Realistic, Strong and Provable Key Exchange Security

Luke Garratt

Wolfson College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Trinity 2018

Abstract

Authenticated key exchange protocols are ubiquitous in modern-day life. They are used to secure numerous types of data exchange, ranging from online banking to instant messaging conversations. In this thesis, we extend the state of the art for authenticated key exchange security, following the three themes of realistic, strong and provable.

First, we tackle the theme of provable security. We develop generic techniques, applicable to a wide range of security models, to assist in proofs of security of cryptographic schemes. Specifically, a "game hopping" proof is a powerful technique to prove strong statements about the security of a protocol. However, the proof technique often involves esoteric and repetitive steps. We demystify the methodology and prove generic theorems to reduce the length and complexity of a wide class of game hopping proofs.

Next, we use this new framework to analyse strong and previously unstudied formal security guarantees of authenticated key exchange protocols. These protocols are able to achieve strong security guarantees because they share and update state across sessions. Using our framework, we are able to push the state of the art by defying the folklore of what was previously thought provable of authenticated key exchange protocols. Specifically, it was previously thought that no protocol can achieve a security guarantee about communication with a peer if the peer was already fully compromised. We show that this is not true by formally defining a new concept, which we name post-compromise security. We capture an intuitive "self-healing" property of protocols. We then construct and formally prove protocols can achieve this strong security notion.

Finally, we leverage all of this work to analyse realistic protocols. Signal, which is used to encrypt messages in WhatsApp, Facebook Messenger and Google Allo amongst others, is one such protocol we consider. Despite being used by billions of people, Signal was completely unstudied in the academic literature prior to this work. Two reasons for this were that Signal had no documentation at the time and it did not fit into existing models: it shares state across sessions and has over ten different types of key, most of which continually update. Despite its complexity, we are able to use the other work in this thesis to give the first ever analysis and proof of security of Signal. Overall, this thesis aims to bridge the gap between theory and practice in the current state of the art of authenticated key exchange protocols.

Realistic, Strong and Provable Key Exchange Security

Luke Garratt

Wolfson College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2018

Acknowledgements

This thesis would not be possible without the support of many people. First, I would like to express my utmost respect and gratitude to my supervisor, Professor Cas Cremers. Cas has been the ideal patient and intelligent supervisor. I am immensely grateful to him for being my mentor, for challenging me and believing in me. I have also been fortunate to have worked with some brilliant people during my DPhil. In the Oxford group, I learned a great deal when I shared authorship of papers with Katriel Cohn-Gordon and Kevin Milner. It was also a pleasure to work with the inimitable Douglas Stebila and Ben Dowling on our paper analysing the security guarantees of the Signal protocol. I would also like to thank Trevor Perrin from Open Whisper Systems for helpful discussions relating to this work. I am thankful I had the opportunity to work with Jon Millican from Facebook to follow up this work in the group chat context. Last but not least, I started my DPhil by reading and attempting to follow up work from Michèle Feltz on key exchange. It was a fantastic experience to be able to be a co-author on a paper analysing the limits of such protocols by the end of my DPhil.

I am thankful to Professor Andrew Simpson, who acted as my secondary supervisor during the first year of my doctorate. I would also like to extend my wider thanks to the whole Information Security Group at Oxford. Our whole group has worked well together as a team. Everyone was always willing to read each other's work or listen to a presentation before offering thoughtful advice and encouragement. The whole department worked extremely well together in research and organising events. I am grateful for everything they have done, from organising conferences and summer schools abroad, to organising BBQs and pub outings.

I will finish my time in Oxford not only with a master's degree and a doctorate but also some of the best memories of my life. Wolfson College has been a home to me for four years. It has been a wonderful community and I will miss living there very much. I have had countless fun times with my group, so thank you to Claudia, Claudio, Corina, Lydia, Maurits and Nina for making my time at Oxford uniquely hilarious. I am also privileged to have some amazing friends from all over the world: Aaron, Adéla, Amy, Bharath, Harriet, Maysa, Nick, Pedro, Sisi, Vicky and others. We have shared some great times and many laughs together, and I am truly thankful.

Finally, I would like to thank my parents, David and Mandy, for their constant love and support throughout my life. I have been an extremely difficult son, and I have probably caused them a great deal of stress over the years with my shenanigans, but I hope now they can finally relax.

Abstract

Authenticated key exchange protocols are ubiquitous in modern-day life. They are used to secure numerous types of data exchange, ranging from online banking to instant messaging conversations. In this thesis, we extend the state of the art for authenticated key exchange security, following the three themes of realistic, strong and provable.

First, we tackle the theme of provable security. We develop generic techniques, applicable to a wide range of security models, to assist in proofs of security of cryptographic schemes. Specifically, a "game hopping" proof is a powerful technique to prove strong statements about the security of a protocol. However, the proof technique often involves esoteric and repetitive steps. We demystify the methodology and prove generic theorems to reduce the length and complexity of a wide class of game hopping proofs.

Next, we use this new framework to analyse strong and previously unstudied formal security guarantees of authenticated key exchange protocols. These protocols are able to achieve strong security guarantees because they share and update state across sessions. Using our framework, we are able to push the state of the art by defying the folklore of what was previously thought provable of authenticated key exchange protocols. Specifically, it was previously thought that no protocol can achieve a security guarantee about communication with a peer if the peer was already fully compromised. We show that this is not true by formally defining a new concept, which we name post-compromise security. We capture an intuitive "self-healing" property of protocols. We then construct and formally prove protocols can achieve this strong security notion.

Finally, we leverage all of this work to analyse realistic protocols. Signal, which is used to encrypt messages in WhatsApp, Facebook Messenger and Google Allo amongst others, is one such protocol we consider. Despite being used by billions of people, Signal was completely unstudied in the academic literature prior to this work. Two reasons for this were that Signal had no documentation at the time and it did not fit into existing models: it shares state across sessions and has over ten different types of key, most of which continually update. Despite its complexity, we are able to use the other work in this thesis to give the first ever analysis and proof of security of Signal. Overall, this thesis aims to bridge the gap between theory and practice in the current state of the art of authenticated key exchange protocols.

Contents

Appendices

List of Abbreviations

0-RTT . . . . . