot apply since the freshness predicate in case $k = 0$ requires E to occur.

Assume now that the inductive hypothesis is true for stage $t \leq k - 1$; we show that it is also true for $t = k$. As before, if the adversary queries $\mathsf{Test}(u, i, t)$, then this means stage t must be fresh. Let RO be the event that the adversary queried the random oracle and received the key of the Test stage as a reply.

If RO does not hold, then since since the adversary is not allowed to reveal the key because of the freshness predicate, the only option is for a key replication attack. We can perform a single game hop in which we replace the stage key with a random value. Since the random oracle response is by construction a random value, this replacement is indistinguishable and the resulting advantage for the adversary is zero.

Thus, we conclude that RO must hold. Since random oracle produce collisions with only negligible probability, it must be the case that the adversary queried the KDF on the same input that u did on the stage key computation in the stage with sid (u, i, k). In particular, it must have queried the random oracle on the stage key as that is one of the inputs. This adversary therefore has a distinguishing advantage against the previous stage, (noting that this is case (ii) so it is fresh by definition). This contradicts our induction hypothesis.

Specifically, given such an adversary $\mathcal{A}$ we can construct an adversary $\mathcal{A}'$ which wins with non-negligible probability against stage $k-1$. $\mathcal{A}'$ simply simulates $\mathcal{A}$ without changing any values and recording all random oracle queries; the simulation is thus trivially faithful. When $\mathcal{A}$ issues a $\mathsf{Test}(u, i, k)$ query, $\mathcal{A}'$ issues a $\mathsf{Test}(u, i, k-1)$ query and compares the resulting key to all of $\mathcal{A}$'s random oracle queries. If it appears in a random oracle query, $\mathcal{A}'$ outputs $b = 0$; otherwise, it outputs $b = 1$. By construction, the stage with sid $(u, i, k-1)$ is fresh and its stage key is an argument to the random oracle, so the advantage of $\mathcal{A}'$ is non-negligible.

This contradicts our inductive hypothesis that no adversary can win against a stage less than k with non-negligible probability; the result thus holds in case (ii) by induction. □

We conclude:

Theorem 4. *ART is a secure key exchange protocol in our model.*

Proof. Lemma 2 bounds the advantage against the partnering game, and Lemma 3 bounds the advantage against the indistinguishability game. By definition, a protocol admitting these two bounds is secure in our model. □

5.6 Extensions

We here remark on various possible extensions to our ART design. In general, because we use standard tree-DH techniques, much of the existing literature is directly applicable. This means that we can directly apply well-studied techniques which do not require interactive communication rounds.

These designs are not yet formally analysed, but many are important for practical use cases. In particular, dynamic groups are already part of the MLS draft specification, since the membership list of most groups in use is not necessarily known at creation time.

5.6.1 Chain keys

The Signal protocol introduced the concept of *chain keys* to support out-of-order message receipt as well as a fine-grained form of forward secrecy. Instead of using a shared secret to encrypt messages directly, Signal derives a new encryption key for each message from a hash chain.

The shared secret derived by ART can be directly used in the same way, and we give an example key schedule in Figure 5.6. Instead of sending messages directly encrypted under the group key, we envision that implementations would derive separate sending chains for each group member, and advance a member's sending chain for every message sent by that member. This would convey the same benefits

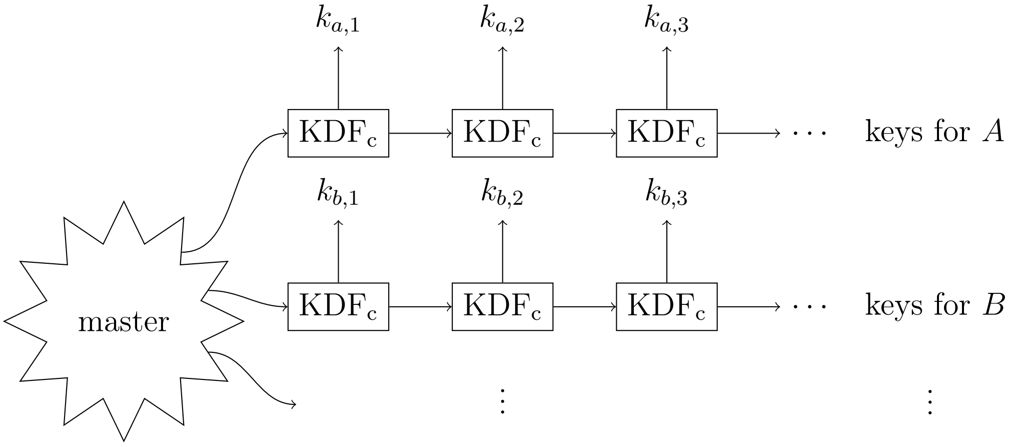


Figure 5.6: Example key schedule for deriving from a master secret multiple chain keys $k_{u,i}$ for each user u. Each group member derives their own chain of sending keys from the master secret, perhaps by using their identity as an input to the KDF. Sending chains then work as for Signal; this merely generalises its two-chain design.

as it does in Signal (simple out-of-order message handling and a new key for each encryption) at a storage cost linear in the number of chains.

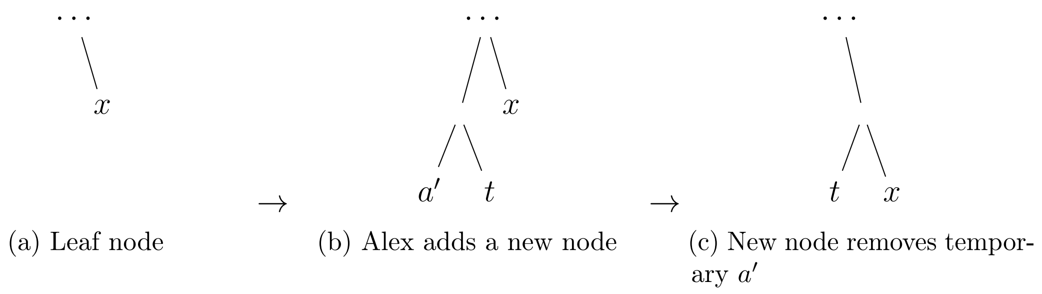
We do not envision that a security analysis for this design would be challenging. A simple goal would be to show that it is no less secure than just using the master secret directly, and with an inductive cleanness predicate like that of Definition 9 on page 60 one could capture the precise guarantees of each key with respect to a state-corrupting adversary model.

5.6.2 Sender-specific authentication

As early as 1999, Wallner, Harder and Agee [165] pointed out the issue of "sender-specific authentication": in a system which derives a shared group key known to all members, there is no cryptographic proof of *which* group member sent a particular message. That is, if Alex, Blake and Charlie share a group key k, and Charlie receives a message $\{m\}_k$ encrypted under k, it is only correct for Charlie to conclude that "either Alex or Blake sent m" and not "Alex sent m" or "Blake sent m".

Various works have discussed ways to provide cryptographic proof of the sender. The most common design is to assign to each group member a signature key with which they sign all their messages, which is the approach taken by the Sender Key variant of Signal. There, each agent broadcasts not just a symmetric encryption key but also a public signature key over pairwise channels, and then signs all their messages to be verified with the signature key. It would not be hard to incorporate a similar design into ART, and we conjecture that its security analysis would not be too challenging as long as the properties of the pairwise channels are assumed to be sufficiently strong.

However, one problem with using a permanent signature key is that it will not provide PCS. Indeed, an adversary who has ever compromised e.g., Alex would forever be able to impersonate Alex as long as they are able to send messages to



(a) Leaf node

(b) Alex adds a new node

(c) New node removes temporary a'

Figure 5.7: Alex (a) can add a new node t to any leaf x in the tree. To do so, Alex generates a temporary DH key a', sends $g^{a'}$ to t and $g^{g^{a't}}$ to x, informs x of the new tree shape, and deletes a'. When t sends their first message, they can remove the intermediate node a' and move themselves up to become a sibling of x.

the group at all. This is because they would know Alex's private signature key, and hence be able to forge signatures from Alex.

There are more complex designs which may not have this drawback. Indeed, each agent in ART is already allocated a private key—their leaf key—whose public key is known to every group member. We could therefore imagine a design in which each agent signs messages under their leaf key. We conjecture that these signatures would provide authentication of the message sender even in a post-compromise scenario, since the leaf keys used for signing are continuously updated.

The security analysis of signatures under leaf keys would be more challenging, because this design violates a principle of separation of purpose for cryptographic keys: leaf keys would be used both for DH operations and for signatures. This violation is not uncommon, and indeed we have already seen the need to work around Signal's use of identity keys for both DH and signing.

A cleaner design would therefore likely be to derive separate leaf and signature keys from the same master secret, rotating both on every update. We leave this optimistically to future work.

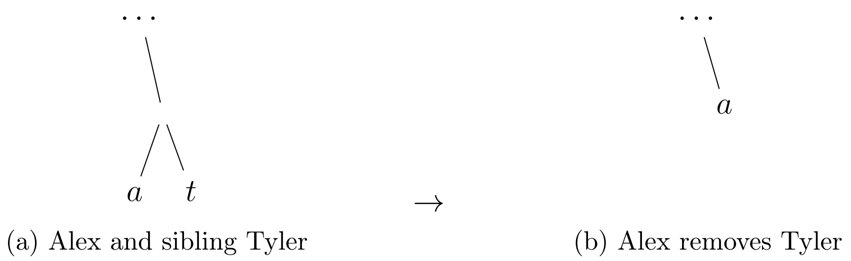
5.6.3 Dynamic groups

In practical use, most groups are not static: new users are frequently added to or removed from existing groups. There has been a significant amount of research into supporting these operations in group messaging protocols, and we refer the reader to e.g. Kim, Perrig and Tsudik [101] for a summary of some previous work.

Many existing designs work with tree-based group messaging protocols in general, and thus can be applied to our asynchronous designs. However, as with group creation, we may need to modify existing protocols for join or leave operations in order to make them asynchronous.

Additions For example, suppose Alex (with leaf key a) wishes to add Tyler (with leaf key t) to a group. Using a system similar to the setup keys in §5.4.1, Alex can in one step

- pick any node x in the tree,



(a) Alex and sibling Tyler

(b) Alex removes Tyler

Figure 5.8: Alex (a) can remove their sibling Tyler (t) from a tree. To do so, Alex broadcasts a new copath to the group in which Tyler is not present, removing the leaf node with key t and moving up one level in the tree.

- generate a temporary DH key a',
- add a new node with children a' and t as a sibling to x,
- inform x of the changes, and
- broadcast the updated copath to all other group members.

On a subsequent update, either Tyler or the holder of x can remove the temporary node a', resulting in a tree now with Tyler as a member.

Depending on the location where Alex adds Tyler, this form of addition may affect performance by unbalancing the tree. There are interesting avenues of future work to avoid this, either by supporting asynchronous rebalancing operations or by cooperation between group members to add group members into optimal locations.

Removals It is also important to support removals from the group, and there are various potential designs to do so. A basic observation is that any node can easily remove their *sibling* from the group by sending an update moving themselves one level up in the tree, as depicted in Figure 5.8. This technique generalises to removing any cousin i.e., node with the same grandparent. However, it does not apply to nodes more distant in the tree, since nodes do not have enough information to compute the new copaths for broadcast unless the removal is of a cousin.

Removing arbitrary nodes from the tree is more challenging, and we believe a valuable problem for future work. One technique is for Alex to generate a fresh random value which we might call a "deletion secret", and broadcast that (encrypted) value to all group members except Tyler. Mixing this value into the group key will result in a shared secret known to all members except Tyler, and thus in a sense Tyler is removed from the group.

We remark that, by asymmetrically encrypting to the public keys of the nodes on Tyler's copath, it is possible to perform this broadcast at only logarithmic cost and without assuming separate pairwise channels. This technique is used in other contexts as a form of hierarchical KEM.

The downside of this strategy is that while the resulting key is indeed secret from Tyler, the leaf node t is still contained in the tree, and thus Tyler has the ability to compute all keys at the root of the tree. On its own this is insufficient to derive message encryption keys due to ART's key chaining, but if Tyler colludes with a compromising adversary we can see that the security guarantees are weakened.

5.6.4 Multiple Devices

One important motivation for supporting group messaging is to enable users to communicate using more than one of their own devices. By treating each device as a separate group member, ART of course supports this use case. However, the tree structure can be optimised for this particular scenario: all of Alex's devices can be stored in a single subtree, so that the "leaves" of the group tree are themselves roots of device-specific trees. This has two particular benefits.

First, at any point before creating or joining a group Alex can execute the group key agreement protocol just between the devices they control, and use the resulting shared secret as an ephemeral prekey or "pretree". This allows other group members to retrieve a single key for Alex, instead of adding all devices as separate entities in the group tree. If most users have multiple devices, this can significantly reduce the size of group trees.

Second, when any of Alex's devices performs a key update, the other group members only need to know the public keys from the root of Alex's subtree to the root of the group tree. In particular, Alex does not need to broadcast to the group their set of devices or the metadata about which device is performing a key update. Thus, Alex can enjoy end-to-end encryption with improved privacy properties.

We see improved support for multiple devices, and a fuller understanding of the authentication properties when a single user owns multiple private keys, as a valuable avenue for future work.

5.7 Conclusion

While modern messaging applications such as Signal offer strong security guarantees, they typically allow only two parties to communicate. Since large-scale deployed messaging systems allow for multiple users each with multiple devices, two-party protocols are not directly applicable. This has led to a number of different designs for asynchronous end-to-end encrypted group messaging, each providing different security and efficiency properties. In particular, in the most common design, the effective security guarantees of group messaging are strictly weaker than those of two-party messaging, despite there being no user-visible difference.

In this chapter, we showed that it is possible to build an efficient group messaging system which is still asynchronous and strongly secure. To do so, we combined techniques from synchronous group messaging with strong modern security guarantees from asynchronous messaging. Our resulting ART design combines the bandwidth benefits of group messaging with the strong security guarantees of modern point-to-point protocols. This paves the way for modern messaging applications to offer the same type of security for groups that they are currently only offering for two-party communications.

Our construction is of independent interest, since it provides a blueprint for generically applying insights from synchronous group messaging in the asynchronous setting. We expect this to lead to many more alternative designs in future works.

5.7.1 Related Work

We briefly mention some related designs.

OTR-style OTR [37] is a classic messaging protocol which has achieved moderate uptake as a plugin for instant messaging applications. Goldberg et al. [86] generalise it to mpOTR, aiming for security as well as deniability. mpOTR has since given rise to a number of interactive protocols such as $(N+1)$sec [77].

OTR-style protocols are generally comprised of similar phases:

- first, parties conduct a number of interactive rounds of communication in order to derive a group key
- second, parties send and receive messages, perhaps performing additional cryptographic operations; and
- finally, there may be a closing phase (for instance, to assess transcript consistency between all participants).

As far as we know, all OTR-style protocols require interactive rounds of communication, and thus are not suitable for our context of asynchronous messaging.

Burmester-Desmedt Instead of using a tree of DH keys, Burmester and Desmedt [45] propose a ring structure: each agent generates a secret key x_i and broadcasts $k_i = g^{x_i}$, and then after receiving their neighbours' keys k_{i-1} and k_{i+1} also broadcasts $K_i = (k_{i+1}/k_{i-1})^{x_i}$. This allows the i^{th} agent to compute a shared session key as

$$\begin{aligned} K &= k_{i-1}^{nx_i} K_i^{n-1} K_{i+1}^{n-2} \cdots K_{i-2} \\ &= \ldots \\ &= g^{x_1x_2 + x_2x_3 + \cdots + x_nx_1} \end{aligned}$$

One particularly good property of Burmester-Desmedt group key exchange is its scaling property: since each agent only computes with their neighbours' values, the number of exponentiations is constant in the group size. However, a downside in our context is that the three phases depend serially on their predecessors. This makes it challenging to give an asynchronous version of the protocol: while we could use prekeys to perform the initial round of communication asynchronously, it is less clear how to avoid the second round, since each agent must perform a computation before the final key can be computed. An interesting research question is whether it is possible to use some of the techniques described earlier to avoid this round and build an asynchronous variant of Burmester-Desmedt.

A number of works, including [34, 76], have built on this design.

Assuming an authentic network Cachin and Strobl [46] discuss "asynchronous" group key exchange in the setting of distributed systems, in the sense that they do not rely on a centralised clock but allow parties to execute independently and without synchronisation. Their approach allows certain parties to crash without preventing the protocol from terminating. However, they require several interactive rounds of communication, and do not provide PCS. Steiner, Tsudik and Waidner [159] also work in this framework.

Physical approaches Some designs use physical constraints to restrict malicious group members. For example, HoPoKey [135] has its participants arrange themselves into a circle, with neighbours interacting. This allows it to derive strong security properties. Our goal, however, is to allow users to communicate without requiring physical co-location.

SafeSlinger SafeSlinger [79] is a secure messaging app whose goal is usable, "privacy-preserving and secure group credential exchange". It aims for message secrecy under an adversary model that allows for malicious group members. It shares many of our goals, and uses a DH tree of keys, but does not aim for PCS or asynchronicity. It is nevertheless closely related to our work.

We see two core differences. First, ART is explicitly designed to achieve PCS of message keys, while SafeSlinger instead aims for (non-forward) secrecy. Of necessity [57], ART must therefore support stateful and iterated key derivations, while SafeSlinger derives a single group key which is not updated. Using the unbalanced DH key tree of SafeSlinger, while reducing the computational load on the initiator, would cause ART's key updates to take linear (versus logarithmic) time.

Second, SafeSlinger is a synchronous protocol in which all group members must be online concurrently. This allows them to have commitment, verification and secret-sharing rounds, after which the trust requirements upon all participants are the same. ART, on the other hand, is an asynchronous protocol which allows group creation and message sending even with members who are not currently online; in the initial phase during which not all members have participated in the group, ART thus places additional trust requirements on the creator.

CHAPTER 6
LOOKING FORWARD: BETTER SYMBOLIC MODELS

This chapter is based on the discussion from

> *Katriel Cohn-Gordon and Cas Cremers.* Mind the Gap: Where Provable Security and Real-World Messaging Don't Quite Meet. *Cryptology ePrint Archive, Report 2017/982.* `http://eprint.iacr.org/2017/982`. *2017*

and on unpublished work with Cas Cremers, Dennis Jackson and Ralf Sasse on symbolic modelling of broken primitives. The core contributions are all my own.

We aim to write a paper on this topic in due course. This chapter contains some core ideas and a number of case studies to illustrate the approach; the full paper will also contain a more systematic theoretical background.

Modern messaging systems are becoming too complex to analyse systematically without tool support. Indeed, while our Signal proof covered much of what we considered the cryptographic core, for ART we had to leave out authentication properties entirely! There are plenty of other protocol components that are excluded from many computational models but which we believe could be captured symbolically.

Mechanised symbolic protocol verifiers such as TAMARIN are able to tackle the increased complexity of these systems. This allows us to model more components of a protocol, and thus provide guarantees about a larger class of attacks. The major drawback of symbolic models is that their abstraction of cryptographic primitives is coarser than the computational one, and thus that there are significant attacks which can be found in the latter context but not in the former.

In this chapter we make two main points. First, in §6.1 we recall that computational models have their own limitations, not just in scope but also in the guarantees they provide. Second, in §6.2 we give an extended series of case studies showing that symbolic models can find many attacks previously thought lost in the abstraction. We'll conclude that for large, complex protocols, symbolic models may be the way forward.

6.1 Limitations of Computational Models

We begin by clarifying some of the technical limitations of computational models.

6.1.1 Complexity

Perhaps the greatest downside of the computational approach to protocol security is its enormous complexity.

The complexity stems from a number of sources. First, the actual definitions are built on top of Turing machines and probability theory, so that even the simplest games assume a way for two pptms to interact with each other and send messages back and forth. Kuesters and Tuengerthal [110] give more formal definitions of these interactions, including assigning names to the different tapes used for communication and defining a scheduling mechanism for machines to hand over execution to each other. Their definitions run to two dozen pages before beginning to state any theorems. In contrast, most authors (of necessity) sweep these definitions under the rug and use instead an intuitive, unformalised notion of computation and interaction.

Second, the systems under consideration have steadily grown in size: early computational proofs dealt with encryption or signature primitives, while ours describe protocol algorithms which have many varied branches and possible outcomes. Each of these cases requires its own case in the reduction and its own simulator, and while many of these are similar at a high level they differ in crucial details. It is not feasible to work out every single case in full detail, so as with many other reductions we instead work out a single case and then leave similar cases "by analogy". Of course, this is valid only if the similar cases really are analogous.

Third, there are many small technical details which are usually glossed over in proofs but which technically can invalidate the correctness of reductions. These include for example differences in distributions of DH values across games. (A concrete example from the reduction for Signal: we began the game-hopping sequence by aborting if two honestly generated DH values ever collide. Later, we replaced some DH values with group elements received from a PRF-ODH challenger. In an earlier draft of the reduction we omitted to bound the probability that the latter values collided with an honestly generated one. This meant that we did not exclude the possibility of an adversary which successfully attacked Signal but which failed when presented with challenge values.)

6.1.2 Tightness

"Proofs" in computational models are more accurately described as reductions. A cryptographic reduction, given an adversary $\mathcal{A}$ winning a game $\mathcal{P}$ with probability p, produces an adversary $\mathcal{B} = \mathcal{B}(\mathcal{A})$ winning another game $\mathcal{Q}$ with probability $q = q(p)$. If such a reduction exists we can conclude that $\mathcal{P}$ is at least as hard to win as $\mathcal{Q}$.

For example, Bellare and Rogaway [25] define a signature scheme called Full Domain Hash (FDH), and reduce the problem of forging FDH signatures ($\mathcal{P}$ in our notation above) to the problem[1] $\mathcal{Q}$ of computing $f^{-1}(y) = y^d \pmod N$ for some fixed d and integers $y \pmod N$. Their reduction produces an adversary which wins $\mathcal{Q}$ with probability at least $q = p/n_{\text{queries}}$. Equivalently, given a bound q on the probability of winning $\mathcal{Q}$, their reduction bounds the probability of winning $\mathcal{P}$ by $q \times (n_{\text{signatures}} + n_{\text{hashes}})$, where $n_{\text{signatures}}$ and n_{hashes} count the number of signatures and hash queries performed by the $\mathcal{P}$-adversary.

(We remark in passing that cryptographic reductions are more-or-less the same as the reductions that might be covered in an undergraduate class on complexity theory, but with some interesting differences. In particular, in cryptography we want to show that *most* instances of the problem $\mathcal{P}$ are hard, not just that there definitely exists *some* hard instance[2]. Cryptographic reductions also commonly fail with high but non-overwhelming probability, while most complexity-theoretic reductions are deterministic.)

The existence of a reduction from $\mathcal{P}$ to $\mathcal{Q}$ is not necessarily enough to convince us that $\mathcal{P}$ corresponds to a system which is secure in a practical sense. One failure mode is that there is some practical attack which does not correspond to a pptm adversary as input to the reduction. Another is that $q(p)$, while polynomial in p, turns out to be rather smaller than might be hoped.

Indeed, Bellare and Rogaway [25] explicitly compute their bound on the probability of producing a FDH signature forgery: if an adversary has access to 2^{30} signatures, can compute 2^{60} hashes and can solve $\mathcal{Q}$ with probability 2^{-61}, then they prove that it can forge a signature with probability at most $1/2$. (For exactly this reason, in the same paper the authors define PSS, for which they can prove a bound close to 2^{-61} in the above context. Coron [62] later gave a reduction with a factor only of $n_{\text{signatures}}$, and an impossibility result claiming that all reductions of FDH to $\mathcal{Q}$ must contain that factor. Kakvi and Kiltz [98] showed a gap in this impossibility result and gave a reduction from $\mathcal{P}$ to a harder problem $\mathcal{Q}'$ which does not contain the factor $n_{\text{signatures}}$.)

We say, waving our hands somewhat, that a reduction is **tight** if q and p are about equal, and loose otherwise. (Usually we also require that the running times of the two adversaries are approximately the same.) A non-tight reduction means that the concrete security parameter for the source of the reduction may need to be much larger than that required for the believed hardness of the target.

The actual utility of non-tight reductions has been discussed at length in the community. Goldreich [88] argues that they can show "plausibility" of security, and Damgård [67] that they are useful because they rule out polynomial-time attacks such as those that sidestep the hard problem entirely, while Chatterjee, Menezes and Sarkar [53] give attacks coming precisely from non-tight reductions.

[1] Hardness of this latter problem is known as the "trapdoor one-wayness" assumption for the RSA function. A function f is trapdoor one-way if there is some value d such that it is hard to invert f without knowing d, but easy to invert it if d is known.

[2] That is, we consider average-case not worst-case complexity. A worst-case-secure scheme would not be good enough: consider for example a signature scheme with a proof that there exists *some* key for which signatures are hard to forge.

Computational security proofs for *protocols* are particularly prone to looseness, and indeed Bader et al. [6] recently showed that a large class of protocol reductions cannot possibly be tight. The source of looseness in most reductions in the B-R framework is a "guessing" game hop, in which the constructed adversary from the reduction guesses a particular simulated role oracle to modify in the simulation.

More precisely, to show a reduction in the B-R framework, given an adversary $\mathcal{A}$ against a protocol we construct a new adversary $\mathcal{B}$ against its primitive, and relate the success probability of $\mathcal{B}$ to that of $\mathcal{A}$. $\mathcal{B}$ receives a challenge value from its own game, and then simulates the security experiment for $\mathcal{A}$, including simulating all of its role oracles and answering all of its queries. However, for one particular oracle, $\mathcal{B}$ modifies the oracle execution to include some part of its challenge value. Constructed correctly, if $\mathcal{A}$ targets that same oracle to attack and does so successfully, then $\mathcal{B}$ can deduce a correct response for its own challenge.

Reductions like this one are loose, because $\mathcal{A}$ might not choose to attack the right role oracle i.e., the one which depends on $\mathcal{B}$'s challenge values. We can show (by independence) that $\mathcal{A}$ will choose that oracle with probability no worse than 1 in the total number of oracles, but this type of failure leads to a factor of n_{oracles} in the resulting security bound. Seen one way, this is a technicality; seen another, it means that our proof of Signal does not in fact rule out attacks when Signal is used with standard security parameters.

A naïve idea might be to insert the challenge values in *every* oracle instead of simply guessing one. However, the simulated adversary is only guaranteed to work on an average instance of its problem, or equivalently if it cannot tell the difference between the simulation and the actual game. Naïvely inserting the same challenge into every oracle would likely be detectable by an adversary, in which case we cannot conclude that it would still win the simulated game. (For a simple counterexample, consider unauthenticated DH and an adversary which gives up if it sees an agent output the same DH value twice. This adversary might win against a real protocol which samples DH values randomly, but would always fail if every DH value was replaced with the same challenge.)

It is sometimes possible to work around this limitation by embedding only a part of the challenge value in every oracle. This way, no matter which oracle the adversary chooses to attack, its result can be used to break the challenge. However, doing so leads to another problem: in B-R models the adversary is permitted to reveal the keys of any session not under attack. If the hard problem is embedded in *every* oracle, then it is no longer quite so clear how the simulator should respond to RevSessKey queries.

Recently, Bader et al. [7] developed some new AKE protocols with tight reductions. They work around the guessing problem using two new techniques:

(i) a new "doubled" form of signature scheme, and

(ii) a new proof technique that allows embedding parts of a challenge problem into every oracle while still accurately simulating queries.

While this work raises many interesting questions and enjoys the great benefit of a tight reduction, it works only for certain protocols which they describe. In particular, there is currently no known technique for constructing a tight reduction that is applicable to e.g., the Signal protocol.

6.1.3 DH Assumptions

Looseness is one way in which a reduction can fail to be useful. Another is if the problem $\mathcal{Q}$ is itself not hard.

DH assumptions are some of the most common problems $\mathcal{Q}$ to which we reduce the security of modern AKE protocols. They encode the difficulty of solving certain discrete-logarithm-type problems in finite groups; we give formal definitions of DDH, Computational Diffie–Hellman (CDH), GDH and PRF-ODH [41] in Appendix A.1 on page 129 and plenty more exist.

The simplest of these assumptions is perhaps Computational Diffie–Hellman (CDH). To solve a CDH problem, an adversary which is given group elements $h_1 = g^{x_1}$ and $h_2 = g^{x_2}$ must output the value $g^{x_1 x_2}$. The naïve technique to compute this value is first to compute the discrete logarithm $x_1 = \log_g(h_1)$ and then to raise h_2 to the power x_1. However, discrete logarithms are believed to be hard to compute in most groups, and there is a widespread believe that in fact CDH is hard in cyclic groups $\mathbb{Z}_p$ and in the prime subgroups of some elliptic curves.

Assuming hardness of CDH (or even its weaker sibling DDH) is sometimes enough to prove a protocol secure. That is, for some protocols there exist reductions from their security experiment to the CDH game. However, in security games with modern queries (e.g. RevRand) and complex key derivations, or for protocols in ACCE-like models with key confirmation messages, it may be the case that no reduction is known to CDH. The reason that CDH is not enough for usual proof techniques is roughly that the normal approach to building a simulator in these cases would allow the adversary to detect whether it is being simulated, by injecting DH values and inspecting the resulting messages. In turn, this means that the constructed adversary does not necessarily succeed on an average-case instance of the security experiment, invalidating the reduction.

In more detail, consider first the case of TLS-DHE, which includes a `Finished` message derived from the session key. (This example is from [96, §8].) Consider an adversary $\mathcal{A}$ which corrupts some agent Blake and impersonates Blake to Alex, but replacing Blake's DH ephemeral key k_B with a fresh one $k_{\mathcal{A}}$ known only to $\mathcal{A}$. Since $\mathcal{A}$ knows $k_{\mathcal{A}}$ it can compute the value of the `Finished` message it expects to receive. If $\mathcal{A}$ receives an incorrect `Finished` message we define it to abort; otherwise, it replies with the correct response causing Alex to accept.

How might we simulate the security experiment to such an adversary in order to reduce to e.g. CDH? As usual, we could select an oracle and replace its DH ephemerals with random values. If $\mathcal{A}$ chooses that oracle as its target in the above execution, then the simulator must be able to construct a valid `Finished` message, since otherwise the simulation is not accurate. However, the `Finished` message depends on the session key, and computing that session key is precisely the CDH problem!

A similar problem occurs when constructing security reductions in eCK models for protocols with key derivations which depend on both long-term and ephemeral keys. Here, the problem is to simulate not a `Finished` message but a RevSessKey query. Again, an adversary can inject its own ephemeral key in a session and compute the expected key of that session, aborting if they differ. A simulator must therefore be able to simulate RevSessKey queries against all fresh sessions, but cannot do so for those using the injected challenge value.

If the simulator is not required to simulate such RevSessKey queries, then CDH or even DDH can suffice. This can happen when the adversary is forbidden from issuing them by the freshness predicate, when they can be accurately simulated by a random value, or when they do not depend on the challenge values. For example, Canetti and Krawczyk [50] reduce the security of an unauthenticated DH protocol to the DDH problem, because the challenge values are only injected into a single session and thus the simulator can correctly reply to RevSessKey queries against any other session.

6.1.3.1 Stronger DH Assumptions

It is still possible to give reductions for protocols in the cases above, but we must make a stronger assumption in order to do so. There are two main assumptions used in this case, GDH+ROM and PRF-ODH, but both have drawbacks.

GDH+ROM The Gap Diffie–Hellman (GDH) assumption states that it is hard to solve CDH i.e., to output g^{xy} given g^x and g^y, even when given access to an oracle for DDH. If the unsimulatable value as described above is derived by passing a DH shared secret through a KDF, and we assume that that KDF is instantiated as a random oracle, then under GDH there is a way to prevent the adversary from detecting the simulation.

To do so, we use the important fact that the adversary cannot "internally" query the random oracle: it must ask the simulator to perform the query for it, and the simulator has access to the query arguments at this point. Using the DDH oracle, the simulator can check whether the adversary has already solved the CDH challenge: if so, its work is done; if not, the random oracle assumption means that returning the "wrong" value will be undetectable.

Thus, suppose for example that we are analysing a protocol whose session keys are derived as $\mathrm{KDF}(g^{xy})$ for DH ephemeral values x and y, and we wish to construct a reduction to the GDH problem. As above, we define a simulator $\mathcal{S}$ which takes a protocol adversary $\mathcal{A}$ and constructs a GDH adversary. $\mathcal{S}$ simulates $\mathcal{A}$ in the protocol security experiment, and thus must be able to answer RevSessKey queries against some sessions, in particular sessions where one party's ephemeral is the GDH challenge value and the other party's ephemeral was generated by the adversary.

In a reduction to CDH, it is not clear how to simulate responses to such queries: they have session key $\mathrm{KDF}(g^{ub})$ where g^u is one of the GDH challenge values, and g^b is an ephemeral key generated by the adversary. $\mathcal{S}$ cannot compute this value directly because it knows neither u nor b, but $\mathcal{A}$ can compute it as $(g^u)^b$ and thus can detect whether $\mathcal{S}$ has given the correct response.

If we assume that KDF is instantiated by a random oracle, however, then it is possible to give an accurate simulation. To do so, instead of returning $\mathrm{KDF}(g^{ub})$ $\mathcal{S}$ returns a random value sampled from the output distribution of the random oracle. $\mathcal{S}$ additionally monitors all queries q which $\mathcal{A}$ makes to the random oracle. For each q, $\mathcal{S}$ itself queries the DDH oracle provided by the GDH game on (the relevant substring of) q to decide whether it is a correct solution to the CDH challenge.

If it is a solution to the CDH challenge, $\mathcal{S}$ has no more need of $\mathcal{A}$: it stops simulating, extracts the CDH response, and submits that to its challenger to win the game. If it is not a solution, it is possible to show based on the random oracle assumption that $\mathcal{A}$ can no longer detect whether $\mathcal{S}$ has replaced the value (roughly, because the random oracle gives independent random outputs on distinct inputs).

Barthe et al. [15] formalise this technique enough to give a machine-checked reduction to GDH for the NAXOS protocol, as well as a pen-and-paper reduction for HMQV.

PRF-ODH The key problem in both cases above is that in some cases the simulator $\mathcal{S}$ must be able to compute a value of the form $\mathrm{PRF}(\ldots, f(\ldots, u))$ for some f and PRF, where u is the challenge value it received from its own game. If $\mathcal{S}$ had an oracle for this function, therefore, its job would be straightforward: where necessary, $\mathcal{S}$ queries this oracle for the value it needs to know.

The PRF-ODH family of assumptions are that the DDH problem is hard even with access to exactly such an oracle. By construction, it is possible to reduce protocols like the above to the PRF-ODH problem: indeed, it was initially introduced by Jager et al. [96] as a reduction target for TLS.

The main advantage of PRF-ODH over GDH is that it does not require reductions to assume the ROM, and thus it can lead to "standard model" proofs. Unfortunately, Brendel et al. [41] give some strong impossibility results showing that in fact even the weaker variants of PRF-ODH cannot be reduced to common cryptographic problems without assuming the ROM, casting some doubt on the usefulness of PRF-ODH.

6.2 Not Quite Computational Soundness

We discussed symbolic approaches to protocol verification in §2.2.2 on page 20, but in the end chose to analyse Signal and ART with computational, pen-and-paper proofs. However, symbolic approaches to cryptography have been used from the start [72], and indeed the world of cryptography is often split into the two different camps.

Symbolic methods have two particular advantages over computational: they are easier to

- *formalise*, because their underlying term algebra is much simpler to understand than the computational world of interacting pptms; and

- *mechanise*, because reasoning about derivations of terms is much simpler than about relationships between similar probability distributions.

(We do not mean to imply that computational models cannot be formalised or mechanised. Indeed, both of these are active topics of research in the community.)

So, why do we not use symbolic models for all our analyses? The most common justification is that they do not capture as many attacks as computational models do, because the abstraction of cryptographic primitives as terms does not in fact fully reflect reality. That is, computational models are argued to be a closer approximation of the "real" instantiation of primitives in software, and thus a computational security reduction gives a more practical guarantee than a symbolic verification.

The particular class of attacks which can be captured computationally but not symbolically is somewhat challenging to pin down. There are certainly many practical considerations which are usually not considered in either class of approaches (see §2.2.2 on page 20), and many which are captured in both.

Example: Merkle-Damgård (MD) hash functions Many "computational-only" attacks have historically been those which depend on properties of cryptographic primitives not true of their abstract definition. These properties are often abstracted away symbolically, but captured in the computational model which directly uses the definition of the primitive.

One example of such a property comes from the MD construction [134], which builds a collision-resistant hash function from a collision-resistant one-way function, and is the basis of many popular cryptographic hash functions (including MD5, SHA1 and SHA2 but not SHA3). In most symbolic models, MD hash functions are modelled as abstract function symbols h with a single constructor, such that the only way to construct a term $h(x)$ is to apply h to the term x.

This is a good approximation for many purposes. However, MD hash functions have additional features which are not captured by the approximation as a function symbol. In particular, because they are constructed by splitting their input into blocks and feeding each block in turn into the compression function, given $h(x)$ it is easy to compute $h(x\|y)$ for any y. To do so, one simply needs to invoke the compression function on $h(x)$ and y. This is called a *length extension*.

Note that the existence of a length extension for a hash function h does not contradict collision resistance (since $h(x\|y) \neq h(x)$), and that it is not captured by the traditional symbolic definition (where the only way to construct $h(x\|y)$ is to apply h to $x\|y$). It *is* allowed by the computational definition, which requires only that it is hard to produce two separate inputs to h with the same output (§A.2.1.1).

Duong and Rizzo [75] noted that the Flickr API had implemented a custom authentication protocol: clients and the Flickr server shared a secret symmetric key k, and API requests with payload p were required to include the value $h(k\|p)$, with the goal of preventing network adversaries from forging requests without knowing k. However, Flickr instantiated their protocol with the MD hash function MD5, and thus a network adversary who overheard a valid API request $h(k\|p)$ could construct requests for any payload $p\|q$. We give an example of a related attack in §6.2.2.3.

The existence of a length extension broke the intended guarantee of Flickr's API authentication protocol, in a way that would not be captured by a traditional symbolic model of the protocol but would have prevented a computational proof.

There are many other attacks in this vein: breaking the authentication of the Station-to-Station (STS) protocol using an algebraic feature of many signature schemes, or the group negotiation of a DH protocol when a single weak group is present, or the confidentiality of Wi-Fi Protected Access (WPA) session keys due to the catastrophic failure of AES in Galois/Counter Mode (AES-GCM) with reused nonces.

Gaps like these between symbolic and computational models are often due not to a fundamental limitation of the symbolic approach but to historical modelling choices in the precise abstractions used for various cryptographic primitives. We'll illustrate the above examples with mechanised case studies, and show that a wide range of properties and attacks which traditionally lived only in the computational domain can in fact be captured relatively easily using current symbolic tools.

Computational Soundness A large body of literature in recent years aims to provide symbolic security properties which entail computational ones; that is, if there exists a computational attack then there also exists one in the symbolic model. Since the symbolic approximation is generally too optimistic, this line of research usually requires either strong computational assumptions or a weakened symbolic model.

Computational soundness results are notoriously challenging to prove, and often require strong assumptions on the underlying systems (e.g. the absence of key cycles, or the existence of a tagging scheme for terms). Our goal in this chapter is to produce symbolic models which, while better approximating practical primitives, do *not* necessarily admit computational soundness. This frees us from the task of solving a very hard problem indeed, while still allowing us to find a large class of attacks. In particular, any attack that we find is guaranteed to be an attack in the computational model.

6.2.1 The Tamarin Prover

We use the Tamarin prover [18, 65, 129, 155] as our symbolic verification tool of choice, mainly because of its expressive power: it allows for a wide range of equational theories [74] as well as protocols with complex conditional branches and loops.

There are many detailed explanations of Tamarin in the literature, and we refer the reader to the excellent manual [161] for a thorough introduction and pointers to further descriptions. We give here only a very brief sketch of its approach.

Tamarin takes as input a security protocol theory (spthy), which specifies a list of possible actions taken by the different participants in the protocol. For example, it might have a rule for the initiator to generate a random value and send out to the network a handshake message, and a rule for the responder to receive that message and process it. The security protocol theory (spthy) also contains rules for any abilities the adversary may have over and above active network control, and security properties written in a simple first-order language. Tamarin's output is a proof of (or counterexample to) the specified security properties, even under concurrent interleaving of arbitrarily many agents and active network control by the adversary and constructed by a backwards search. Its underlying execution model is a labelled transition system: at any time point, its state is a multiset of "facts" encoding

- the knowledge of the adversary,
- messages sent over the network, and
- the state of all protocol participants.

The actions in the spthy are interpreted as labelled multiset rewriting rules, turning one state (i.e., multiset of facts) into another. A sequence of rule applications gives rise to the sequence of rules labels, called a "trace", and the security properties are interpreted over these traces. For example, a simple secrecy property might be encoded as "there does not exist a trace in which there exists some value k which is a key and also known to the adversary".

Tamarin supports reasoning under arbitrary confluent and terminating equational theories satisfying the finite variant property [61]. That is, terms are considered equal if they are equal with respect to the any of the equations specified in the input spthy. For example, a classical signature scheme can be modelled in Tamarin with functions `sign`, `verify`, `pk` and `true` (of arities 2, 3, 1 and 0 respectively) and a single equation `verify(sign(m, sk), m, pk(sk)) = true`.

Tamarin was designed to support reasoning about DH terms, with binary functions $-\hat{}-$ and $- * -$ such that $(x\hat{}y)\hat{}z = x\hat{}(y * z)$. We use this builtin in our examples below, but otherwise specify all other equational theories explicitly.

6.2.2 Case Studies

We begin with a running example of a simple three-message signed DH protocol, depicted in Figure 6.1. An early version of this protocol was known as SIG-DH [50], and it is similar to STS [70], the SIGMA family [107] and a protocol from the ISO/IEC 9798 standard for entity authentication. We'll consider a few different variants, but all have the same core message flows: two parties Alex and Blake exchange DH ephemeral keys g^x and g^y as well as signatures and optionally MACs, deriving at the end of the protocol run a shared symmetric key. The goal of each protocol we consider is to provide a secret, authenticated session key between its two participants, against an active network adversary who has the power to additionally compromise certain agents' secret values.

Specifically, we will consider the following four cases:

(i) randomised signature schemes which fail on known random coins (§6.2.2.1),
(ii) signature schemes which have the Duplicate Signature Key Selection (DSKS) property (§6.2.2.2),
(iii) MACs implemented as a hash function with length extensions (§6.2.2.3), and
(iv) group negotiation in the presence of a weak DH group (§6.2.2.4).

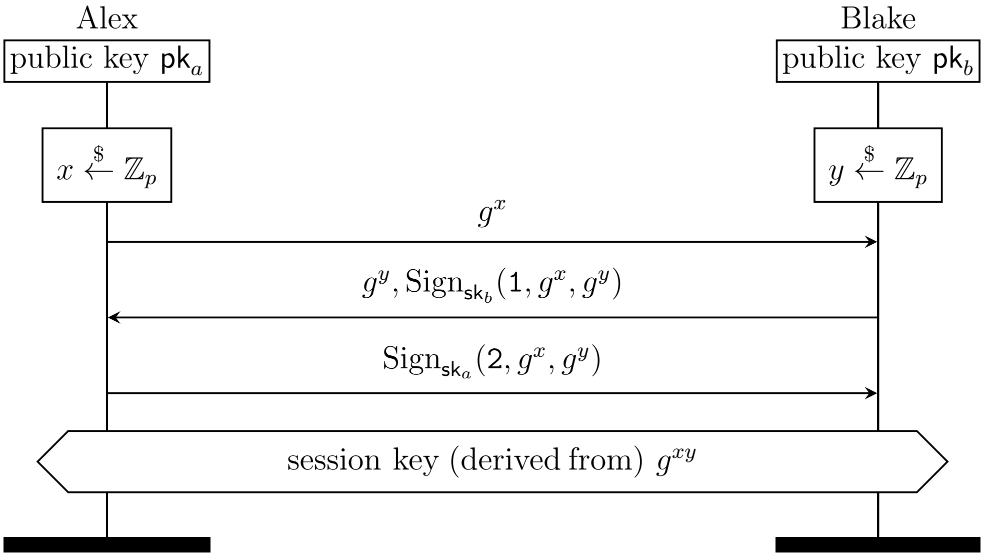

Alex
public key pk_a
Blake
public key pk_b
$x \xleftarrow{\$} \mathbb{Z}_p$
$y \xleftarrow{\$} \mathbb{Z}_p$
g^x
$g^y, \mathrm{Sign}_{\mathsf{sk}_b}(1, g^x, g^y)$
$\mathrm{Sign}_{\mathsf{sk}_a}(2, g^x, g^y)$
session key (derived from) g^{xy}

Figure 6.1: A simplified diagram of a three-message signed DH protocol. Many details (such as local computations, session identifiers and signature verifications) are omitted from this diagram in order to highlight the core message exchanges. We give a number of variants of this base protocol in Figure 6.2.

In each case, we will show that while traditional symbolic models do not capture a particular feature of their primitives that enables the known attack, it is possible to express the additional properties in Tamarin. We give Tamarin models finding the attack, and verify any suggested fixes.

6.2.2.1 Randomised Signatures

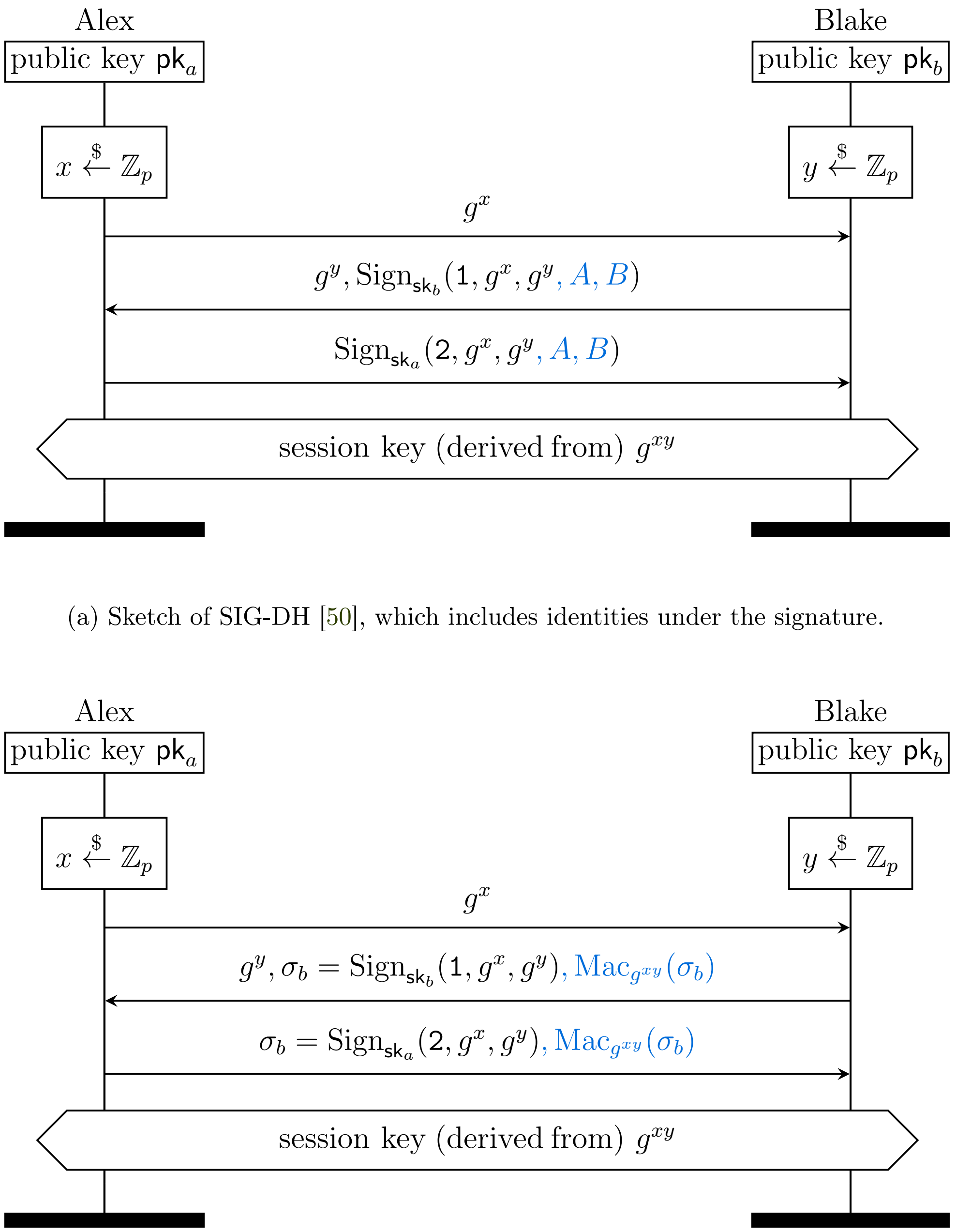
SIG-DH is intended to work with any signature scheme. However, LaMacchia, Lauter and Mityagin [112] note that many signature schemes are *randomised*, in the sense that the signature algorithm requires random coins in addition to the message and key. If the random coins are not kept secret, many signature schemes fail catastrophically and reveal the long-term key used for signing. SIG-DH may thus be insecure if the random coins used for the signature are revealed; indeed, this was one of the motivations for the design of the NAXOS protocol.

We can capture the attack on SIG-DH symbolically using Tamarin. To do so, we must first define equations for a randomised signature scheme:

```
  functions: verify/3, sign/2, pk/1, true/0
  equations: verify(sign(m, sk, coins), m, pk(sk)) = true
```

and a rule which reveals the long-term key if `coins` is known

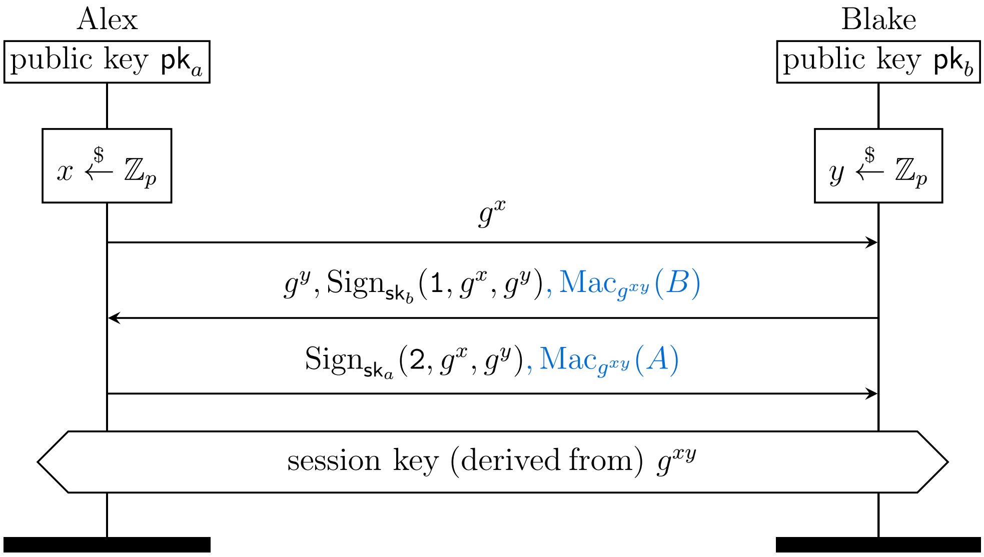
```
rule SignatureWithKnownRandomness:
      let signature = sign(message, key, coins)
      in
      [ In(<signature, coins>) ]
      --[ Kaboom(key) ]->
      [ Out(key) ]
```


Alex
public key pk_a
Blake
public key pk_b
$x \xleftarrow{\$} \mathbb{Z}_p$
$y \xleftarrow{\$} \mathbb{Z}_p$
g^x
$g^y, \mathrm{Sign}_{\mathsf{sk}_b}(1, g^x, g^y, A, B)$
$\mathrm{Sign}_{\mathsf{sk}_a}(2, g^x, g^y, A, B)$
session key (derived from) g^{xy}
Alex
public key pk_a
Blake
public key pk_b
$x \xleftarrow{\$} \mathbb{Z}_p$
$y \xleftarrow{\$} \mathbb{Z}_p$
g^x
$g^y, \sigma_b = \mathrm{Sign}_{\mathsf{sk}_b}(1, g^x, g^y), \mathrm{Mac}_{g^{xy}}(\sigma_b)$
$\sigma_b = \mathrm{Sign}_{\mathsf{sk}_a}(2, g^x, g^y), \mathrm{Mac}_{g^{xy}}(\sigma_b)$
session key (derived from) g^{xy}

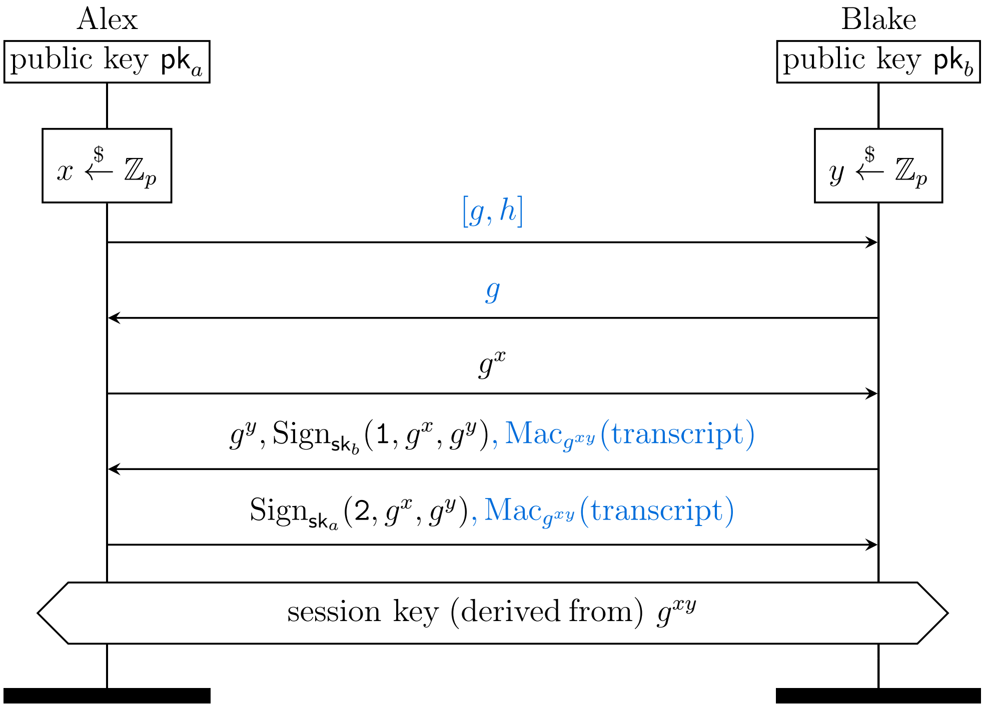
(a) Sketch of SIG-DH [50], which includes identities under the signature.

(b) Sketch of STS [70], which includes a MAC over the signature under the session key.

Figure 6.2: A zoo of signed-DH protocols. In each case, we mark in color the differences from the core protocol depicted in Figure 6.1. We describe in §6.2.2.4 attacks against each of these protocols which are not expressible in traditional symbolic models, and give a modern symbolic model to find them and verify any fixes.


Alex
public key pk_a
$x \xleftarrow{\$} \mathbb{Z}_p$
Blake
public key pk_b
$y \xleftarrow{\$} \mathbb{Z}_p$
g^x
$g^y, \mathrm{Sign}_{\mathsf{sk}_b}(1, g^x, g^y), \mathrm{Mac}_{g^{xy}}(B)$
$\mathrm{Sign}_{\mathsf{sk}_a}(2, g^x, g^y), \mathrm{Mac}_{g^{xy}}(A)$
session key (derived from) g^{xy}

(c) Sketch of SIGMA [107], which includes a MAC over the sender identity.


Alex
public key pk_a
$x \xleftarrow{\$} \mathbb{Z}_p$
Blake
public key pk_b
$y \xleftarrow{\$} \mathbb{Z}_p$
$[g, h]$
g
g^x
$g^y, \mathrm{Sign}_{\mathsf{sk}_b}(1, g^x, g^y), \mathrm{Mac}_{g^{xy}}(\mathrm{transcript})$
$\mathrm{Sign}_{\mathsf{sk}_a}(2, g^x, g^y), \mathrm{Mac}_{g^{xy}}(\mathrm{transcript})$
session key (derived from) g^{xy}

(d) Sketch of a basic DH negotiation phase: Alex proposes a list of groups, and Blake replies by choosing one. The choice is authenticated with a MAC over the entire transcript, consisting of the first three messages and the server ephemeral.

Figure 6.2: A zoo of signed-DH protocols, continued.

Remark 9 (Sources lemmas). In Tamarin, when using rules of this form, we often need to prove an additional "sources" lemma to prevent nontermination. Specifically, in its precomputation Tamarin inspects the premises of each rule, and for each fact generates a list of all possible combinations of rules which could produce that fact. In rules like the above, Tamarin cannot determine constraints on the term `key`, and leaves the source as a so-called partial deconstruction. Partial deconstructions often prevent termination, because Tamarin will consider them as a source for any possible term.

We can help Tamarin by proving a lemma about which terms can be produced by rules like the above. In our case, we prove that if a term `key` is output by the rule, then either

(i) the adversary knew `key` before the rule was invoked, or
(ii) `key` was used as a key by an honest protocol participant.

Tamarin proves this lemma automatically, and it prevents the partial deconstruction. Meier [128] gave the original description of sources lemmas and partial deconstructions, there called typing lemmas and open chains.

If we model SIG-DH (Figure 6.2a) with this type of signature scheme, we indeed find the attack described above. In this attack, an adversary relays messages for an honest session at some responder R, and then reveals the random coins of that session. Using the random coins and the signature which was transmitted in the clear, it derives the long-term key of R, whom it can subsequently impersonate to anyone.

LaMacchia, Lauter and Mityagin's suggested fix is the NAXOS protocol, which does not include signatures at all; Schmidt et al. [155] modelled NAXOS in Tamarin and verified its security in the eCK model.

6.2.2.2 DSKS Signatures and STS

For signature schemes with the DSKS property, given a signature $\sigma = \text{Sign}(m, sk)$ on a message m with secret key sk, it is possible to construct a public key pk' such that $\text{Verify}(pk', m, \sigma) = 1$. That is, pk' is a second public key against which the signature is also considered valid. Signatures can achieve Existential Unforgeability under an Adaptive Chosen Message Attack (EUF-CMA) while still admitting the DSKS property, because the former only requires that it is hard to forge a signature for a *given* key pair. Many signature schemes (including Rivest-Shamir-Adleman (RSA), Rabin, ElGamal, Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)) are of this form, at least in some circumstances.

It is possible to model such signature schemes in Tamarin, using an extended equational theory which adds this equation. Conventional signature schemes are defined by:

```
functions: verify/3, sign/2, pk/1, true/0
equations: verify(sign(m, sk), m, pk(sk)) = true
```

and the additional DSKS property can be modelled with:

```
functions: dsks/1
equations: verify(sign(m, sk), m, pk(dsks(sign(m, sk)))) = true
```

Blake-Wilson and Menezes [30] show a UKS attack on the STS protocol (depicted in Figure 6.2b where the identities are omitted) if it is instatiated with such a signature scheme. In this attack, an honest initiator I accepts a key with an honest responder R, but R accepts the same key for the adversary $\mathcal{A}$. To cause this, the adversary relays messages between the two parties, until the final message $\mathrm{Sign}(g^x \| g^y, sk_I)$. This message is signed under the long-term key of I, but R is expecting a signature under the long-term key of the adversary. However, $\mathcal{A}$ can use the DSKS property of the signature scheme to compute and register a new public key of their own, under which the signature indeed validates. R thus accepts the modified signature.

To find this attack in Tamarin, we extend the STS model of Schmidt et al. [155] with the equational theory above. The attack is found automatically in a few seconds on a modern desktop computer, with trace shown in Figure 6.3: the adversary intercepts an initiator's signature, registers the DSKS public key as owned by some agent $I.1$, and convinces a responder to accept the initiator's message as if it were from $I.1$.

One suggested fix is to include identities in the signatures. Indeed, with this modification Tamarin verifies the lack of an attack in a few seconds. We also verify that it is *not* enough for public key registrations to require proof of knowledge of the corresponding secret key: such a requirement prevents some attacks in the case of a non-DSKS signature scheme, but does not block the attack described above. We give the full models for both the broken and the repaired protocols in Appendix B.2.

6.2.2.3 Length Extension and MACs

SIGMA [107] is a well-studied protocol based closely on STS but with a more thorough analysis. Instead of using a signature over all of the important protocol-level data, it combines

- a signature with a long-term key over just the sender's DH ephemeral, and
- a MAC with the session key over the sender's identity

This allows it to provide a form of identity protection while still binding the signature to the participants' identities. We can state a minimal security property as a lemma:

```
lemma responder_key_agreement_without_length_extension:
  /* If the responder R accepts a key to use with the initiator I then... */
  "All sessionkey R I #i. RAccepts(sessionkey, R, I) @ i ==>

  /* either I previously accepted the same key to use with R */
    (Ex #j. #j < #i & IAccepts(sessionkey, I, R) @ j)

  /* or I's key was compromised beforehand */
    | (Ex #j. #j < #i & LtkRev(I) @ j)
```

This is a form of non-injective agreement on session keys and their purpose from the point of view of the responder, and Tamarin verifies it as true in a few seconds.

Remark 10. There is a natural dual property stating key agreement for the initiator: if the initiator accepts a key then the responder must have committed to it (though not yet necessarily accepted). This property unfortunately does not hold of SIGMA, because only the sender's identity—not the receiver's—is included under the MACs.

A common protocol error is to use a cryptographic hash function where a MAC is

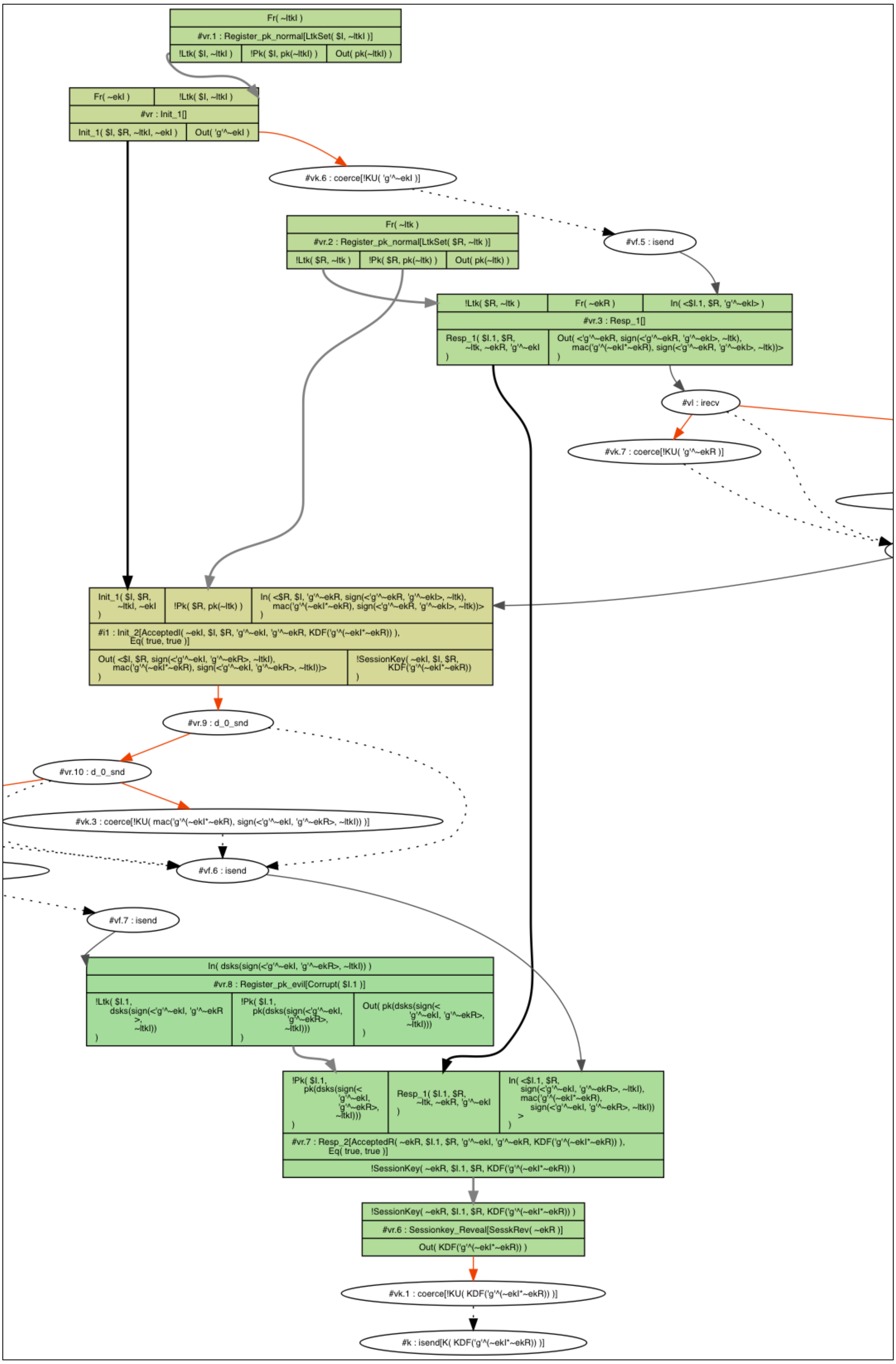

Fr(~ltkI)
#vr.1 : Register_pk_normal[LtkSet($I, ~ltkI)]
!Ltk($I, ~ltkI)
!Pk($I, pk(~ltkI))
Out(pk(~ltkI))
Fr(~ekI)
!Ltk($I, ~ltkI)
#vr : Init_1[]
Init_1($I, $R, ~ltkI, ~ekI)
Out('g'^~ekI)
#vk.6 : coerce[!KU('g'^~ekI)]
Fr(~ltk)
#vr.2 : Register_pk_normal[LtkSet($R, ~ltk)]
!Ltk($R, ~ltk)
!Pk($R, pk(~ltk))
Out(pk(~ltk))
#vf.5 : isend
!Ltk($R, ~ltk)
Fr(~ekR)
In(<$I.1, $R, 'g'^~ekI>)
#vr.3 : Resp_1[]
Resp_1($I.1, $R, ~ltk, ~ekR, 'g'^~ekI)
Out(<'g'^~ekR, sign(<'g'^~ekR, 'g'^~ekI>, ~ltk), mac('g'^(~ekI*~ekR), sign(<'g'^~ekR, 'g'^~ekI>, ~ltk))>)
#vl : irecv
#vk.7 : coerce[!KU('g'^~ekR)]
Init_1($I, $R, ~ltkI, ~ekI)
!Pk($R, pk(~ltk))
In(<$R, $I, 'g'^~ekR, sign(<'g'^~ekR, 'g'^~ekI>, ~ltk), mac('g'^(~ekI*~ekR), sign(<'g'^~ekR, 'g'^~ekI>, ~ltk))>)
#i1 : Init_2[AcceptedI(~ekI, $I, $R, 'g'^~ekI, 'g'^~ekR, KDF('g'^(~ekI*~ekR))), Eq(true, true)]
Out(<$I, $R, sign(<'g'^~ekI, 'g'^~ekR>, ~ltkI), mac('g'^(~ekI*~ekR), sign(<'g'^~ekI, 'g'^~ekR>, ~ltkI))>)
!SessionKey(~ekI, $I, $R, KDF('g'^(~ekI*~ekR)))
#vr.9 : d_0_snd
#vr.10 : d_0_snd
#vk.3 : coerce[!KU(mac('g'^(~ekI*~ekR), sign(<'g'^~ekI, 'g'^~ekR>, ~ltkI)))]
#vf.6 : isend
#vf.7 : isend
In(dsks(sign(<'g'^~ekI, 'g'^~ekR>, ~ltkI)))
#vr.8 : Register_pk_evil[Corrupt($I.1)]
!Ltk($I.1, dsks(sign(<'g'^~ekI, 'g'^~ekR >, ~ltkI)))
!Pk($I.1, pk(dsks(sign(<'g'^~ekI, 'g'^~ekR>, ~ltkI))))
Out(pk(dsks(sign(< 'g'^~ekI, 'g'^~ekR>, ~ltkI))))
!Pk($I.1, pk(dsks(sign(< 'g'^~ekI, 'g'^~ekR>, ~ltkI))))
Resp_1($I.1, $R, ~ltk, ~ekR, 'g'^~ekI)
In(<$I.1, $R, sign(<'g'^~ekI, 'g'^~ekR>, ~ltkI), mac('g'^(~ekI*~ekR), sign(<'g'^~ekI, 'g'^~ekR>, ~ltkI)) >)
#vr.7 : Resp_2[AcceptedR(~ekR, $I.1, $R, 'g'^~ekI, 'g'^~ekR, KDF('g'^(~ekI*~ekR))), Eq(true, true)]
!SessionKey(~ekR, $I.1, $R, KDF('g'^(~ekI*~ekR)))
!SessionKey(~ekR, $I.1, $R, KDF('g'^(~ekI*~ekR)))
#vr.6 : Sessionkey_Reveal[SesskRev(~ekR)]
Out(KDF('g'^(~ekI*~ekR)))
#vk.1 : coerce[!KU(KDF('g'^(~ekI*~ekR)))]
#k : isend[K(KDF('g'^(~ekI*~ekR)))]

Figure 6.3: (Truncated) TAMARIN graph showing a DSKS attack on the STS protocol. The attack trace shows two roles: an initator session at agent I with intended peer R, and a responder session at R with intended peer $I.1 \neq I$, constructed from a DSKS signature.

required, including the key as part of its input. The standard symbolic definition of these primitives is the same: an abstract function symbol. Until now, therefore, symbolic verification would find no attack on SIGMA where the MAC is instantiated as $h(k\|m)$ for some cryptographic hash function h.

It is possible to conduct a finer-grained analysis, as we show now. We can model length extensions on a hash function with an oracle rule in Tamarin:

```
rule AbstractLengthExtension:
    [ In(<h(<X, Y>), Z>) ]
    --[ Vwoooop() ]->
    [ Out(h(<X, Z>)) ]
```

Remark 11 (Pairing and associativity). The conventional symbolic model of term concatenation is pairing: functions $\mathtt{pair}, \mathtt{fst}, \mathtt{snd}$ and equations $\mathtt{fst}(\mathtt{pair}(x, y)) = x$ and $\mathtt{snd}(\mathtt{pair}(x, y)) = y$. (In Tamarin, $\langle x, y\rangle$ is syntactic sugar for $\mathtt{pair}(x, y)$.) However, unlike concatenation this pairing operation is not associative: $\big\langle x, \langle y, z\rangle\big\rangle \neq \big\langle\langle x, y\rangle, z\big\rangle$. This is because the equational theory of an associative pair operator does not have the finite variant property, which is necessary for its symbolic proof search to be correct. (That is, while the language is expressive enough to state associativity of a pair operator, Tamarin is not able to search for proofs in which that equational theory holds.)

The non-associativity of Tamarin's pair operator means that the naïve oracle

$$h(X), Z \mapsto h\big(\langle X, Z\rangle\big)$$

does not in fact faithfully model length extension attacks: extending $h\big(\langle k, m\rangle\big)$ with x would give $h\Big(\big\langle\langle k, m\rangle, x\big\rangle\Big) \neq h\Big(\big\langle k, \langle m, x\rangle\big\rangle\Big)$. There are various ways around this technical limitation, but for the illustrative context here we choose a simple one. Our oracle accepts only hashes of pairs, and instead of concatenating the extension to the original messages instead just overwrites it. This is an overapproximation, so it may find attacks which are not real.

With this rule in our model, we can return to SIGMA and verify its security property. To keep the model simple we limit the adversary to long-term and session key reveals (except against the target session), and prove only the limited property above. The full Tamarin input file is given in Appendix B.3 on page 142.

Tamarin finds an attack in this model in a few seconds. In the attack, Alex and Blake both accept a session key, but Blake believes the key is for Alex while Alex believes it is for Eli. In order to cause this, the adversary intercepts the second message from Blake to Alex, and uses the length extension to rewrite $h(g^{xy}\|B)$ to $h(g^{xy}\|E)$, which Alex accepts. We depict the attack flow in Figure 6.4.

The fix is to use a MAC instead of a hash function. This is equivalent to using our model but without the length extension oracle, and indeed Tamarin easily verifies the fix in a few seconds.

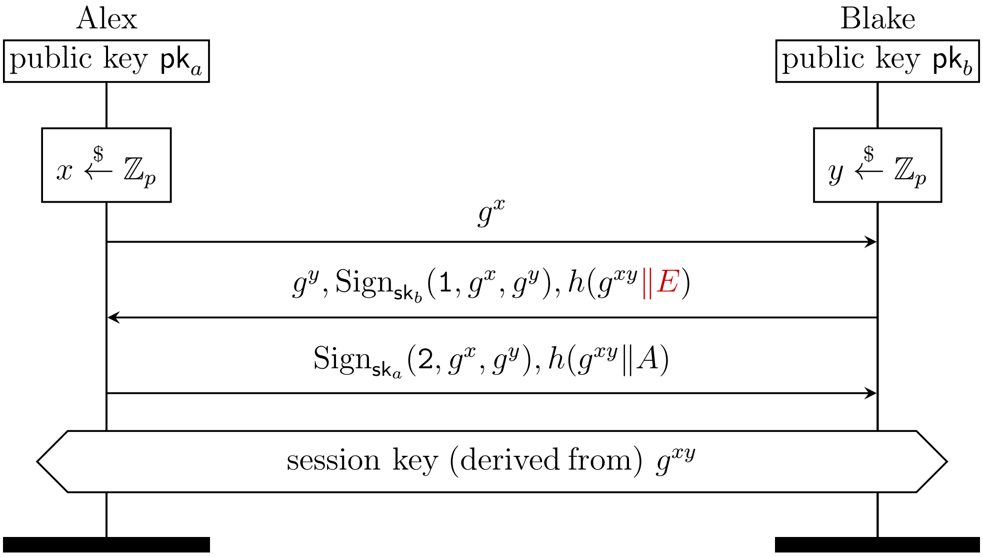

Alex
public key pk_a
Blake
public key pk_b
$x \xleftarrow{\$} \mathbb{Z}_p$
$y \xleftarrow{\$} \mathbb{Z}_p$
g^x
$g^y, \mathrm{Sign}_{\mathsf{sk}_b}(1, g^x, g^y), h(g^{xy} \| E)$
$\mathrm{Sign}_{\mathsf{sk}_a}(2, g^x, g^y), h(g^{xy} \| A)$
session key (derived from) g^{xy}

Figure 6.4: The attack on SIGMA if an extensible hash function is used in place of its MAC.

6.2.2.4 DH negotiation

Modern protocols usually support algorithm agility: instead of specifying the primitives to be used at the protocol layer, they instead begin with a negotiation phase in which the parties agree on a mutually-acceptable set of modes. For example, TLS clients and servers might support different DH groups, and so TLS allows the client to offer a list of supported groups and the server to select one. We give a simple example in Figure 6.2d, modifying SIGMA to allow the initiator to offer two groups and the responder to choose the one they prefer.

While protocol modes using strong primitives may be secure in isolation, various attacks have been found in negotiation protocols where some of the primitives are weak. Bhargavan et al. [28] give formal definitions and models for downgrade security in general. In a concrete example of a downgrade attack, Adrian et al. [4] found that many TLS servers were configured to accept 512-bit export grade cryptography, and that by computing discrete logarithms in these weak groups they could break TLS connections.

Tamarin can easily express weak DH groups using a rule

```
rule WeakDH:
     [ Exponential(h, x) ] --[ Dlog(x) ]-> [ Out(x) ]
```

where the protocol rules computing exponentiations are annotated with `Exponential` facts. We model a SIGMA variant with a negotiation phase in Tamarin; the full model is given in Appendix B.4. We remark that we use the m4 preprocessor to simplify writing the model.

Tamarin finds the known downgrade attack [28] in a few seconds. In this attack, an honest client offers a strong group (generator) g to the server, but the adversary modifies this message in transit to offer only a weak h. Both parties expect a MAC under the resulting session key in order before they accept it, but because h is a weak group the adversary can compute a discrete logarithm to learn the session key, and thus forge the expected MAC.

A standard fix for the downgrade attack, assuming the signature scheme is strong, is to include (a hash of) the transcript under the server signature. This is a simple change to the model, and we indeed verify that the attack is no longer present in this case.

6.2.2.5 Key Reinstallation Attack (KRACK)

KRACK [164] is an attack on the four-way handshake in WPA version 2, in which a client associates to an access point. By replaying a certain message, the attack causes a client to accept and "install" a traffic encryption key twice. The key comes with a replay counter which is usually set to zero when installing a key, and this leads to nonce reuse when a key is installed twice. This nonce reuse can catastrophically break authenticated encryption schemes such as AES-GCM.

To model KRACK we must capture the consequences of nonce reuse in Tamarin. We first give a symbolic definition of authenticated encryption with a nonce:

```
  equations: sndec(snenc(message, key, nonce), key) = message
```

We can then model the nonce reuse as a rule, taking as input two ciphertexts with the same key and nonce but different messages and outputting the key

```
rule NonceReuse:
     let m1 = snenc(x1, key, nonce)
         m2 = snenc(x2, key, nonce)
     in
     [ In(<m1, m2>) ]
     --[ Neq(x1, x2), Kaboom(key) ]->
     [ Out(key) ]
```

As before, we need to write a fairly straightforward sources lemma to help Tamarin reason about this lemma. This time, we prove that if nonce reuse reveals a key then either that key was already known to the adversary, or it was honestly generated by a protocol participant.

```
lemma nonce_reuse_isnt_magic[sources]:
      "All key #i. Kaboom(key) @ i ==> (
      // The adversary knew key beforehand
      (Ex #j. #j < #i & KU(key) @ j) |

      // key was used honestly before
      (Ex #j. #j < #i & HonestlyUsed(key) @ j)
      )"
```

KRACK is enabled by two different mechanisms which interact badly. First, various messages in WPA can be replayed and accepted by the recipient. Replayed messages overwrite the previous state. In particular, replaying a message which causes the recipient to derive a new key will cause them to rederive that key. Tamarin supports replays and loops in a fairly straightforward manner: since the underlying state is a multiset of facts, we simply need to write a rule which consumes later facts and produces earlier ones.

Second, WPA uses a counter-based nonce, starting from zero when a key is derived and incrementing the counter once per encryption. In particular, when a key is rederived due to a replay, its counter is reset to zero.

The following three rules model the retransmission of the third WPA message. The first rule models a “supplicant” (client) in the START state who receives an encrypted handshake message. Upon receiving that message they transition to the NEGOTIATING state; this is encoded in the rule by consuming the `Supp_PTK_START` fact and producing a `Supp_PTK_NEGOTIATING` fact instead.

The second rule models installing the received key. It consumes the state fact `Supp_PTK_NEGOTIATING` and produces `Supp_PTK_DONE`, as well as an `EncryptionKey` fact which is passed to the record layer for encrypting data messages.

Finally, the third rule models a supplicant which receives a replay of the encrypted handshake message after already installing the key. Its state transition is therefore from DONE back to NEGOTIATING.

```
rule Supp_recv_m3:
    let PTK = KDF(<~PMK, ANonce, SNonce>)
        ctr_plus_1 = ctr + ’0’
    m3 = <ctr_plus_1, senc(<’handshake’, GTK>, PTK)>
    m4 = ctr_plus_1
    in
    [ Supp_PTK_START(~suppID, ~PMK, ctr, ANonce, SNonce), In(m3) ]
    --[ SupplicantReceivesM3(~suppID, ~PMK, ctr) ]->
    [ Supp_PTK_NEGOTIATING(~suppID, ~PMK, PTK, GTK, ANonce, SNonce, ctr_plus_1)
    , Out(m4) ]

rule Supp_negotiate:
    [ Supp_PTK_NEGOTIATING(~suppID, ~PMK, PTK, GTK, ANonce, SNonce, ctr)
    , Fr(~kid) ]
    --[ SupplicantInstalled(~suppID, ~PMK, GTK, ctr) ]->
    [ Supp_PTK_DONE(~suppID, ~PMK, PTK, ANonce, SNonce, ctr)
    , EncryptionKey(~kid, PTK, ctr) ]

// These retransmissions use an incremented EAPOL replay counter.
rule Supp_rerecv_m3:
    let ctr_plus_1 = ctr + ’0’
        m3 = <ctr_plus_1, senc(<’handshake’, GTK>, PTK)>
        m4 = ctr_plus_1
    in
    [ Supp_PTK_DONE(~suppID, ~PMK, PTK, ANonce, SNonce, ctr), In(m3) ]
    --[ SupplicantRereceivesM3(~suppID, ~PMK, ctr_plus_1) ]->
    [ Supp_PTK_NEGOTIATING(~suppID, ~PMK, PTK, GTK, ANonce, SNonce, ctr_plus_1)
    , Out(m4) ]
```

Finally, we add one more rule to model encrypting data under an installed key, increasing the counter each time.

```
rule Encrypt:
    [ EncryptionKey(~id, key, nonce) ]
    --[ EncryptedWith(~id) ]->
    [ Out(snenc(<’data’, $message>, key, nonce))
    , EncryptionKey(~id, key, nonce + ’0’) ]
```

The full model is given in Appendix B.5 on page 146. It is based on the description of WPA by Vanhoef and Piessens [164], and includes the full handshake including retries of both the first and third messages. However, as a simple mechanism to limit the state space explosion due to the counters and retries, we artificially restrict Tamarin to a bounded number of rule invocations. The security properties we state are that the adversary cannot learn any key that is installed by either endpoint:

```
lemma gtk_secret_auth_without_extension:
    "All authID PMK GTK ctr #i. AuthenticatorInstalled(authID, PMK, GTK, ctr) @ i
     ==> (Ex key #j. Kaboom(key) @ j) | not(Ex #j. K(GTK) @ j)"

lemma gtk_secret_supp_without_extension:
    "All suppID PMK GTK ctr #i. SupplicantInstalled(suppID, PMK, GTK, ctr) @ i
     ==> (Ex key #j. Kaboom(key) @ j) | not(Ex #j. K(GTK) @ j)"
```

Note that nonce reuse is ruled out in these properties, due to the presence of the Kaboom action in the consequent of the lemma. Tamarin verifies them (in the bounded setting) in a few minutes on a modern desktop computer, though we stress that this does not correspond to a verification of WPA since it does not consider attacks which do not fit in the bounds. We can allow for nonce reuse by removing this consequent:

```
lemma gtk_secret_auth_with_extension:
    "All authID PMK GTK ctr #i. AuthenticatorInstalled(authID, PMK, GTK, ctr) @ i
     ==> not(Ex #j. K(GTK) @ j)"

lemma gtk_secret_supp_with_extension:
    "All suppID PMK GTK ctr #i. SupplicantInstalled(suppID, PMK, GTK, ctr) @ i
     ==> not(Ex #j. K(GTK) @ j)"
```

Tamarin now finds attacks on both. As expected, in the attack an honest handshake takes place and the supplicant sends a message encrypted under the resulting handshake key. The adversary then replays the final handshake message, and the supplicant reinstalls the key with the nonce (counter) reset to zero. When the supplicant then sends an encrypted message under the reset counter, the adversary can use the nonce reuse to derive the traffic key and violate the security property. Note that attacks in the bounded setting are still valid in the unbounded setting, and hence these correspond to actual attacks on the modelled protocol.

CHAPTER 7 CONCLUSIONS

The initial motivation for the research in this thesis was to study the Signal protocol, which we did in fairly fine detail. This required us to build a new computational framework which could capture the ratcheting design, and a security property which encodes PCS as well as other strong properties such as forward secrecy and KCI resistance. With all of this put together we gave a reduction of Signal to the PRF-ODH assumption in the ROM.

With Signal's analysis behind us, we carried on in two directions. First, since Signal does not natively support group messaging with PCS, we defined a new protocol called ART, which uses old ideas in a new way to provide efficient ratcheting in a group. This allows it to have efficient, asynchronous updates while still maintaining strong security properties. ART has had widespread interest from industry, and we are currently turning it into the MLS standard at the IETF.

Second, having set our work so far in computational frameworks, we returned to the symbolic model and start to tackle some of its limitations. In particular, we gave a number of case studies showing that many attacks which are traditionally captured only by computational analyses are in fact relatively easily expressible in Tamarin's input language. We found known attacks on a number of protocols using our models, and where relevant verified the fixes.

A brief summary of our contributions follows.

Signal We provided the first formal analysis of the Signal messaging protocol, used by more than a billion people:

- We developed a multi-stage key exchange security model with adversarial queries and freshness conditions that capture the security properties intended by Signal.
- We gave the first formal definition of Signal's security goals, including forward secrecy, KCI resistance and PCS and capturing the subtle differences between security properties of keys derived via symmetric and asymmetric ratcheting.
- We proved that the cryptographic core of Signal is secure in our model by

giving an explicit reduction to PRF-ODH in the random oracle model.

ART We designed ART, an efficient asynchronous tree-based group key exchange protocol that offers modern strong security properties including PCS:

- We gave a computational security model for ART.
- We explicitly reduced the unauthenticated core of our ART protocol to DDH, and performed a symbolic verification of its authentication property.

Symbolic Models We argued that symbolic techniques are a powerful tool for verifying ART and future protocols like it:

- We gave a series of case studies on a signed DH protocol, showing that randomised signatures, DSKS signatures, length extensions and weak DH groups can all be modelled symbolically.
- We give a particular example with the KRACK attack on WPA, finding the attack in with a bounded verification.

7.1 Future Work

Detailed conclusions were included at the end of each chapter. Instead of rehashing them further, we give three specific directions of future work which we see as continuing the spirit of this thesis.

Retry Logic The transport layer used to carry encrypted messages from mobile devices to the cloud tends to be unreliable, especially in areas with poor wireless connectivity. Abu-Salma et al. [3] conclude from multiple interviews that users actually value reliability over security, to the extent of using the former as a proxy for the latter. This has not gone unnoticed by service providers, who often decide to implement a retry mechanism to hide transient failures from users. When such a mechanism is in effect, endpoints may decide to re-send encrypted messages if they believe the initial transmission was lost.

This logic can go further than might be expected, and is directly relevant to message secrecy: WhatsApp will silently re-encrypt a sent message to a new key if it believes that the recipient has changed device, a feature originally branded a backdoor by the Guardian [84] but now referred to as a "tradeoff" [95]. This means that an adversary with temporarily control of the PKI can disconnect Blake, and Alex will silently re-send messages to them.

A formal analysis of the retry logic of WhatsApp and other implementations of Signal might highlight differences between them, as well as identify the precise impact on security properties caused by a given retry policy.

Public Key Infrastructure If a powerful adversary wished to attack Signal, perhaps the easiest approach would be to infiltrate or otherwise coerce the identity server. Indeed, as we saw in §5.5.1 on page 81, with control of this server a network adversary can attack any conversation by providing their own keys when asked for Blake's. They can even evade detection by actively forwarding messages. Signal has a fingerprint verification in which users can meet in person and confirm whether the identity server acted honestly, but this requires active participation and physical colocation by users and has not seen wide uptake.

Google's Key Transparency project aims to solve or at least reduce the impact of this problem, by storing all identity keys in a verifiable data structure which can be audited. This is a noble goal, but there are a number of concrete questions about the approach.

- Transparency logs have traditionally assumed the presence of "gossip", which is not present in most practical deployments. What is the impact of running a transparency log without gossip, and how do current candidates for gossip behave in the presence of nation-state adversaries?
- Are there ways to build gossip systems which are simpler and easier to analyse? Most academic proposals for transparency logs do not require it, instead taking the approach of having certain protocol agents sign off on data which would traditionally be disseminated through the gossip protocols.
- How does Key Transparency work when users can register multiple devices under the same identity? Should devices be required to authorise new registrations, and if not then which attacks are possible and when might they be detected?
- Is it possible to apply the theory of Milner et al. [136] about detection of disagreement to identity keys in a group messaging context? If so, can it be used to provide similar guarantees to those of Key Transparency, without the overhead of running a transparency log, auditors and a gossip protocol?

Systematic Modelling of Symbolic Primitives We saw a number of case studies in §6.2 in which we found known attacks on existing protocols by giving more accurate symbolic models of their cryptographic primitives. This worked well, but to be fully scientific would require a more systematic approach than the one we took earlier. Can we borrow from the ideas of computational soundness to give a less ad-hoc treatment of symbolic primitives? Is it possible to take existing TAMARIN models and modify the primitives they use without too much human intervention?

Appendices

APPENDIX A NOTATION

The following definitions are brief and not intended to be fully formal; they are included to fix our notation and clarify any ambiguities in word usage. We refer the reader to e.g., Katz and Lindell [99] for full definitions.

A.1 Basics

Functions and Variables We write $\{a, b, \dots\}$ for an unordered set of values, and $[a, b, \dots]$ for an ordered list. We write $x \leftarrow a$ to denote assigning a to the variable x, and $x \leftarrow f(a)$ to denote evaluating the function f at a and assigning the result to x.

A function f is injective or an injection if it preserves distinctness; that is, if $x \neq y$ implies $f(x) \neq f(y)$ for all x and y.

A function $\mathrm{negl}(x)$ is negligible if it is asymptotically less than any polynomial; equivalently, if for every positive integer c there exists a positive integer N_c such that $|\mathrm{negl}(x)| < {}^{1}\!/_{x^c}$ for all $x > N_c$. Negligibility is in terms of some security parameter x, but in order to avoid writing x everywhere we almost always omit it where it is clear, and abuse notation to say that certain values are negligible (meaning that they are a negligible function of the implicit security parameter).

Probabilities Where S is a probability distribution, we write $x \leftarrow_{\$} S$ to denote sampling a random element from the distribution S and assigning it to the variable x.

If f is a randomised function, we keep its randomness implicit instead of explicit, so that $f(a)$ is a probability distribution of outputs. Following the above, we write $x \leftarrow_{\$} f(a)$ to denote evaluating the randomised function on f, with uniformly sampled random input, and assigning the result to x.

An event is a set of possible outcomes i.e., values from a probability distribution. For example, the event "roll an even number on a six-sided die" corresponds to the subset $\{2, 4, 6\} \subset \{1, 2, 3, 4, 5, 6\}$ of possible outcomes. The probability of an event is the probability that it contains an outcome sampled from the distribution.

We say that two families $\{D_n\}_{n\in\mathbb{N}}$ and $\{E_n\}_{n\in\mathbb{N}}$ of probability distributions are computationally indistinguishable if the function $\delta_{\mathcal{A}}(n)$ is negligible in n for all pptms $\mathcal{A}$, where

$$\delta_{\mathcal{A}}(n) = \Big| \Pr\left[\mathcal{A}(x) = 1 | x \leftarrow D_n\right] - \Pr\left[\mathcal{A}(x) = 1 | x \leftarrow E_n\right] \Big|.$$

In other words, D_n and E_n are computationally indistinguishable if there is no pptm which can distinguish between samples from D_n and samples from E_n with non-negligible probability.

We call $\delta_{\mathcal{A}}(n)$ the *advantage* of $\mathcal{A}$ in distinguishing between D and E. We usually use the concept of advantage to measure the success probability of an adversary over and above random guessing in a game where it must guess a bit $b \in \{0, 1\}$ sampled uniformly at random by the challenger. In this context, the advantage of $\mathcal{A}$ is the difference of its success probability from $1/2$, the probability of success by random guessing. (Some authors define the advantage to be twice this value, since then it ranges from 0 to 1. Our advantages are all negligible, and thus the distinction is unimportant for us.)

Trees We define binary trees as a combination of

- *nodes*, which contain two nested children, and
- *leaves*, which contain no children,

along with associated data at each node and leaf: $\mathrm{tree} ::= (\mathrm{node}(\mathrm{tree}, \mathrm{tree}), \cdot) \,|\, (\mathrm{leaf}, \cdot)$. For a binary tree T, we use the notation $|T|$ to refer to the total number of leaves in the tree. We label each node of a tree with an index pair (x, y), where x represents the level of the node: the number of nodes (including the node itself) in the path to the root at index $(0, 0)$. The children of a node at index (x, y) are $(x + 1, 2y)$ and $(x + 1, 2y + 1)$. We write $T_{x,y}$ for the data at index (x, y) in a tree T. All tree nodes but the root have a parent node and a sibling (the other node directly contained in the parent). We refer to the *copath* of an node in a tree as the set comprising its sibling in the tree, the sibling of its parent node in the tree, and so on until reaching the root.

Groups We use groups extensively for DH. A group is a set G together with a binary operation $\cdot : G \times G \to G$ satisfying three properties: associativity of $\cdot$, existence of an identity element, and existence of inverses.

We write our groups multiplicatively (i.e. using $\times$ as the group operation) instead of additively (using $+$). For an integer x we define the exponentiation g^x for a group element g as

$$g^x = \overbrace{g \times g \times \cdots \times g}^{x \text{ times}}$$

Some authors prefer to write their groups additively, particularly when using groups of points on elliptic curves. In that notation, for a given base point P and integer x

they write xP

$$xP = \overbrace{P + P + \cdots P}^{x \text{ times}}.$$

This notation is equivalent and we use the former throughout this thesis.

The group $\mathbb{Z}_q$ is defined to be the set of congruence classes of integers modulo a prime q, equipped with the binary operation of multiplication and reduction modulo q; it is a straightforward exercise to show that this is a group.

A discrete logarithm $\log_b(a)$ is an integer k such that $b^k = a$. The discrete logarithm problem is the problem of computing $\log_b(a)$ for general groups G and elements a and b. (More precisely, an algorithm $\mathcal{A}$ solves the discrete logarithm problem if on input a group G, generator g and element $h \in G$ it outputs with non-negligible probability $x \in \mathbb{Z}_q$ such that $h = g^x$.) Computing discrete logarithms is considered to be computationally intractable except in special cases.

For most cryptographic protocols it is not enough to assume hardness of the discrete logarithm problem, and there are a number of stronger assumptions which are frequently made:

Decisional Diffie–Hellman (DDH) is the assumption that the probability distributions

$$\left\{(g, g^a, g^b, g^{ab}) \mid (a, b) \xleftarrow{\$} \mathbb{Z}_q^2\right\} \quad \text{and} \quad \left\{(g, g^a, g^b, g^c) \mid (a, b, c) \xleftarrow{\$} \mathbb{Z}_q^3\right\}$$

are computationally indistinguishable. We call (g, g^a, g^b, g^{ab}) a DDH tuple.
DDH implies hardness of discrete logarithms: it is easy to detect DDH tuples by taking the discrete logarithm of each element and comparing the last to the product of the middle two. However, there are groups for which DDH is not computationally hard:

- in $\mathbb{Z}_q^*$, where the Legendre symbol can be used to distinguish DDH tuples from random
- in an elliptic curve group with an efficiently computable bilinear pairing (such as those based on supersingular elliptic curves), where the pairing can be used to distinguish DDH tuples from random

Computational Diffie–Hellman (CDH) is the assumption that there is no pptm algorithm $\mathcal{A}$ which on input some group G, generator g and elements $h_1 = g^{x_1}, h_2 = g^{x_2} \in G$ outputs $g^{x_1 x_2}$ with non-negligible probability.
If solving discrete logarithms in G is easy then so is CDH: $\mathcal{A}$ outputs $h_2^{\log_g(h_1)}$.
CDH is a stronger assumption than hardness of discrete logarithms, though: if solving discrete logarithms in G is hard it is not known whether CDH is also hard.
If solving CDH in G is easy then so is solving DDH: $\mathcal{A}$ solves CDH for its challenge values g^a and g^b and compares the result to the challenge group element. The inverse is believed but not known to be false, with e.g., $\mathbb{Z}_p$ as an example group in which CDH is believed hard but DDH known easy.

CDH is believed to be hard in $\mathbb{Z}_p$ and in the prime subgroups of some elliptic curves.

Gap Diffie–Hellman (GDH) as defined by Okamoto and Pointcheval [140] is the assumption that CDH is computationally intractable even when given access to a DDH oracle. That is, GDH holds if there is no algorithm $\mathcal{A}$ which can with non-negligible probability output g^{ab} on input (g, g^a, g^b) even when given access to an oracle which determines whether a given tuple is a DDH tuple.

GDH is dual to DDH in a particular sense. Roughly [140],

- if GDH is easy then CDH is equivalent to DDH, and
- if DDH is easy then CDH is equivalent to GDH.

GDH is useful when combined with the ROM in proofs where certain values are replaced with challenge values from a CDH oracle but where computations depending on those values must be simulated. (For example, in some protocol proofs the session keys depend on DH values which are used in other sessions. In order to replace them with CDH challenge values, the changes to the other sessions must be simulated. The DDH oracle is used in these cases roughly to tell whether the adversary can detect the simulation.)

Some proofs use the "strong DH" assumption, which is similar to GDH but slightly weaker due to constraints on the oracle queries.

Pseudorandom Function Oracle Diffie–Hellman (PRF-ODH) [96] is another oracle-style DH assumption, initially introduced to study TLS 1.2. It is the assumption for some PRF that $\mathrm{PRF}(g^{uv}, x^\star)$ is indistinguishable from random for an adversary-chosen $x^\star$, even if given g^u and g^v and access to two oracles:

- an oracle outputting $\mathrm{PRF}(S^u, x)$ on input S and x, and
- an oracle outputting $\mathrm{PRF}(T^v, x)$ on input T and x.

PRF-ODH was originally proposed to avoid using the ROM in situations such as the above GDH example. Indeed, the PRF oracle is precisely necessary in order to simulate the required changes to other protocol sessions. However, Brendel et al. [41] give some impossibility results that indicate that in fact it is hard to prove security of certain types of PRF-ODH assumption without assuming the ROM.

A.2 Cryptographic Primitives

A.2.1 Definitions

A.2.1.1 Symmetric

Hash functions A cryptographic hash function is an efficient, deterministic function from arbitrary-length bit strings to fixed-length bit strings with one of the following three properties:

(i) collision resistance: it is hard to produce two messages m_1 and m_2 with $h(m_1) = h(m_2)$

(ii) preimage resistance: given an output $h(m)$ it is hard to produce m

(iii) second preimage resistance: given m_1, it is hard to produce m_2 with $h(m_1) = h(m_2)$

The formal definitions of hash function security are somewhat painful, but carefully catalogued by Rogaway and Shrimpton [151]. (One irritation is that collisions exist for all hash functions by the pigeonhole principle, so there exists an algorithm which outputs in constant time a precomputed collision. The definitions must therefore be over a family of hash functions instead of for a fixed one.)

Pseudorandom Functions A PRF, or pseudorandom function (family), is a family of efficiently-computable functions which are hard to distinguish from a random oracle, in the sense that if a function is drawn at random from the PRF distribution, then all of that function's outputs appear random regardless of its inputs. Formally, a function family F_k is a PRF if for all pptms $\mathcal{D}$

$$\left|\Pr\left[\mathcal{D}^{F_k}(n) = 1 \mid k \xleftarrow{\$} \{0,1\}^n\right] - \Pr\left[\mathcal{D}^{f}(n) = 1 \mid f \xleftarrow{\$} \{0,1\}^{\{0,1\}}\right]\right| \leq \mathrm{negl}(n)$$

where we write $\mathcal{D}^g$ to indicate that $\mathcal{D}$ has oracle access to the function g.

Encryption A symmetric encryption scheme is a pair (Enc, Dec) of ppt algorithms:

- Enc outputs a ciphertext c on input a key $k \xleftarrow{\$} \{0,1\}^n$ and message $m \in \{0,1\}*$, and
- Dec outputs a message (or an error $\perp$) on input a key k and ciphertext c.

We require for all keys k and messages m that $\mathrm{Dec}(k, \mathrm{Enc}(k, m)) = m$, and usually write $\{m\}_k$ for $\mathrm{Enc}(k, m)$.

The strongest widely-used definition of security for such schemes is Indistinguishability under an Adaptive Chosen-Ciphertext Attack (IND-CCA), requiring than an adversary cannot with non-negligible probability distinguish between encryptions of two chosen messages under an unknown key, even when given access to encryption and decryption oracles for that key. We refer the reader to any standard textbook for a formal definition.

Message Authentication Codes A MAC scheme is a pair (Mac, Verify) of ppt algorithms:

- Mac outputs a tag t on input a key $k \xleftarrow{\$} \{0,1\}^n$ and a message $m \in \{0,1\}^*$, and
- Verify outputs a bit $b \in \{0,1\}$ on input a key k, a message m and a tag t.

We require for all keys k and messages m that $\mathrm{Verify}(k, m, \mathrm{Mac}(k, m)) = 1$.

The standard definition of security for a MAC is EUF-CMA, requiring that an adversary cannot with non-negligible probability output a new valid tag for any message under an unknown key, even when given access to an Mac oracle under that key. We refer the reader to any standard textbook for a formal definition.

Authenticated Encryption with Associated Data (AEAD) [150] is a form of symmetric encryption with two added features: first it provides integrity i.e., one must know the key in order to construct a valid ciphertext, and second it allows for "associated data" which is integrity-protected but not confidential. This turns out to be a very useful notion for practical protocols: integrity is important in real systems, and packet headers must not be confidential. An Authenticated Encryption with Associated Data (AEAD) scheme is a pair $(\mathrm{Enc}, \mathrm{Dec})$ of ppt algorithms:

- Enc outputs a ciphertext c on input a key k, message m, nonce n and header h, and
- Dec outputs a message m (or an error $\perp$) on input a key k, nonce n, header h and ciphertext c.

As for symmetric encryption, we require decryption of a valid ciphertext to return the original plaintext.

The definition of security for an AEAD scheme has two parts. First, ciphertexts must be indistinguishable from random bitstrings under chosen plaintext attacks (a stronger notion than the traditional chosen-plaintext one, which requires ciphertexts to be indistinguishable from *the encryptions of* random bitstrings). Second, it must be hard for an adversary to generate a new valid ciphertext even when given access to an encryption oracle. We refer the reader to Rogaway [150] for the formal definitions.

A.2.1.2 Asymmetric

Signatures A digital signature scheme is a tuple $(\mathrm{KeyGen}, \mathrm{Sign}, \mathrm{Verify})$ of ppt algorithms:

- KeyGen outputs a pair $(\mathsf{pk}, \mathsf{sk})$ of keys on input the security parameter n,
- Sign outputs a signature σ on input a private key sk and message m, and
- Verify outputs a bit $b \in \{0, 1\}$ on input a public key pk, message m and signature σ.

We require that for all n and all $(\mathsf{pk}, sk) \overset{\$}{\leftarrow} \mathrm{KeyGen}(n)$, $\mathrm{Verify}(\mathsf{pk}, m, \mathrm{Sign}(\mathsf{sk}, m)) = 1$ except with negligible probability.

The standard definition of security for signature schemes is EUF-CMA, requiring that an adversary cannot output a new signature for a chosen message m which is valid against a known public key pk, even when given access to a signature oracle under the corresponding secret key sk. We refer the reader to any standard textbook for a formal definition.

Encryption An asymmetric encryption scheme is very similar to a symmetric encryption scheme, except with distinct keys for encryption and decryption. It comprises a tuple $(\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$ of ppt algorithms:

- KeyGen outputs a pair $(\mathsf{pk}, \mathsf{sk})$ of keys on input the security parameter n,
- Enc outputs a ciphertext c on input a public key pk and message m, and
- Dec outputs a message m or special symbol $\perp$ on input a private key sk and ciphertext c.

The strongest commonly-used definition of security here is again Indistinguishability under an Adaptive Chosen-Ciphertext Attack (IND-CCA), though this time the encryption oracle is unnecessary since the adversary is given access to the public key and can compute encryptions itself. Again, we refer the reader to any standard textbook for a formal definition.

APPENDIX B TAMARIN MODELS

B.1 Randomised Signatures

We give a model finding an attack on a SIG-DH protocol with randomised signatures which leak the signing key if their random input is known.

```
/*
 * Protocol:  SIG-DH with leaky randomised signatures
 * Modeler:   Katriel Cohn-Gordon
 * Date:  April 2018
 * Status:  working
 */

theory leaky_randomised_signatures
begin

builtins: diffie-hellman

functions: verify/3, sign/3, pk/1, true/0                // Signature scheme
functions: KDF/2, h/1                                    // Other primitives
equations: verify(sign(m, sk, coins), m, pk(sk)) = true // Normal signatures

section{* SIG-DH *}

// Honest parties can register fresh keys
rule Register_pk_normal:
  [ Fr(~ltk) ]
  --[ LtkSet($A, ~ltk) ]->
  [ !Ltk($A, ~ltk), !Pk($A, pk(~ltk)), Out(pk(~ltk)) ]

rule Init_1:
  let ekI = h(<'private_key', ~coinsI>)
      epkI = 'g' ^ ekI
      sigCoins = h(<'signature_coins', ~coinsI>)
      m1 = epkI
  in
  [ Fr(~coinsI), !Ltk($I, ~ltkI) ]
  --[ SidI_1(~coinsI, $I, $R, epkI), Neq($I, $R) ]->
  [ Init_1($I, $R, ~ltkI, ~coinsI), !Random(~coinsI), Out(m1) ]

rule Resp_1:
  let m1 = epkI
      ekR = h(<'private_key', ~coinsR>)
      epkR = 'g' ^ ekR
      sigCoins = h(<'signature_coins', ~coinsR>)
      sigR = sign(<'R', epkI, epkR, $I, $R>, ~ltkR, sigCoins)
      pms = epkI ^ ekR
      m2 = <epkR, sigR>
  in
    [ !Ltk($R, ~ltkR)
    , Fr(~coinsR)
    , In(m1)
    ]
  --[ SidR_1(~coinsR, $I, $R, epkI, epkR, pms)
    , Neq($I, $R)
    , SignedWith(~ltkR)
    ]->
    [ Resp_1(~coinsR, $I, $R, ekR, epkI, pms)
    , !Random(~coinsR)
    , Out(m2)
    ]

rule Init_2:
  let ekI = h(<’private_key’, ~coinsI>)
      epkR = ’g’^ekR
      peerTid = ekR
      m2 = <epkR, sigR>
      epkI = ’g’ ^ ekI
      sigCoins = h(<’signature_coins’, ~coinsI>)
      sigI = sign(<’I’, epkR, epkI, $I, $R>, ~ltkI, sigCoins)
      pms = epkR ^ ekI
      m3 = sigI
      sessionkey = KDF(’sk’, pms)
  in
    [ Init_1($I, $R, ~ltkI, ~coinsI)
    , !Pk($R, pkR)
    , In(m2)
    ]
  --[ SidI_2(~coinsI, $I, $R, epkI, epkR, pms)
    , Accepted(ekI, peerTid, $I, $R, epkI, epkR, sessionkey)
    , Eq(verify(sigR, <’R’, epkI, epkR, $I, $R>, pkR), true)
    , SignedWith(~ltkI)
    ]->
    [ Out(m3),
      !SessionKey(~coinsI, $I, $R, sessionkey)
    ]

rule Resp_2:
  let epkR = ’g’ ^ ekR
      epkI = ’g’ ^ peerTid
      m3 = sigI
      sessionkey = KDF(’sk’, pms)
  in
    [ !Pk($I, pkI)
    , Resp_1(~coinsR, $I, $R, ekR, epkI, pms )
    , In(m3)
    ]
  --[ SidR_2(~coinsR, $I, $R, epkI, epkR, pms)
    , Accepted(ekR, peerTid, $R, $I, epkI, epkR, sessionkey)
    , Eq(verify(sigI, <’I’, epkR, epkI, $I, $R>, pkI), true)
     ]->
    [ !SessionKey(~coinsR, $I, $R, pms) ]


// ------------- ADVERSARY POWERS    -------------

// learn any session key
// rule SesskRev:
//     [ !SessionKey(~tid, $I, $R, sessionkey) ]
//   --[ SesskRev(~tid) ]->
//     [ Out(sessionkey) ]

// learn any long-term key
rule LtkRev:
     [ !Ltk($X, ltkX) ]
     --[ LtkRev($X) ]->
     [ Out(ltkX) ]

// learn any random coins
rule RandomRev:
     [ !Random(coins) ]
     --[ RandomRev(coins) ]->
     [ Out(coins) ]

// given a signature and its coins, learn the signature key
rule SignatureWithKnownRandomness:
     let signature = sign(message, key, coins)
     in
     [ In(<signature, coins>) ]
     --[ Kaboom(key) ]->
     [ Out(key) ]


// ------------- SECURITY PROPERTIES -------------

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

restriction inequalities_hold:
      "All x y #i. Neq(x, y) @ i ==> not(x = y)"

lemma sane: exists-trace
  "Ex coinsI coinsR I R epkI epkR sessionkey #i #j.
      Accepted(coinsI, coinsR, I, R, epkI, epkR, sessionkey) @ i
    & Accepted(coinsR, coinsI, R, I, epkI, epkR, sessionkey) @ j
    & not(coinsI = coinsR)
  "

/* We need a sources lemma for the SignatureWithKnownRandomness rule, since it
can output arbitrary terms. Specifically, we prove that either it reveals a
party’s random coins, or the adversary knew the key it would reveal
beforehand. */

```

```
lemma signature_typing[sources]:
      " All key #i. Kaboom(key) @ i ==>

      /* either the adversary knew the key beforehand */
      (Ex #j. #j < #i & KU(key) @ j) |

      /* or it was actually a key used to sign something by an honest party */
      (Ex #j. SignedWith(key) @ j)
      "

/* This is a weak cleanness predicate for an AKE protocol. We allow corruptions of
arbitrary random coins and long-term keys, except
  - no corruption of the randomnes of the Test session or its partner
  - no corruption of the long-term key of Test session's peer before it completes

We leave out session key reveals (which are necessary for actual protocols) to
keep the property simple, and for this property we also prevent using random
coins to learn the responder's long-term signature key.  */

lemma secrecy_of_initiator_key_without_derandomising_signatures:
  "not (Ex #i1 #i2 ttest I R k epkI epkR.
            SidI_2(ttest, I, R, epkI, epkR, k) @ i1 & K( k ) @ i2

      & not(Ex #i3 #i4 ltkR. LtkSet(R, ltkR) @ i3 & Kaboom(ltkR) @ i4)
      /* Didn't reveal the randomness of the test session */
            /* Note that the session key is g^xy, so revealing the randomness of
            the test session reveals the session key. Using the NAXOS trick
            would prevent this attack, in which case which we could uncomment
            the LtkRev. */
            & not(Ex /* #i3 */ #i4. /* LtkRev(I) @ i3 & */ RandomRev( ttest ) @ i4)

            /* Same comment as above but for matching session */
      /* Note that we need uksID to stop finding the UKS attack. */
            & not(Ex #i3 /* #i4 */ #i5 tpartner kpartner uksID.
                  SidR_1( tpartner,/*I*/ uksID,R,epkI,epkR,kpartner ) @i3
      /* & LtkRev( R ) @ i4 */
      & RandomRev( tpartner ) @ i5 )

      /* Longterm-key-reveal of partner only if there is a matching session. */
            & (All #i3. LtkRev( R ) @ i3 ==>
            (Ex #i4 tpartner kpartner.
                      /* (i1 < i3) | */
                      SidR_1( tpartner,I,R,epkI,epkR,kpartner ) @i4)))"

/* As above, but now allow learning R's ltk from a derandomised signature */
lemma secrecy_of_initiator_key_with_derandomising_signatures:
  "not (Ex #i1 #i2 ttest I R k epkI epkR.
            SidI_2(ttest, I, R, epkI, epkR, k) @ i1 & K( k ) @ i2
            & not(Ex #i4. RandomRev( ttest ) @ i4)
            & not(Ex #i3 #i5 tpartner kpartner uksID.
              SidR_1(tpartner, uksID, R, epkI, epkR, kpartner) @ i3
      & RandomRev( tpartner ) @ i5)
            & (All #i3. LtkRev( R ) @ i3 ==> (Ex #i4 tpartner kpartner.
              SidR_1( tpartner,I,R,epkI,epkR,kpartner ) @i4)))"

end
```

B.2 UKS attack on STS with DSKS signatures

```
/*
 * Protocol:  Station-To-Station, MAC variant, DSKS attacks
 * Modeler:   Cas Cremers, Katriel Cohn-Gordon
 * Date:  November 2016
 * Source:  "Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol"
 *    Blake-Wilson, Simon and Menezes, Alfred
 *    PKC '99, Springer, 1999
 *
 * Status:  working
 */

theory dsks_on_STS
begin

builtins: diffie-hellman

functions: verify/3, sign/2, pk/1, true/0, dsks/1               // Signature scheme
functions: h/1, KDF/1, mac/2                                    // Other primitives
equations: verify(sign(m, sk), m, pk(sk)) = true                // Normal signatures
         , verify(sign(m, sk), m, pk(dsks(sign(m, sk)))) = true // DSKS property

// In a DSKS attack, the attacker can take a signature sign(m, sk) of a message m,
// and from it construct a *new* public key pk(dsks(sign(m, sk))) against which
// the signature *also* validates.

section{* The Station-To-Station Protocol (MAC version) *}

// Honest parties can register fresh keys
```

```
rule Register_pk_normal:
  [ Fr(~ltk) ]
  --[ LtkSet($A, ~ltk) ]->
  [ !Ltk($A, ~ltk), !Pk($A, pk(~ltk)), Out(pk(~ltk)) ]

// Evil parties can register any keys, but only against corrupted parties
rule Register_pk_evil:
  [ In(k) ]
  --[ Corrupt($E) ]->
  [ !Ltk($E, k), !Pk($E, pk(k)), Out(pk(k)) ]

// Attacker can learn any session key (rendering the session non-fresh)
rule Sessionkey_Reveal:
    [ !SessionKey(~tid, $I,$R,k) ]
  --[ SesskRev(~tid) ]->
    [ Out(k) ]

// Protocol
rule Init_1:
  [ Fr(~ekI), !Ltk($I, ~ltkI) ]
  -->
  [ Init_1($I, $R, ~ltkI, ~ekI ), Out(< 'g'^~ekI >) ]

rule Resp_1:
  let sigR = sign(<'g'^~ekR, X>, ~ltkR)
  in
    [ !Ltk($R, ~ltkR)
    , Fr(~ekR)
    , In( <$I, $R, X > ) ]
  -->
    [ Resp_1( $I, $R, ~ltkR, ~ekR, X )
    , Out(< 'g'^~ekR, sigR, mac(X^~ekR, sigR) >)
    ]

rule Init_2:
  let epkI = 'g'^~ekI
      sigI = sign(<epkI, Y>, ~ltkI)
      pms = Y^~ekI
  in
    [ Init_1( $I, $R, ~ltkI, ~ekI )
    , !Pk($R, pkR)
    , In(< $R, $I, Y, sigR, mac(pms, sigR) >)
    ]
  --[ AcceptedI(~ekI,$I,$R,epkI,Y, KDF(pms))
    , Eq(verify(sigR, <Y, epkI>, pkR), true) ]->
    [ Out(< $I, $R, sigI, mac(pms, sigI) >),
      !SessionKey(~ekI,$I,$R, KDF(pms))
    ]

rule Resp_2:
  let epkR = 'g'^~ekR
      pms = X^~ekR
  in
    [ !Pk($I, pkI)
    , Resp_1( $I, $R, ~ltkR, ~ekR, X )
    , In( <$I, $R, sigI, mac(pms, sigI) >)
    ]
  --[ AcceptedR(~ekR,$I,$R,X,epkR, KDF(pms))
    , Eq(verify(sigI, <X, epkR>, pkI), true) ]->
    [ !SessionKey(~ekR,$I,$R, KDF(pms)) ]


// ------------- SECURITY PROPERTIES -------------

axiom equalities_hold:
      "All x y #i. Eq(x, y) @ i ==> x = y"

lemma KI_Perfect_Forward_Secrecy_I:
  "not (Ex ttest I R sessKey #i1 #k hki hkr.
     AcceptedI(ttest,I,R,hki,hkr,sessKey) @ i1 &
     not (Ex #r. Corrupt(I) @ r) &
     not (Ex #r. Corrupt(R) @ r) &
     K(sessKey) @ k &
     // No session keymat reveal of test
     not (Ex #i3. SesskRev(ttest) @ i3) &
     // No session keymat reveal of partner
     not (Ex #i3 #i4 tpartner kpartner. SesskRev(tpartner) @ i3
          & AcceptedR(tpartner,I,R,hki,hkr,kpartner) @ i4
         )
     )
  "

lemma KI_Perfect_Forward_Secrecy_R:
  "not (Ex ttest I R sessKey #i1 #k hki hkr.
     AcceptedR(ttest,I,R,hki,hkr,sessKey) @ i1 &
     not (Ex #r. Corrupt(I) @ r) &
     not (Ex #r. Corrupt(R) @ r) &
     K(sessKey) @ k &
     // No session keymat reveal of test
     not (Ex #i2. SesskRev(ttest) @ i2) &
     // No session keymat reveal of partner
     not (Ex #i2 #i3 tpartner kpartner. SesskRev(tpartner) @ i2
          & AcceptedI(tpartner,I,R,hki,hkr,kpartner) @ i3
         )
     )
  "


end
```

```
/*
 * Protocol:  Station-To-Station, MAC variant, DSKS attacks
 * Modeler:   Cas Cremers, Katriel Cohn-Gordon
 * Date:  November 2016
 * Source:  "Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol"
 *     Blake-Wilson, Simon and Menezes, Alfred
 *     PKC '99, Springer, 1999
 *
 * Status:  working
 */

theory dsks_on_STS_fixed
begin

builtins: diffie-hellman

functions: verify/3, sign/2, pk/1, true/0, dsks/1                // Signature scheme
functions: h/1, KDF/1, mac/2                                     // Other primitives
equations: verify(sign(m, sk), m, pk(sk)) = true                 // Normal signatures
         , verify(sign(m, sk), m, pk(dsks(sign(m, sk)))) = true // DSKS property

// In a DSKS attack, the attacker can take a signature sign(m, sk) of a message m,
// and from it construct a *new* public key pk(dsks(sign(m, sk))) against which
// the signature *also* validates.

section{* The Station-To-Station Protocol (MAC version) *}

// Honest parties can register fresh keys
rule Register_pk_normal:
  [ Fr(~ltk) ]
  --[ LtkSet($A, ~ltk) ]->
  [ !Ltk($A, ~ltk), !Pk($A, pk(~ltk)), Out(pk(~ltk)) ]

// Evil parties can register any keys, but only against corrupted parties
rule Register_pk_evil:
  [ In(k) ]
  --[ Corrupt($E) ]->
  [ !Ltk($E, k), !Pk($E, pk(k)), Out(pk(k)) ]

// Attacker can learn any session key (rendering the session non-fresh)
rule Sessionkey_Reveal:
    [ !SessionKey(~tid, $I,$R,k) ]
  --[ SesskRev(~tid) ]->
    [ Out(k) ]

// Protocol
rule Init_1:
  [ Fr(~ekI), !Ltk($I, ~ltkI) ]
  -->
  [ Init_1($I, $R, ~ltkI, ~ekI ), Out(< 'g'^~ekI >) ]

rule Resp_1:
  let sigR = sign(<'1', 'g'^~ekR, X, $I, $R>, ~ltkR)
  in
    [ !Ltk($R, ~ltkR)
    , Fr(~ekR)
    , In( <$I, $R, X > ) ]
  -->
    [ Resp_1( $I, $R, ~ltkR, ~ekR, X )
    , Out(< 'g'^~ekR, sigR, mac(X^~ekR, sigR) >)
    ]

rule Init_2:
  let epkI = 'g'^~ekI
      sigI = sign(<'2', epkI, Y, $I, $R>, ~ltkI)
      pms = Y^~ekI
  in
    [ Init_1( $I, $R, ~ltkI, ~ekI )
    , !Pk($R, pkR)
    , In(< $R, $I, Y, sigR, mac(pms, sigR) >)
    ]
  --[ AcceptedI(~ekI,$I,$R,epkI,Y, KDF(pms))
    , Eq(verify(sigR, <'1', Y, epkI, $I, $R>, pkR), true) ]->
    [ Out(< $I, $R, sigI, mac(pms, sigI) >),
      !SessionKey(~ekI,$I,$R, KDF(pms))
    ]

rule Resp_2:
  let epkR = 'g'^~ekR
      pms = X^~ekR
  in
    [ !Pk($I, pkI)
    , Resp_1( $I, $R, ~ltkR, ~ekR, X )
    , In( <$I, $R, sigI, mac(pms, sigI) >)
    ]
  --[ AcceptedR(~ekR,$I,$R,X,epkR, KDF(pms))
    , Eq(verify(sigI, <'2', X, epkR, $I, $R>, pkI), true) ]->
    [ !SessionKey(~ekR,$I,$R, KDF(pms)) ]


// ------------- SECURITY PROPERTIES -------------

axiom equalities_hold:
      "All x y #i. Eq(x, y) @ i ==> x = y"

lemma KI_Perfect_Forward_Secrecy_I:
  "not (Ex ttest I R sessKey #i1 #k hki hkr.
     AcceptedI(ttest,I,R,hki,hkr,sessKey) @ i1 &
     not (Ex #r. Corrupt(I) @ r) &
     not (Ex #r. Corrupt(R) @ r) &
     K(sessKey) @ k &
     // No session keymat reveal of test
     not (Ex #i3. SesskRev(ttest) @ i3) &
     // No session keymat reveal of partner
     not (Ex #i3 #i4 tpartner kpartner. SesskRev(tpartner) @ i3
          & AcceptedR(tpartner,I,R,hki,hkr,kpartner) @ i4
         )
     )
  "

lemma KI_Perfect_Forward_Secrecy_R:
  "not (Ex ttest I R sessKey #i1 #k hki hkr.
     AcceptedR(ttest,I,R,hki,hkr,sessKey) @ i1 &
     not (Ex #r. Corrupt(I) @ r) &
     not (Ex #r. Corrupt(R) @ r) &
     K(sessKey) @ k &
     // No session keymat reveal of test
     not (Ex #i2. SesskRev(ttest) @ i2) &
     // No session keymat reveal of partner
     not (Ex #i2 #i3 tpartner kpartner. SesskRev(tpartner) @ i2
          & AcceptedI(tpartner,I,R,hki,hkr,kpartner) @ i3
         )
     )
  "


end
```

B.3 SIGMA with an extensible hash function

```
/*
 * Protocol:  SIGMA, with length extensions
 * Modeler:   Katriel Cohn-Gordon
 * Date:        April 2018
 * Status:  working
 */

theory sigma_with_length_extensions
begin

builtins: diffie-hellman

functions: verify/3, sign/2, pk/1, true/0                        // Signature scheme
equations: verify(sign(m, sk), m, pk(sk)) = true
functions: KDF/2, h/1                                            // Other primitives

section{* SIGMA but with a length extension *}

// Honest parties can register fresh keys
rule Register_pk_normal:
  [ Fr(~ltk) ]
  --[ LtkSet($A, ~ltk) ]->
  [ !Ltk($A, ~ltk), !Pk($A, pk(~ltk)), Out(pk(~ltk)) ]

rule Init_1:
  let ekI = h(<'ephemeral_key', ~coinsI>)
      epkI = 'g' ^ ekI
      m1 = epkI
  in
  [ Fr(~coinsI), !Ltk($I, ~ltkI) ]
  --[ SidI_1(~coinsI, $I, $R, epkI), Neq($I, $R) ]->
  [ Init_1(~coinsI, $I, $R, ~ltkI, ekI ), !Random(~coinsI), Out(m1) ]

rule Resp_1:
  let ekR = h(<'ephemeral_key', ~coinsR>)
      m1 = epkI
      epkR = 'g' ^ ekR
      sigR = sign(<'R', epkI, epkR>, ~ltkR)
      pms = epkI ^ ekR
      Km = KDF('Km', pms)
      macR = h(<Km, $R>)
      m2 = <epkR, sigR, macR>
      sessionkey = KDF('sk', pms)
  in
    [ !Ltk($R, ~ltkR)
    , Fr(~coinsR)
    , In(m1)
    ]
  --[ SidR_1(~coinsR, $I, $R, epkI, epkR, pms)
    , Neq($I, $R)
    , RCommit(sessionkey, $R, $I)
    ]->
    [ Resp_1(~coinsR, $I, $R, ekR, epkI, pms)
    , !Random(~coinsR)
    , Out(m2)
    ]

rule Init_2:
  let epkR = 'g'^ekR
      peerTid = ekR
      m2 = <epkR, sigR, macR>
      epkI = 'g'^ekI
      sigI = sign(<'I', epkR, epkI>, ~ltkI)
      pms = epkR^ekI
      Km = KDF('Km', pms)
      macI = h(<Km, $I>)
      m3 = <sigI, macI>
      sessionkey = KDF('sk', pms)
  in
    [ Init_1(~coinsI, $I, $R, ~ltkI, ekI )
    , !Pk($R, pkR)
    , In(m2)
    ]
  --[ SidI_2(~coinsI, $I, $R, epkI, epkR, pms)
    , IAccepts(sessionkey, $I, $R)
    , Eq(verify(sigR, <'R', epkI, epkR>, pkR), true)
    , Eq(macR, h(<Km, $R>))
    ]->
    [ Out(m3),
      !SessionKey(~coinsI, $I, $R, sessionkey)
    ]

rule Resp_2:
  let epkR = 'g' ^ ekR
      epkI = 'g' ^ peerTid
      m3 = <sigI, macI>
      Km = KDF('Km', pms)
      sessionkey = KDF('sk', pms)
  in
    [ !Pk($I, pkI)
    , Resp_1(~coinsR, $I, $R, ekR, epkI, pms)
    , In(m3)
    ]
  --[ SidR_2(~coinsR, $I, $R, epkI, epkR, pms)
    , RAccepts(sessionkey, $R, $I)
    , Eq(verify(sigI, <'I', epkR, epkI>, pkI), true)
    , Eq(macI, h(<Km, $I>))
     ]->
    [ !SessionKey(~coinsR, $I, $R, sessionkey) ]


// ------------- ADVERSARY POWERS    -------------

// Attacker can learn any session key (rendering the session non-fresh)
rule SesskRev:
    [ !SessionKey(~tid, $I, $R, sessionkey) ]
  --[ SesskRev(~tid) ]->
    [ Out(sessionkey) ]

rule LtkRev:
     [ !Ltk($X, ltkX) ]
     --[ LtkRev($X) ]->
     [ Out(ltkX) ]

/* Length extension: instead of fiddling around with the type of various
   terms, we just consider appending data to change the second element of a
   pair. */
rule RatherAbstractedLengthExtension:
    [ In(<h(<X, Y>), Z>) ]
    --[ Vwoooop() ]->
    [ Out(h(<X, Z>)) ]

// ------------- SECURITY PROPERTIES -------------

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

restriction inequalities_hold:
      "All x y #i. Neq(x, y) @ i ==> not(x = y)"

lemma sane: exists-trace
  "Ex tidI tidR I R epkI epkR sessionkey #i #j.
      SidI_2(tidI, I, R, epkI, epkR, sessionkey) @ i
    & SidR_2(tidR, I, R, epkI, epkR, sessionkey) @ j
    & not(tidI = tidR)
    & not(Ex #k. Vwoooop() @ k)
  "

lemma responder_key_agreement_without_length_extension:
  /* If the responder R accepts a key to use with the initiator I then... */
  "All sessionkey R I #i. RAccepts(sessionkey, R, I) @ i ==>

  /* either I previously accepted the same key to use with R */
    (Ex #j. #j < #i & IAccepts(sessionkey, I, R) @ j)

  /* or I's key was compromised beforehand */
    | (Ex #j. #j < #i & LtkRev(I) @ j)

  /* or there was a length extension */
    | (Ex #j. Vwoooop() @ j)"

/* Same as the above, but now allow length extensions */
lemma responder_key_agreement_with_length_extension:
  "All sessionkey R I #i. RAccepts(sessionkey, R, I) @ i ==>
    (Ex #j. #j < #i & IAccepts(sessionkey, I, R) @ j)
    | (Ex #j. #j < #i & LtkRev(I) @ j)"


/* The property doesn't hold for the initiator even without length extensions,
because SIGMA doesn't include the recipient's identity in the MAC */
lemma initiator_key_agreement_without_length_extension:
  "All sessionkey R I #i. IAccepts(sessionkey, R, I) @ i ==>
    (Ex #j. #j < #i & RCommit(sessionkey, I, R) @ j)
    | (Ex #j. #j < #i & LtkRev(I) @ j)
    | (Ex #j. Vwoooop() @ j)"

lemma initiator_key_agreement_with_length_extension:
  "All sessionkey R I #i. IAccepts(sessionkey, R, I) @ i ==>
    (Ex #j. #j < #i & RCommit(sessionkey, I, R) @ j)
    | (Ex #j. #j < #i & LtkRev(I) @ j)"

end
```

B.4 SIGMA with a negotiation protocol

In the following, we give an m4 preprocessor specification for a spthy file. This is so that we can write the rule ServerChoose once as a macro, and then instantiate it twice: once to allow the server to choose the generator g if it is present, and once to allow the server to choose the generator h if g is not offered.

```
/*
 * Protocol:  Simple DH group negotiation
 * Modeler:   Katriel Cohn-Gordon
 * Date:  April 2018
 * Status:  working

dnl // Tamarin uses ’ ’ which is an m4 close quote. So use <! !> for quoting instead
changequote(<!,!>)
changecom(<!@,@!>)

 */

theory dh_negotiation
begin

section{* DH version negotiation *}

builtins: diffie-hellman
functions: HMAC/3, KDF/1

// group generators (one good, one bad)
functions: g/0, h/0

// signatures
functions: sign/2, verify/3, pk/1, true/0
equations: verify(sign(m, sk), m, pk(sk)) = true

rule WeakDH:
     [ Exponential(h, x) ] --[ Dlog(x) ]-> [ Out(x) ]

rule Setup:
     [ Fr(~ltkC), Fr(~ltkS) ]
     --[ Setup($C, $S, ~ltkC, ~ltkS) ]->
     [ !KeysForClient(~ltkC, ~ltkS, $C, $S), !KeysForServer(~ltkC, ~ltkS, $C, $S) ]

rule Client1_gh:
     let m1 = <g, h> in
     [ Fr(~x), !KeysForClient(~ltkC, ~ltkS, $C, $S) ]
     -->
     [ Out(m1), Client1(~x, $C, $S, ~ltkC, ~ltkS, <m1>) ]

define(ServerChoose, <!rule Server_case_$1:
     let m1 = <a, b>
         m2 = $2 in
     [ In(m1), Fr(~y), !KeysForServer(~ltkC, ~ltkS, $C, $S) ]
     --[    PairContains($2, a, b),
         NotPairContains($3, a, b) ]->
     [ Out(m2), Server1(~y, $C, $S, ~ltkC, ~ltkS, <m1, m2>) ]!>)

ServerChoose(just_choose_g, g, 'potato')
ServerChoose(choose_h_if_no_g, h, g)

/* This rule would let the server choose h even if g were present. We leave
it out, so that in honest operation the client will never complete a
session using h. */
/* ServerChoose(just_choose_h, h, 'potato') */

rule Client2:
     let base = m2
         m3 = base^~x in
     [ Client1(~x, $C, $S, ~ltkC, ~ltkS, <m1>), In(m2) ]
     --[ SentM3(m3), PairContains(base, g, h) ]->
     [ Out(m3)
     , Client2(~x, $C, $S, ~ltkC, ~ltkS, base, <m1, m2, m3>)
     , Exponential(base, ~x) ]

rule Server2:
     let clientexp = m3
         base = m2
         serverexp = base^~y
         sessk = KDF(m3 ^ ~y)
         transcript = <m1, m2, m3, serverexp>
   signature = sign(<'1', clientexp, serverexp>, ~ltkS)
         mac = HMAC('server_mac', transcript, sessk)
   m4 = <serverexp, signature, mac>
in
     [ Server1(~y, $C, $S, ~ltkC, ~ltkS, <m1, m2>), In(m3) ]
     -->
     [ Out(m4)
     , Server2(~y, $C, $S, ~ltkC, ~ltkS, clientexp, serverexp, transcript, sessk)
     , Exponential(base, ~y) ]

rule Client3:
     let m4 = <serverexp, serversignature, servermac>
         clientexp = m3
         sessionkey = KDF(serverexp^~x)
         transcript = <m1, m2, m3, serverexp>
   clientmac = HMAC('client_mac', transcript, sessionkey)
   clientsignature = sign(<'2', clientexp, serverexp>, ~ltkC)
   m5 = <clientsignature, clientmac>
   in
     [ In(m4), Client2(~x, $C, $S, ~ltkC, ~ltkS, base, <m1, m2, m3>) ]
     --[ ClientAccepted(sessionkey)
       , ClientUsedBase(base)
       , Eq(servermac, HMAC('server_mac', transcript, sessionkey))
       , Eq(true, verify(serversignature, <'1', clientexp, serverexp>, pk(~ltkS)))
     ]->
     [ Out(m5) ]

rule Server3:
     let m5 = <clientsignature, clientmac>
   in
     [ Server2(~y, $C, $S, ~ltkC, ~ltkS, clientexp, serverexp, transcript, sessk)
     , In(m5) ]
     --[ ServerAccepted(sessk)
       , Eq(clientmac, HMAC('client_mac', transcript, sessk))
       , Eq(true, verify(clientsignature, <'2', clientexp, serverexp>, pk(~ltkC)))
     ]->
     [  ]

restriction    PairContains_ok: "All x a b #i.    PairContains(x, a, b) @ i
                ==>    ((x = a) | (x = b))"
restriction NotPairContains_ok: "All x a b #i. NotPairContains(x, a, b) @ i
                ==> not((x = a) | (x = b))"
restriction Equalities: "All x y #i. Eq(x, y) @ i ==> x = y"

lemma e: exists-trace "Ex k #j. ServerAccepted(k) @ j"

lemma executable_with_g: exists-trace
    "Ex k #i #j. ClientAccepted(k) @ i
             & ServerAccepted(k) @ j
         & ClientUsedBase(g) @ i"

lemma not_executable_with_h_without_dlog:
    "All k #i. ClientAccepted(k) @ i & not(Ex x #j. Dlog(x) @ j)
     ==> ClientUsedBase(g) @ i"

lemma dlog_lets_h_be_used:
    "All k #i. ClientAccepted(k) @ i ==> ClientUsedBase(g) @ i"
```

```

lemma client_key_secrecy_without_dlog:
    "All k #i. ClientAccepted(k) @ i & not(Ex x #j. Dlog(x) @ j)
         ==> not(Ex #j. KU(k) @ j)"

lemma client_key_secrecy_with_dlog:
    "All k #i. ClientAccepted(k) @ i ==> not(Ex #j. KU(k) @ j)"

end
```

B.5 KRACK

As before, we give an m4 preprocessor specification for a spthy file.

```
/*
 * Protocol:  WPA with KRACK attacks
 * Modeler:   Katriel Cohn-Gordon
 * Date:  April 2018
 * Status:  working
 */

dnl // Tamarin uses '' which is an m4 close quote. So use <! !> for quoting instead.
changequote(<!,!>)
changecom(<!/*!>, <!*/!>)

dnl Cas m4 macro wizardry
include(at_most_of.m4i)

dnl define(<!Supp_PTK_START!>, F_Supp_PTK_START($@))
theory wpa_bounded
begin

functions: KDF/1, snenc/3, sndec/2
builtins: multiset, symmetric-encryption
equations: sndec(snenc(message, key, nonce), key) = message

at_most_of(1, SupplicantStarts, 3)
at_most_of(1, AuthenticatorStarts, 3)
dnl at_most_of(1, SupplicantRereceivesM1, 3)
at_most_of(1, SupplicantRereceivesM3, 3)
at_most_of(3, EncryptedWith, 1)
at_most_of(1, Init, 1)

restriction Neq:
    "All x y #i. Neq(x, y) @ i ==> not(x = y)"

// Setup

rule Begin:
     [ Fr(~PMK), Fr(~suppID), Fr(~authID) ]
     --[ Init(~PMK), Setup(~suppID, ~PMK), Setup(~authID, ~PMK) ]->
     [ Supp_PTK_INIT(~suppID, ~PMK, '0'), Auth_PTK_INIT(~suppID, ~PMK, '0') ]


// State machine for the supplicant

rule Supp_recv_m1:
     let m1 = <ctr, ANonce>
         m2 = <ctr, ~SNonce>
     in
     [ Supp_PTK_INIT(~suppID, ~PMK, ctr), Fr(~SNonce), In(m1) ]
     --[ SupplicantStarts(~suppID, ~PMK, ctr)
       , SupplicantReceivesM1(~suppID, ~PMK, ctr) ]->
     [ Supp_PTK_START(~suppID, ~PMK, ctr, ANonce, ~SNonce), Out(m2) ]

rule Supp_rerecv_m1:
     let m1 = <ctr, ANonce>
         m2 = <ctr, ~SNonce>
     in
     [ Supp_PTK_START(~suppID, ~PMK, ctr, blank_1, blank_2)
     , Fr(~SNonce)
     , In(m1) ] // TODO does it use the same SNonce or a new one?
     --[ SupplicantRereceivesM1(~suppID, ~PMK, ctr) ]->
     [ Supp_PTK_START(~suppID, ~PMK, ctr, ANonce, ~SNonce), Out(m2) ]

rule Supp_recv_m3: // TODO MIC-Verified && !ReplayedMsg
     let PTK = KDF(<~PMK, ANonce, SNonce>)
         ctr_plus_1 = ctr + '0'
     m3 = <ctr_plus_1, senc(<'handshake', GTK>, PTK)>
     m4 = ctr_plus_1
     in
     [ Supp_PTK_START(~suppID, ~PMK, ctr, ANonce, SNonce), In(m3) ]
     --[ SupplicantReceivesM3(~suppID, ~PMK, ctr) ]->
     [ Supp_PTK_NEGOTIATING(~suppID, ~PMK, PTK, GTK, ANonce, SNonce, ctr_plus_1)
     , Out(m4) ]

rule Supp_negotiate:
     [ Supp_PTK_NEGOTIATING(~suppID, ~PMK, PTK, GTK, ANonce, SNonce, ctr), Fr(~kid) ]
     --[ SupplicantInstalled(~suppID, ~PMK, GTK, ctr) ]->
     [ Supp_PTK_DONE(~suppID, ~PMK, PTK, ANonce, SNonce, ctr)
     , EncryptionKey(~kid, PTK, ctr) ]

rule Supp_Infiniloop:
     [ Supp_PTK_DONE(~suppID, ~PMK, PTK, ANonce, SNonce, ctr) ]
     --[ Loop(~suppID) ]->
     [ Supp_PTK_INIT(~suppID, ~PMK, ctr) ]

// These retransmissions use an incremented EAPOL replay counter.
rule Supp_rerecv_m3: // TODO MIC-Verified && !ReplayedMsg
     let ctr_plus_1 = ctr + '0'
         m3 = <ctr_plus_1, senc(<'handshake', GTK>, PTK)>
         m4 = ctr_plus_1
     in
     [ Supp_PTK_DONE(~suppID, ~PMK, PTK, ANonce, SNonce, ctr), In(m3) ]
     --[ SupplicantRereceivesM3(~suppID, ~PMK, ctr_plus_1) ]->
     [ Supp_PTK_NEGOTIATING(~suppID, ~PMK, PTK, GTK, ANonce, SNonce, ctr_plus_1)
     , Out(m4) ]



// State machine for the authenticator

rule Authenticator_1:
     let m1 = <ctr, ~ANonce>
     in
     [ Auth_PTK_INIT(~authID, ~PMK, ctr), Fr(~ANonce) ]
     --[ AuthenticatorStarts(~authID, ~PMK, ctr), NotAPMK(~ANonce) ]->
     [ Auth_PTK_START(~authID, ~PMK, ctr, ~ANonce), Out(m1) ]

rule Authenticator_2:
     let PTK = KDF(<~PMK, ANonce, SNonce>)
         ctr_plus_1 = ctr + '0'
         m2 = <ctr, SNonce>
         m3 = <ctr_plus_1, senc(<'handshake', ~GTK>, PTK)>
     in
     [ Auth_PTK_START(~authID, ~PMK, ctr, ANonce), Fr(~GTK), In(m2) ]
     --[ NotAPMK(~GTK), HonestlyUsed(PTK) ]->
     [ Auth_PTK_NEGOTIATING(~authID, ~PMK, ctr_plus_1, ~GTK), Out(m3) ]

rule Authenticator_3:
     let foo = '1'
         m4 = ctr
     in
     [ Auth_PTK_NEGOTIATING(~authID, ~PMK, ctr, ~GTK), In(m4) ]
     --[ AuthenticatorInstalled(~authID, ~PMK, ~GTK, ctr) ]->
     [ Auth_PTK_DONE(~authID, ~PMK, ctr) ]

rule Auth_Infiniloop:
     [ Auth_PTK_DONE(~authID, ~PMK, ctr) ] --> [ Auth_PTK_INIT(~authID, ~PMK, ctr) ]


// encryption layer
rule Encrypt:
     [ EncryptionKey(~id, key, nonce) ]
     --[ EncryptedWith(~id) ]->
     [ Out(snenc(<'data', $message>, key, nonce))
     , EncryptionKey(~id, key, nonce + '0') ]


rule NonceReuse:
     let m1 = snenc(x1, key, nonce)
         m2 = snenc(x2, key, nonce)
     in
     [ In(<m1, m2>) ]
     --[ Neq(x1, x2), Kaboom(key) ]->
     [ Out(key) ]


//// Lemmas

lemma you_have_to_receive_an_initial_m1_sometime[sources]:
    "All suppID PMK ctr #i. (SupplicantRereceivesM1(suppID, PMK, ctr) @ i) ==> (
         Ex #j prectr. #j < #i
       & SupplicantReceivesM1(suppID, PMK, prectr) @ j
       & ((Ex x. ctr = prectr + x) | ctr = prectr)
    )"

lemma you_have_to_receive_an_initial_m3_sometime[sources]:
    "All suppID PMK ctr #i. SupplicantRereceivesM3(suppID, PMK, ctr) @ i ==> (
         Ex #j prectr. #j < #i
       & SupplicantReceivesM3(suppID, PMK, prectr) @ j
       & ((Ex x. ctr = prectr + x) | ctr = prectr)
    )"

lemma nonce_reuse_aint_magic[sources]:
      "All key #i. Kaboom(key) @ i ==> (
      // The adversary knew key beforehand
      (Ex #j. #j < #i & KU(key) @ j) |

      // key was used honestly before
      (Ex #j. #j < #i & HonestlyUsed(key) @ j)
      )"

lemma can_setup_key: exists-trace
    "Ex suppID authID PMK GTK ctr #i #j.
          AuthenticatorInstalled(authID, PMK, GTK, ctr) @ i
        & SupplicantInstalled(suppID, PMK, GTK, ctr) @ j"

lemma counters_start_at_zero_auth[reuse,use_induction]:
    "All authID PMK GTK ctr #i. AuthenticatorInstalled(authID, PMK, GTK, ctr) @ i
     ==> Ex #j. #j < #i & Setup(authID, PMK) @ j"

lemma pmks_are_secret[reuse]:
      "All id PMK #i. Setup(id, PMK) @ i ==> not(Ex #j. K(PMK) @ j)"

lemma gtk_secret_auth_without_extension:
    "All authID PMK GTK ctr #i. AuthenticatorInstalled(authID, PMK, GTK, ctr) @ i
     ==> (Ex key #j. Kaboom(key) @ j) | not(Ex #j. K(GTK) @ j)"

lemma gtk_secret_supp_without_extension:
    "All suppID PMK GTK ctr #i. SupplicantInstalled(suppID, PMK, GTK, ctr) @ i
     ==> (Ex key #j. Kaboom(key) @ j) | not(Ex #j. K(GTK) @ j)"

lemma gtk_secret_auth_with_extension:
    "All authID PMK GTK ctr #i. AuthenticatorInstalled(authID, PMK, GTK, ctr) @ i
     ==> not(Ex #j. K(GTK) @ j)"

lemma gtk_secret_supp_with_extension:
    "All suppID PMK GTK ctr #i. SupplicantInstalled(suppID, PMK, GTK, ctr) @ i
     ==> not(Ex #j. K(GTK) @ j)"
end
```

APPENDIX C

TERMS AND ABBREVIATIONS

0-RT Zero Round-Trip

ACCE authenticated and confidential channel establishment

AEAD Authenticated Encryption with Associated Data

AES-GCM AES in Galois/Counter Mode

AIM AOL Instant Messenger

AKC Actor Key Compromise

AKE Authenticated Key Exchange

ARPANET Advanced Research Projects Agency Network

ART Asynchronous Ratcheting Trees

B-R Bellare-Rogaway

CDH Computational Diffie–Hellman

CTSS Compatible Time-Sharing System

DDH Decisional Diffie–Hellman

DH Diffie–Hellman

DSA Digital Signature Algorithm

DSKS Duplicate Signature Key Selection

ECDSA Elliptic Curve Digital Signature Algorithm

eCK Extended Canetti-Krawczyk

EUF-CMA Existential Unforgeability under an Adaptive Chosen Message Attack

FDH Full Domain Hash

GDH Gap Diffie–Hellman

HIBE Hierarchical Identity-Based Encryption

HKDF HMAC-based Key Derivation Function

HMAC Keyed-Hash Message Authentication Code

IETF Internet Engineering Task Force

IND-CCA Indistinguishability under an Adaptive Chosen-Ciphertext Attack

iO indistinguishability obfuscation

KCI Key Compromise Impersonation

KDF Key Derivation Function

KEM Key Encapsulation Mechanism

KRACK Key Reinstallation Attack
MAC Message Authentication Code
MD Merkle-Damgård
MIT Massachusetts Institute of Technology
MLS Messaging Layer Security
mpOTR Multi-Party Off-the-Record Messaging
OMEMO OMEMO Multi-End Message and Object Encryption
OTR Off-the-Record Messaging
PCS Post-Compromise Security
PFS Perfect Forward Secrecy
PGP Pretty Good Privacy
PKI Public Key Infrastructure
ppt Probabilistic Polynomial-time
pptm Probabilistic Polynomial-time Turing Machine
PRF Pseudorandom Function
PRF-ODH Pseudorandom Function Oracle Diffie–Hellman
PSS Probabilistic Signature Scheme
QR Quick Response
QUIC Quick UDP Internet Connections
RFC Request For Comments
RNG Random Number Generator
ROM Random Oracle Model
RSA Rivest-Shamir-Adleman
s/MIME Secure/Multipurpose Internet Mail Extensions
SCIMP Silent Circle Instant Messaging Protocol
SMTP Simple Mail Transfer Protocol
spthy security protocol theory
STS Station-to-Station
TLS Transport Layer Security
UC Universal Composability
UKS Unknown Key Share
WPA Wi-Fi Protected Access
XMPP Extensible Messaging and Presence Protocol

APPENDIX D
REFERENCES

[1] Martín Abadi and Phillip Rogaway. 'Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption)'. In: *Journal of Cryptology* 15.2 (2002), pp. 103–127.

[2] Michel Abdalla, Céline Chevalier, Mark Manulis and David Pointcheval. 'Flexible Group Key Exchange with On-demand Computation of Subgroup Keys'. In: *AFRICACRYPT 10: 3rd International Conference on Cryptology in Africa.* Vol. 6055. Lecture Notes in Computer Science. Stellenbosch, South Africa: Springer, Heidelberg, Germany, 2010, pp. 351–368.

[3] Ruba Abu-Salma, M. Angela Sasse, Joseph Bonneau, Anastasia Danilova, Alena Naiakshina and Matthew Smith. 'Obstacles to the Adoption of Secure Communication Tools'. In: *2017 IEEE Symposium on Security and Privacy.* San Jose, CA, USA: IEEE Computer Society Press, 2017, pp. 137–153.

[4] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella Béguelin and Paul Zimmermann. 'Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice'. In: *ACM CCS 15: 22nd Conference on Computer and Communications Security.* Denver, CO, USA: ACM Press, 2015, pp. 5–17.

[5] Michael Backes, Birgit Pfitzmann and Michael Waidner. 'A General Composition Theorem for Secure Reactive Systems'. In: *TCC 2004: 1st Theory of Cryptography Conference.* Vol. 2951. Lecture Notes in Computer Science. Cambridge, MA, USA: Springer, Heidelberg, Germany, 2004, pp. 336–354.

[6] Christoph Bader, Tibor Jager, Yong Li and Sven Schäge. 'On the Impossibility of Tight Cryptographic Reductions'. In: *Advances in Cryptology – EUROCRYPT 2016, Part II.* Vol. 9666. Lecture Notes in Computer Science. Vienna, Austria: Springer, Heidelberg, Germany, 2016, pp. 273–304.

[7] Christoph Bader, Dennis Hofheinz, Tibor Jager, Eike Kiltz and Yong Li. 'Tightly-Secure Authenticated Key Exchange'. In: *TCC 2015: 12th Theory of Cryptography Conference, Part I*. Vol. 9014. Lecture Notes in Computer Science. Warsaw, Poland: Springer, Heidelberg, Germany, 2015, pp. 629–658.

[8] Gergei Bana, Pedro Adão and Hideki Sakurada. 'Computationally Complete Symbolic Attacker in Action'. In: *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*. Vol. 18. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012, pp. 546–560. URL: https://doi.org/10.4230/LIPIcs.FSTTCS.2012.546.

[9] Gergei Bana and Hubert Comon-Lundh. 'A Computationally Complete Symbolic Attacker for Equivalence Properties'. In: *ACM CCS 14: 21st Conference on Computer and Communications Security*. Scottsdale, AZ, USA: ACM Press, 2014, pp. 609–620.

[10] Gergei Bana and Hubert Comon-Lundh. *Towards Unconditional Soundness: Computationally Complete Symbolic Attacker*. Cryptology ePrint Archive, Report 2012/019. http://eprint.iacr.org/2012/019. 2012.

[11] Gergei Bana and Hubert Comon-Lundh. 'Towards Unconditional Soundness: Computationally Complete Symbolic Attacker'. In: *Principles of Security and Trust - First International Conference, POST 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012, Proceedings*. Vol. 7215. Lecture Notes in Computer Science. Springer, 2012, pp. 189–208. URL: https://doi.org/10.1007/978-3-642-28641-4_11.

[12] Gergei Bana, Koji Hasebe and Mitsuhiro Okada. 'Computationally complete symbolic attacker and key exchange'. In: *ACM CCS 13: 20th Conference on Computer and Communications Security*. Berlin, Germany: ACM Press, 2013, pp. 1231–1246.

[13] Richard Barnes, Jon Millican, Emad Omara, Katriel Cohn-Gordon and Raphael Robert. *The Messaging Layer Security Protocol*. Internet-Draft draft-barnes-mls-protocol-00. IETF Secretariat, Feb. 2018. URL: https://www.ietf.org/internet-drafts/draft-barnes-mls-protocol-00.txt.

[14] Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt and Pierre-Yves Strub. 'EasyCrypt: A Tutorial'. In: *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures*. Vol. 8604. Lecture Notes in Computer Science. Springer, 2013, pp. 146–166. URL: https://doi.org/10.1007/978-3-319-10082-1_6.

[15] Gilles Barthe, Juan Manuel Crespo, Yassine Lakhnech and Benedikt Schmidt. 'Mind the Gap: Modular Machine-Checked Proofs of One-Round Key Exchange Protocols'. In: *Advances in Cryptology – EUROCRYPT 2015, Part II*. Vol. 9057. Lecture Notes in Computer Science. Sofia, Bulgaria: Springer, Heidelberg, Germany, 2015, pp. 689–718.

[16] David A. Basin, Cas J. F. Cremers and Marko Horvat. ‘Actor Key Compromise: Consequences and Countermeasures’. In: *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*. IEEE Computer Society, 2014, pp. 244–258. URL: https://doi.org/10.1109/CSF.2014.25.

[17] David A. Basin, Cas J. F. Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse and Pawel Szalachowski. ‘ARPKI: Attack Resilient Public-Key Infrastructure’. In: *ACM CCS 14: 21st Conference on Computer and Communications Security*. Scottsdale, AZ, USA: ACM Press, 2014, pp. 382–393.

[18] David A. Basin, Cas Cremers, Jannik Dreier and Ralf Sasse. ‘Symbolically analyzing security protocols using tamarin’. In: *SIGLOG News* 4.4 (2017), pp. 19–30. URL: http://doi.acm.org/10.1145/3157831.3157835.

[19] Mihir Bellare. ‘Practice-Oriented Provable-Security (Invited Lecture)’. In: *ISW’97: 1st International Workshop on Information Security*. Vol. 1396. Lecture Notes in Computer Science. Tatsunokuchi, Japan: Springer, Heidelberg, Germany, 1998, pp. 221–231.

[20] Mihir Bellare, Ran Canetti and Hugo Krawczyk. ‘A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols (Extended Abstract)’. In: *30th Annual ACM Symposium on Theory of Computing*. Dallas, TX, USA: ACM Press, 1998, pp. 419–428.

[21] Mihir Bellare and Chanathip Namprempre. ‘Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm’. In: *Advances in Cryptology – ASIACRYPT 2000*. Vol. 1976. Lecture Notes in Computer Science. Kyoto, Japan: Springer, Heidelberg, Germany, 2000, pp. 531–545.

[22] Mihir Bellare, David Pointcheval and Phillip Rogaway. ‘Authenticated Key Exchange Secure against Dictionary Attacks’. In: *Advances in Cryptology – EUROCRYPT 2000*. Vol. 1807. Lecture Notes in Computer Science. Bruges, Belgium: Springer, Heidelberg, Germany, 2000, pp. 139–155.

[23] Mihir Bellare and Phillip Rogaway. ‘Entity Authentication and Key Distribution’. In: *Advances in Cryptology – CRYPTO’93*. Vol. 773. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1994, pp. 232–249.

[24] Mihir Bellare and Phillip Rogaway. ‘Provably Secure Session Key Distribution: The Three Party Case’. In: *27th Annual ACM Symposium on Theory of Computing*. Las Vegas, NV, USA: ACM Press, 1995, pp. 57–66.

[25] Mihir Bellare and Phillip Rogaway. ‘The Exact Security of Digital Signatures: How to Sign with RSA and Rabin’. In: *Advances in Cryptology – EUROCRYPT’96*. Vol. 1070. Lecture Notes in Computer Science. Saragossa, Spain: Springer, Heidelberg, Germany, 1996, pp. 399–416.

[26] Mihir Bellare and Bennet S. Yee. 'Forward-Security in Private-Key Cryptography'. In: *Topics in Cryptology – CT-RSA 2003*. Vol. 2612. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, 2003, pp. 1–18.

[27] 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, Part III*. Vol. 10403. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2017, pp. 619–650.

[28] Karthikeyan Bhargavan, Christina Brzuska, Cédric Fournet, Matthew Green, Markulf Kohlweiss and Santiago Zanella Béguelin. 'Downgrade Resilience in Key-Exchange Protocols'. In: *2016 IEEE Symposium on Security and Privacy*. San Jose, CA, USA: IEEE Computer Society Press, 2016, pp. 506–525.

[29] A.K. Bhushan, K.T. Pogran, R.S. Tomlinson and J.E. White. *Standardizing Network Mail Headers*. RFC 561. RFC Editor, Sept. 1973.

[30] Simon Blake-Wilson and Alfred Menezes. 'Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol'. In: *PKC'99: 2nd International Workshop on Theory and Practice in Public Key Cryptography*. Vol. 1560. Lecture Notes in Computer Science. Kamakura, Japan: Springer, Heidelberg, Germany, 1999, pp. 154–170.

[31] Bruno Blanchet. 'A Computationally Sound Mechanized Prover for Security Protocols'. In: *2006 IEEE Symposium on Security and Privacy*. Berkeley, CA, USA: IEEE Computer Society Press, 2006, pp. 140–154.

[32] Bruno Blanchet and David Pointcheval. 'Automated Security Proofs with Sequences of Games'. In: *Advances in Cryptology – CRYPTO 2006*. Vol. 4117. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2006, pp. 537–554.

[33] Florian Böhl and Dominique Unruh. 'Symbolic universal composability'. In: *Journal of Computer Security* 24.1 (2016), pp. 1–38. URL: https://doi.org/10.3233/JCS-140523.

[34] Jens-Matthias Bohli, Maria Isabel Gonzalez Vasco and Rainer Steinwandt. *Burmester-Desmedt Tree-Based Key Transport Revisited: Provable Security*. Cryptology ePrint Archive, Report 2005/360. http://eprint.iacr.org/2005/360. 2005.

[35] Dan Boneh and Alice Silverberg. 'Applications of multilinear forms to cryptography'. In: *Topics in Algebraic and Noncommutative Geometry: Proceedings in Memory of Ruth Michler*. Vol. 324. Contemporary Mathematics. American Mathematical Society, 2003.

[36] Dan Boneh and Mark Zhandry. 'Multiparty Key Exchange, Efficient Traitor Tracing, and More from Indistinguishability Obfuscation'. In: *Advances in Cryptology – CRYPTO 2014, Part I*. Vol. 8616. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2014, pp. 480–499.

[37] Nikita Borisov, Ian Goldberg and Eric A. Brewer. 'Off-the-record communication, or, why not to use PGP'. In: *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society, WPES 2004, Washington, DC, USA, October 28, 2004*. ACM, 2004, pp. 77–84. URL: http://doi.acm.org/10.1145/1029179.1029200.

[38] Colin Boyd and Anish Mathuria. *Protocols for Authentication and Key Establishment*. Information Security and Cryptography. Springer, Heidelberg, Germany, 2003. ISBN: 978-3-642-07716-6.

[39] Colin Boyd, Cas Cremers, Michele Feltz, Kenneth G. Paterson, Bertram Poettering and Douglas Stebila. 'ASICS: Authenticated Key Exchange Security Incorporating Certification Systems'. In: *ESORICS 2013: 18th European Symposium on Research in Computer Security*. Vol. 8134. Lecture Notes in Computer Science. Egham, UK: Springer, Heidelberg, Germany, 2013, pp. 381–399.

[40] Timo Brecher, Emmanuel Bresson and Mark Manulis. 'Fully Robust Tree-Diffie-Hellman Group Key Exchange'. In: *CANS 09: 8th International Conference on Cryptology and Network Security*. Vol. 5888. Lecture Notes in Computer Science. Kanazawa, Japan: Springer, Heidelberg, Germany, 2009, pp. 478–497.

[41] Jacqueline Brendel, Marc Fischlin, Felix Günther and Christian Janson. 'PRF-ODH: Relations, Instantiations, and Impossibility Results'. In: *Advances in Cryptology – CRYPTO 2017, Part III*. Vol. 10403. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2017, pp. 651–681.

[42] John Brooks. *Anonymous instant messaging for real privacy*. URL: https://ricochet.im/ (visited on 08/2018).

[43] Christina Brzuska, Marc Fischlin, Bogdan Warinschi and Stephen C. Williams. 'Composability of Bellare-Rogaway key exchange protocols'. In: *ACM CCS 11: 18th Conference on Computer and Communications Security*. Chicago, Illinois, USA: ACM Press, 2011, pp. 51–62.

[44] Christina Brzuska, Marc Fischlin, Nigel P. Smart, Bogdan Warinschi and Stephen C. Williams. 'Less is more: relaxed yet composable security notions for key exchange'. In: *Int. J. Inf. Sec.* 12.4 (2013), pp. 267–297. URL: https://doi.org/10.1007/s10207-013-0192-y.

[45] Mike Burmester and Yvo Desmedt. 'A Secure and Scalable Group Key Exchange System'. In: *Information Processing Letters* 94.3 (May 2005), pp. 137–143.

[46] Christian Cachin and Reto Strobl. 'Asynchronous group key exchange with failures'. In: *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing, PODC 2004, St. John's, Newfoundland, Canada, July 25-28, 2004*. ACM, 2004, pp. 357–366. URL: http://doi.acm.org/10.1145/1011767.1011820.

[47] Jon Callas, Lutz Donnerhacke, Hal Finney and Rodney Thayer. *OpenPGP Message Format*. RFC 2440. RFC Editor, Nov. 1998. URL: https://www.rfc-editor.org/rfc/rfc2440.txt.

[48] Ran Canetti and Marc Fischlin. 'Universally Composable Commitments'. In: *Advances in Cryptology – CRYPTO 2001*. Vol. 2139. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2001, pp. 19–40.

[49] Ran Canetti, Shai Halevi and Jonathan Katz. 'A Forward-Secure Public-Key Encryption Scheme'. In: *Advances in Cryptology – EUROCRYPT 2003*. Vol. 2656. Lecture Notes in Computer Science. Warsaw, Poland: Springer, Heidelberg, Germany, 2003, pp. 255–271.

[50] Ran Canetti and Hugo Krawczyk. 'Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels'. In: *Advances in Cryptology – EUROCRYPT 2001*. Vol. 2045. Lecture Notes in Computer Science. Innsbruck, Austria: Springer, Heidelberg, Germany, 2001, pp. 453–474.

[51] Ran Canetti and Hugo Krawczyk. 'Security Analysis of IKE's Signature-based Key-Exchange Protocol'. In: *Advances in Cryptology – CRYPTO 2002*. Vol. 2442. Lecture Notes in Computer Science. http://eprint.iacr.org/2002/120/. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2002, pp. 143–161.

[52] Ran Canetti and Tal Rabin. 'Universal Composition with Joint State'. In: *Advances in Cryptology – CRYPTO 2003*. Vol. 2729. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2003, pp. 265–281.

[53] Sanjit Chatterjee, Alfred Menezes and Palash Sarkar. 'Another Look at Tightness'. In: *SAC 2011: 18th Annual International Workshop on Selected Areas in Cryptography*. Vol. 7118. Lecture Notes in Computer Science. Toronto, Ontario, Canada: Springer, Heidelberg, Germany, 2012, pp. 293–319.

[54] Yi-Ruei Chen and Wen-Guey Tzeng. 'Group key management with efficient rekey mechanism: A Semi-Stateful approach for out-of-Synchronized members'. In: *Computer Communications* 98 (2017).

[55] Carl Chudyk. *Innovation*. Board game. 2010. URL: http://www.asmadigames.com.

[56] Katriel Cohn-Gordon and Cas Cremers. *Mind the Gap: Where Provable Security and Real-World Messaging Don't Quite Meet*. Cryptology ePrint Archive, Report 2017/982. http://eprint.iacr.org/2017/982. 2017.

[57] Katriel Cohn-Gordon, Cas J. F. Cremers and Luke Garratt. 'On Post-compromise Security'. In: *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*. IEEE Computer Society, 2016, pp. 164–178. URL: https://doi.org/10.1109/CSF.2016.19.

[58] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt and Douglas Stebila. *A Formal Security Analysis of the Signal Messaging Protocol*. Cryptology ePrint Archive, Report 2016/1013. http://eprint.iacr.org/2016/1013. 2016.

[59] Katriel Cohn-Gordon, Cas J. F. 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 2017, Paris, France, April 26-28, 2017*. IEEE, 2017, pp. 451–466. URL: https://doi.org/10.1109/EuroSP.2017.27.

[60] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican and Kevin Milner. *On Ends-to-Ends Encryption: Asynchronous Group Messaging with Strong Security Guarantees*. Cryptology ePrint Archive, Report 2017/666. http://eprint.iacr.org/2017/666. 2017.

[61] Hubert Comon-Lundh and Stéphanie Delaune. 'The Finite Variant Property: How to Get Rid of Some Algebraic Properties'. In: *Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings*. Vol. 3467. Lecture Notes in Computer Science. Springer, 2005, pp. 294–307. URL: https://doi.org/10.1007/978-3-540-32033-3_22.

[62] Jean-Sébastien Coron. 'Optimal Security Proofs for PSS and Other Signature Schemes'. In: *Advances in Cryptology – EUROCRYPT 2002*. Vol. 2332. Lecture Notes in Computer Science. Amsterdam, The Netherlands: Springer, Heidelberg, Germany, 2002, pp. 272–287.

[63] Véronique Cortier, Stéphanie Delaune and Pascal Lafourcade. 'A survey of algebraic properties used in cryptographic protocols'. In: *Journal of Computer Security* 14.1 (2006), pp. 1–43. URL: http://content.iospress.com/articles/journal-of-computer-security/jcs244.

[64] Cas Cremers. 'Examining indistinguishability-based security models for key exchange protocols: the case of CK, CK-HMQV, and eCK'. In: *ASIACCS 11: 6th ACM Symposium on Information, Computer and Communications Security*. Hong Kong, China: ACM Press, 2011, pp. 80–91.

[65] Cas Cremers. 'Symbolic security analysis using the Tamarin prover'. In: *2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2-6, 2017*. IEEE, 2017, p. 5. URL: https://doi.org/10.23919/FMCAD.2017.8102229.

[66] Cas Cremers and Michele Feltz. *One-round Strongly Secure Key Exchange with Perfect Forward Secrecy and Deniability*. Cryptology ePrint Archive, Report 2011/300. http://eprint.iacr.org/2011/300. 2011.

[67] Ivan Damgård. 'A "proof-reading" of Some Issues in Cryptography (Invited Lecture)'. In: *ICALP 2007: 34th International Colloquium on Automata, Languages and Programming*. Vol. 4596. Lecture Notes in Computer Science. Wroclaw, Poland: Springer, Heidelberg, Germany, 2007, pp. 2–11.

[68] Jean Paul Degabriele, Anja Lehmann, Kenneth G. Paterson, Nigel P. Smart and Mario Strefler. 'On the Joint Security of Encryption and Signature in EMV'. In: *Topics in Cryptology – CT-RSA 2012*. Vol. 7178. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, 2012, pp. 116–135.

[69] Mario Di Raimondo, Rosario Gennaro and Hugo Krawczyk. 'Deniable authentication and key exchange'. In: *ACM CCS 06: 13th Conference on Computer and Communications Security*. Alexandria, Virginia, USA: ACM Press, 2006, pp. 400–409.

[70] Whitfield Diffie, Paul C. van Oorschot and Michael J. Wiener. 'Authentication and Authenticated Key Exchanges'. In: *Designs, Codes and Cryptography* 2.2 (June 1992), pp. 107–125.

[71] Roger Dingledine, Nick Mathewson and Paul Syverson. 'Tor: The Second-generation Onion Router'. In: *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*. SSYM'04. San Diego, CA: USENIX Association, 2004, pp. 21–21. URL: http://dl.acm.org/citation.cfm?id=1251375.1251396.

[72] Danny Dolev and Andrew Chi-Chih Yao. 'On the Security of Public Key Protocols (Extended Abstract)'. In: *22nd Annual Symposium on Foundations of Computer Science*. Nashville, Tennessee: IEEE Computer Society Press, 1981, pp. 350–357.

[73] Benjamin Dowling, Marc Fischlin, Felix Günther and Douglas Stebila. 'A Cryptographic Analysis of the TLS 1.3 Handshake Protocol Candidates'. In: *ACM CCS 15: 22nd Conference on Computer and Communications Security*. Denver, CO, USA: ACM Press, 2015, pp. 1197–1210.

[74] Jannik Dreier, Charles Duménil, Steve Kremer and Ralf Sasse. 'Beyond Subterm-Convergent Equational Theories in Automated Verification of Stateful Protocols'. In: *Principles of Security and Trust - 6th International Conference, POST 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017,*

Proceedings. Vol. 10204. Lecture Notes in Computer Science. Springer, 2017, pp. 117–140. URL: https://doi.org/10.1007/978-3-662-54455-6_6.

[75] Thai Duong and Juliano Rizzo. *Flickr's API Signature Forgery Vulnerability*. Advisory. 2009. URL: https://netifera.com/research/flickr_api_signature_forgery.pdf (visited on 02/2018).

[76] Ratna Dutta and Rana Barua. 'Provably Secure Constant Round Contributory Group Key Agreement in Dynamic Setting'. In: *IEEE Trans. Information Theory* 54.5 (2008), pp. 2007–2025. URL: https://doi.org/10.1109/TIT.2008.920224.

[77] eQualit.ie. $(N+1)$*SEC*. 2016. URL: https://learn.equalit.ie/wiki/Np1sec.

[78] Facebook. *Messenger Secret Conversations*. Tech. rep. 2016. URL: https://fbnewsroomus.files.wordpress.com/2016/07/secret_conversations_whitepaper-1.pdf (visited on 07/2016).

[79] Michael Farb, Yue-Hsun Lin, Tiffany Hyun-Jin Kim, Jonathan M. McCune and Adrian Perrig. 'SafeSlinger: easy-to-use and secure public-key exchange'. In: *The 19th Annual International Conference on Mobile Computing and Networking, MobiCom'13, Miami, FL, USA, September 30 - October 04, 2013*. ACM, 2013, pp. 417–428. URL: http://doi.acm.org/10.1145/2500423.2500428.

[80] Michèle Feltz and Cas Cremers. *On the Limits of Authenticated Key Exchange Security with an Application to Bad Randomness*. Cryptology ePrint Archive, Report 2014/369. http://eprint.iacr.org/2014/369. 2014.

[81] Michèle Feltz and Cas Cremers. 'Strengthening the security of authenticated key exchange against bad randomness'. In: *Des. Codes Cryptography* 86.3 (2018), pp. 481–516. URL: https://doi.org/10.1007/s10623-017-0337-5.

[82] Marc Fischlin and Felix Günther. 'Multi-Stage Key Exchange and the Case of Google's QUIC Protocol'. In: *ACM CCS 14: 21st Conference on Computer and Communications Security*. Scottsdale, AZ, USA: ACM Press, 2014, pp. 1193–1204.

[83] Tilman Frosch, Christian Mainka, Christoph Bader, Florian Bergsma, Jörg Schwenk and Thorsten Holz. 'How Secure is TextSecure?' In: *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*. IEEE, 2016, pp. 457–472. URL: https://doi.org/10.1109/EuroSP.2016.41.

[84] Manisha Ganguly. *WhatsApp design feature means some encrypted messages could be read by third party*. 13th Jan. 2017. URL: https://www.theguardian.com/technology/2017/jan/13/whatsapp-design-feature-encrypted-messages.

[85] Christina Garman, Matthew Green, Gabriel Kaptchuk, Ian Miers and Michael Rushanan. ‘Dancing on the Lip of the Volcano: Chosen Ciphertext Attacks on Apple iMessage’. In: *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*. USENIX Association, 2016, pp. 655–672. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/garman.

[86] Ian Goldberg, Berkant Ustaoglu, Matthew Van Gundy and Hao Chen. ‘Multi-party off-the-record messaging’. In: *ACM CCS 09: 16th Conference on Computer and Communications Security*. Chicago, Illinois, USA: ACM Press, 2009, pp. 358–368.

[87] Oded Goldreich. *On Post-Modern Cryptography*. Cryptology ePrint Archive, Report 2006/461. http://eprint.iacr.org/2006/461. 2006.

[88] Oded Goldreich. ‘On the Foundations of Modern Cryptography (Invited Lecture)’. In: *Advances in Cryptology – CRYPTO’97*. Vol. 1294. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1997, pp. 46–74.

[89] Google. *Key Transparency*. 2017. URL: https://github.com/google/keytransparency.

[90] Matthew D. Green and Ian Miers. ‘Forward Secure Asynchronous Messaging from Puncturable Encryption’. In: *2015 IEEE Symposium on Security and Privacy*. San Jose, CA, USA: IEEE Computer Society Press, 2015, pp. 305–320.

[91] P. Hoffman. *SMTP Service Extension for Secure SMTP over Transport Layer Security*. RFC 3207. RFC Editor, Feb. 2002. URL: https://www.rfc-editor.org/rfc/rfc3207.txt.

[92] Dennis Hofheinz and Victor Shoup. ‘GNUC: A New Universal Composability Framework’. In: *Journal of Cryptology* 28.3 (July 2015), pp. 423–508.

[93] Jelle van den Hooff, David Lazar, Matei Zaharia and Nickolai Zeldovich. ‘Vuvuzela: Scalable Private Messaging Resistant to Traffic Analysis’. In: *Proceedings of the 25th Symposium on Operating Systems Principles*. SOSP ’15. Monterey, California: ACM, 2015, pp. 137–152. ISBN: 978-1-4503-3834-9. URL: http://doi.acm.org/10.1145/2815400.2815417.

[94] internet.org. *State of Connectivity 2015*. Annual Report. 2016. URL: https://fbnewsroomus.files.wordpress.com/2016/02/state-of-connectivity-2015-2016-02-21-final.pdf (visited on 05/2017).

[95] Frederic Jacobs. *On the "WhatsApp backdoor", Trade-Offs and Opportunistic Authentication*. 20th Jan. 2017. URL: https://www.fredericjacobs.com/blog/2017/01/20/whatsapp-backdoor/.

[96] Tibor Jager, Florian Kohlar, Sven Schäge and Jörg Schwenk. 'On the Security of TLS-DHE in the Standard Model'. In: *Advances in Cryptology – CRYPTO 2012*. Vol. 7417. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2012, pp. 273–293.

[97] Antoine Joux. 'A One Round Protocol for Tripartite Diffie-Hellman'. In: *Journal of Cryptology* 17.4 (Sept. 2004), pp. 263–276.

[98] Saqib A. Kakvi and Eike Kiltz. 'Optimal Security Proofs for Full Domain Hash, Revisited'. In: *Advances in Cryptology – EUROCRYPT 2012*. Vol. 7237. Lecture Notes in Computer Science. Cambridge, UK: Springer, Heidelberg, Germany, 2012, pp. 537–553.

[99] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014. ISBN: 9781466570269.

[100] Tiffany Hyun-Jin Kim, Lin-Shung Huang, Adrian Perrig, Collin Jackson and Virgil D. Gligor. 'Accountable key infrastructure (AKI): a proposal for a public-key validation infrastructure'. In: *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*. International World Wide Web Conferences Steering Committee / ACM, 2013, pp. 679–690. URL: http://dl.acm.org/citation.cfm?id=2488448.

[101] Yongdae Kim, Adrian Perrig and Gene Tsudik. 'Communication-Efficient Group Key Agreement'. In: *Trusted Information: The New Decade Challenge*. Springer US, 2001.

[102] Yongdae Kim, Adrian Perrig and Gene Tsudik. 'Simple and Fault-tolerant Key Agreement for Dynamic Collaborative Groups'. In: *Proceedings of the 7th ACM Conference on Computer and Communications Security*. CCS '00. ACM, 2000.

[103] Yongdae Kim, Adrian Perrig and Gene Tsudik. 'Tree-based group key agreement'. In: *ACM Trans. Inf. Syst. Secur.* 7.1 (2004), pp. 60–96. URL: http://doi.acm.org/10.1145/984334.984337.

[104] Kazukuni Kobara, SeongHan Shin and Mario Strefler. 'Partnership in key exchange protocols'. In: *ASIACCS 09: 4th ACM Symposium on Information, Computer and Communications Security*. Sydney, Australia: ACM Press, 2009, pp. 161–170.

[105] Nadim Kobeissi, Karthikeyan Bhargavan and Bruno Blanchet. 'Automated Verification for Secure Messaging Protocols and Their Implementations: A Symbolic and Computational Approach'. In: *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*. IEEE, 2017, pp. 435–450. URL: https://doi.org/10.1109/EuroSP.2017.38.

[106] Hugo Krawczyk. 'HMQV: A High-Performance Secure Diffie-Hellman Protocol'. In: *Advances in Cryptology – CRYPTO 2005*. Vol. 3621. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2005, pp. 546–566.

[107] Hugo Krawczyk. 'SIGMA: The "SIGn-and-MAc" Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols'. In: *Advances in Cryptology – CRYPTO 2003*. Vol. 2729. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2003, pp. 400–425.

[108] Hugo Krawczyk. 'The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?)' In: *Advances in Cryptology – CRYPTO 2001*. Vol. 2139. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2001, pp. 310–331.

[109] Caroline Kudla and Kenneth G. Paterson. 'Modular Security Proofs for Key Agreement Protocols'. In: *Advances in Cryptology – ASIACRYPT 2005*. Vol. 3788. Lecture Notes in Computer Science. Chennai, India: Springer, Heidelberg, Germany, 2005, pp. 549–565.

[110] Ralf Kuesters and Max Tuengerthal. *The IITM Model: a Simple and Expressive Model for Universal Composability*. Cryptology ePrint Archive, Report 2013/025. http://eprint.iacr.org/2013/025. 2013.

[111] Ralf Küsters and Daniel Rausch. 'A Framework for Universally Composable Diffie-Hellman Key Exchange'. In: *2017 IEEE Symposium on Security and Privacy*. San Jose, CA, USA: IEEE Computer Society Press, 2017, pp. 881–900.

[112] Brian A. LaMacchia, Kristin Lauter and Anton Mityagin. 'Stronger Security of Authenticated Key Exchange'. In: *ProvSec 2007: 1st International Conference on Provable Security*. Vol. 4784. Lecture Notes in Computer Science. Wollongong, Australia: Springer, Heidelberg, Germany, 2007, pp. 1–16.

[113] Adam Langley. *Pond*. 2014. URL: https://pond.imperialviolet.org/ (visited on 22/06/2015).

[114] B. Laurie, A. Langley and E. Kasper. *Certificate Transparency*. RFC 6962. RFC Editor, June 2013.

[115] Laurie Law, Alfred Menezes, Minghua Qu, Jerome A. Solinas and Scott A. Vanstone. 'An Efficient Protocol for Authenticated Key Agreement'. In: *Des. Codes Cryptography* 28.2 (2003), pp. 119–134.

[116] Sangwon Lee, Yongdae Kim, Kwangjo Kim and Dae-Hyun Ryu. 'An Efficient Tree-Based Group Key Agreement Using Bilinear Map'. In: *ACNS 03: 1st International Conference on Applied Cryptography and Network Security*. Vol. 2846. Lecture Notes in Computer Science. Kunming, China: Springer, Heidelberg, Germany, 2003, pp. 357–371.

[117] Anja Lehmann and Björn Tackmann. 'Updatable Encryption with Post-Compromise Security'. In: *IACR Cryptology ePrint Archive* 2018 (2018), p. 118. URL: http://eprint.iacr.org/2018/118.

[118] Yong Li and Sven Schäge. 'No-Match Attacks and Robust Partnering Definitions: Defining Trivial Attacks for Security Protocols is Not Trivial'. In: *ACM CCS 17: 24th Conference on Computer and Communications Security*. Dallas, TX, USA: ACM Press, 2017, pp. 1343–1360.

[119] libsignal-protocol-java. GitHub repository. 2016. URL: github.com/WhisperSystems/libsignal-protocol-java (visited on 07/2016).

[120] Joshua Lund. *Signal partners with Microsoft to bring end-to-end encryption to Skype*. Blog. 11th Jan. 2018. URL: https://signal.org/blog/skype-partnership/ (visited on 08/2018).

[121] Fermi Ma and Mark Zhandry. *Encryptor Combiners: A Unified Approach to Multiparty NIKE, (H)IBE, and Broadcast Encryption*. Cryptology ePrint Archive, Report 2017/152. 2017. URL: https://eprint.iacr.org/2017/152.

[122] Moxie Marlinspike. *Advanced cryptographic ratcheting*. Blog. 2013. URL: https://whispersystems.org/blog/advanced-ratcheting/ (visited on 07/2016).

[123] Moxie Marlinspike. *Forward Secrecy for Asynchronous Messages*. Blog. 22nd Aug. 2013. URL: https://whispersystems.org/blog/asynchronous-security/ (visited on 05/2017).

[124] Moxie Marlinspike. *Open Whisper Systems partners with Google on end-to-end encryption for Allo*. Blog. 2016. URL: https://whispersystems.org/blog/allo/ (visited on 07/2016).

[125] Moxie Marlinspike. *Signal Foundation*. Blog post. 21st Feb. 2018. URL: https://signal.org/blog/signal-foundation/ (visited on 02/2018).

[126] Ueli Maurer. 'Constructive Cryptography - A Primer (Invited Paper)'. In: *FC 2010: 14th International Conference on Financial Cryptography and Data Security*. Vol. 6052. Lecture Notes in Computer Science. Tenerife, Canary Islands, Spain: Springer, Heidelberg, Germany, 2010, p. 1.

[127] Ueli Maurer and Renato Renner. 'Abstract Cryptography'. In: *ICS 2011: 2nd Innovations in Computer Science*. Tsinghua University, Beijing, China: Tsinghua University Press, 2011, pp. 1–21.

[128] Simon Meier. 'Advancing automated security protocol verification'. PhD thesis. ETH Zürich, 2013. URL: https://www.research-collection.ethz.ch/handle/20.500.11850/66840 (visited on 04/2018).

[129] Simon Meier, Benedikt Schmidt, Cas Cremers and David A. Basin. 'The TAMARIN Prover for the Symbolic Analysis of Security Protocols'. In: *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*. Vol. 8044. Lecture Notes in Computer Science. Springer, 2013, pp. 696–701. URL: https://doi.org/10.1007/978-3-642-39799-8_48.

[130] 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, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*. USENIX Association, 2015, pp. 383–398. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/melara.

[131] Alfred Menezes and Berkant Ustaoglu. 'Comparing the Pre- and Post-specified Peer Models for Key Agreement'. In: *ACISP 08: 13th Australasian Conference on Information Security and Privacy*. Vol. 5107. Lecture Notes in Computer Science. Wollongong, Australia: Springer, Heidelberg, Germany, 2008, pp. 53–68.

[132] Alfred Menezes and Berkant Ustaoglu. 'On reusing ephemeral keys in Diffie-Hellman key agreement protocols'. In: *IJACT* 2.2 (2010), pp. 154–158. URL: https://doi.org/10.1504/IJACT.2010.038308.

[133] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone. *Handbook of Applied Cryptography*. Boca Raton, Florida: CRC Press, 1996.

[134] Ralph Charles Merkle. 'Secrecy, Authentication, and Public Key Systems.' PhD thesis. PhD thesis. 1979.

[135] Ghita Mezzour, Ahren Studer, Michael Farb, Jason Lee, Jonathan McCune, Hsu-Chun Hsiao and Adrian Perrig. *Ho-Po Key: Leveraging Physical Constraints on Human Motion to Authentically Exchange Information in a Group*. Tech. rep. Carnegie Mellon University, Dec. 2010.

[136] Kevin Milner, Cas Cremers, Jiangshan Yu and Mark Ryan. 'Automatically Detecting the Misuse of Secrets: Foundations, Design Principles, and Applications'. In: *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*. IEEE Computer Society, 2017, pp. 203–216. URL: https://doi.org/10.1109/CSF.2017.21.

[137] Vinnie Moscaritolo, Gary Belvin and Phil Zimmermann. *Silent Circle Instant Messaging Protocol Specification*. Tech. rep. Archived from the original. 5th Dec. 2012. URL: https://web.archive.org/web/20150402122917/https://silentcircle.com/sites/default/themes/silentcircle/assets/downloads/SCIMP_paper.pdf (visited on 07/2016).

[138] Jörn Müller-Quade and Dominique Unruh. 'Long-Term Security and Universal Composability'. In: *Journal of Cryptology* 23.4 (Oct. 2010), pp. 594–671.

[139] Roger M. Needham and Michael D. Schroeder. ‘Using encryption for authentication in large networks of computers’. In: *Communications of the Association for Computing Machinery* 21.21 (Dec. 1978), pp. 993–999.

[140] Tatsuaki Okamoto and David Pointcheval. ‘The Gap-Problems: A New Class of Problems for the Security of Cryptographic Schemes’. In: *PKC 2001: 4th International Workshop on Theory and Practice in Public Key Cryptography*. Vol. 1992. Lecture Notes in Computer Science. Cheju Island, South Korea: Springer, Heidelberg, Germany, 2001, pp. 104–118.

[141] Kenneth G. Paterson, Jacob C. N. Schuldt, Martijn Stam and Susan Thomson. ‘On the Joint Security of Encryption and Signature, Revisited’. In: *Advances in Cryptology – ASIACRYPT 2011*. Vol. 7073. Lecture Notes in Computer Science. Seoul, South Korea: Springer, Heidelberg, Germany, 2011, pp. 161–178.

[142] Trevor Perrin. *Double Ratchet Algorithm*. GitHub wiki. 2016. URL: https://github.com/trevp/double_ratchet/wiki (visited on 22/07/2016).

[143] Birgit Pfitzmann and Michael Waidner. ‘Composition and Integrity Preservation of Secure Reactive Systems’. In: *ACM CCS 00: 7th Conference on Computer and Communications Security*. Athens, Greece: ACM Press, 2000, pp. 245–254.

[144] Bertram Poettering and Paul Rösler. *Ratcheted key exchange, revisited*. Cryptology ePrint Archive, Report 2018/296. 2018. URL: https://eprint.iacr.org/2018/296.

[145] Manoj Prabhakaran and Amit Sahai. ‘New notions of security: achieving universal composability without trusted setup’. In: *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*. ACM, 2004, pp. 242–251. URL: http://doi.acm.org/10.1145/1007352.1007394.

[146] Mario Di Raimondo, Rosario Gennaro and Hugo Krawczyk. ‘Secure off-the-record messaging’. In: *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES 2005, Alexandria, VA, USA, November 7, 2005*. ACM, 2005, pp. 81–89. URL: http://doi.acm.org/10.1145/1102199.1102216.

[147] Blake Ramsdell. *S/MIME Version 3 Message Specification*. RFC 2633. RFC Editor, June 1999. URL: https://www.rfc-editor.org/rfc/rfc2633.txt.

[148] Joel Reardon, David A. Basin and Srdjan Capkun. ‘SoK: Secure Data Deletion’. In: *2013 IEEE Symposium on Security and Privacy*. Berkeley, CA, USA: IEEE Computer Society Press, 2013, pp. 301–315.

[149] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3 revision 23*. Internet-Draft draft-ietf-tls-tls13-23. 5th Jan. 2018. URL: https://tools.ietf.org/html/draft-ietf-tls-tls13-23.

[150] Phillip Rogaway. 'Authenticated-Encryption With Associated-Data'. In: *ACM CCS 02: 9th Conference on Computer and Communications Security*. Washington D.C., USA: ACM Press, 2002, pp. 98–107.

[151] Phillip Rogaway and Thomas Shrimpton. 'Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance'. In: *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*. Vol. 3017. Lecture Notes in Computer Science. Springer, 2004, pp. 371–388. URL: https://doi.org/10.1007/978-3-540-25937-4_24.

[152] Mike Rosulek. 'Universal Composability from Essentially Any Trusted Setup'. In: *Advances in Cryptology – CRYPTO 2012*. Vol. 7417. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2012, pp. 406–423.

[153] Mark Dermot Ryan. 'Enhanced Certificate Transparency and End-to-End Encrypted Mail'. In: *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, 2014. URL: https://www.ndss-symposium.org/ndss2014/enhanced-certificate-transparency-and-end-to-end-encrypted-mail.

[154] P. Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP): Core*. RFC 6120. RFC Editor, Mar. 2011. URL: https://www.rfc-editor.org/rfc/rfc6120.txt.

[155] Benedikt Schmidt, Simon Meier, Cas J. F. Cremers and David A. Basin. 'Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties'. In: *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*. IEEE Computer Society, 2012, pp. 78–94. URL: https://doi.org/10.1109/CSF.2012.25.

[156] Victor Shoup. *On Formal Models for Secure Key Exchange*. Tech. rep. RZ 3120. IBM, 1999.

[157] Victor Shoup. *Sequences of games: a tool for taming complexity in security proofs*. Cryptology ePrint Archive, Report 2004/332. http://eprint.iacr.org/2004/332. 2004.

[158] David G. Steer, Leo Strawczynski, Whitfield Diffie and Michael J. Wiener. 'A Secure Audio Teleconference System'. In: *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*. Vol. 403. Lecture Notes in Computer Science. Springer, 1988, pp. 520–528. URL: https://doi.org/10.1007/0-387-34799-2_37.

[159] Michael Steiner, Gene Tsudik and Michael Waidner. 'Key Agreement in Dynamic Peer Groups'. In: *IEEE Transactions on Parallel and Distributed Systems* 11.8 (Aug. 2000), pp. 769–780.

[160] Jacques Stern. 'Why Provable Security Matters? (Invited Talk)'. In: *Advances in Cryptology – EUROCRYPT 2003*. Vol. 2656. Lecture Notes in Computer Science. Warsaw, Poland: Springer, Heidelberg, Germany, 2003, pp. 449–461.

[161] The Tamarin Team. *Tamarin Prover Manual*. 2016. URL: https://tamarin-prover.github.io/manual/book/001_introduction.html (visited on 04/2018).

[162] Nik Unger and Ian Goldberg. 'Deniable Key Exchanges for Secure Messaging'. In: *ACM CCS 15: 22nd Conference on Computer and Communications Security*. Denver, CO, USA: ACM Press, 2015, pp. 1211–1223.

[163] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg and Matthew Smith. 'SoK: Secure Messaging'. In: *2015 IEEE Symposium on Security and Privacy*. San Jose, CA, USA: IEEE Computer Society Press, 2015, pp. 232–249.

[164] Mathy Vanhoef and Frank Piessens. 'Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2'. In: *ACM CCS 17: 24th Conference on Computer and Communications Security*. Dallas, TX, USA: ACM Press, 2017, pp. 1313–1328.

[165] D. Wallner, E. Harder and R. Agee. *Key Management for Multicast: Issues and Architectures*. RFC. United States, 1999.

[166] WhatsApp. *WhatsApp Encryption Overview*. Tech. rep. 2016. URL: https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf (visited on 07/2016).

[167] Douglas Wikström. 'Simplified Universal Composability Framework'. In: *TCC 2016-A: 13th Theory of Cryptography Conference, Part I*. Vol. 9562. Lecture Notes in Computer Science. Tel Aviv, Israel: Springer, Heidelberg, Germany, 2016, pp. 566–595.

[168] Chung Kei Wong, Mohamed Gouda and Simon S. Lam. 'Secure Group Communications Using Key Graphs'. In: *IEEE/ACM Transactions on Networking* 8.1 (Feb. 2000), pp. 16–30.

[169] Zheng Yang, Chao Liu, Wanping Liu, Daigu Zhang and Song Luo. 'A new strong security model for stateful authenticated group key exchange'. In: *International Journal of Information Security* (2017), pp. 1–18.

[170] Jiangshan Yu, Vincent Cheval and Mark Ryan. 'DTKI: A New Formalized PKI with Verifiable Trusted Parties'. In: *Comput. J.* 59.11 (2016), pp. 1695–1713. URL: https://doi.org/10.1093/comjnl/bxw039.

[171] Jiangshan Yu, Mark Ryan and Cas Cremers. ‘DECIM: Detecting Endpoint Compromise In Messaging’. In: *IEEE Trans. Information Forensics and Security* 13.1 (2018), pp. 106–118. URL: https://doi.org/10.1109/TIFS.2017.2738609.

Don't Panic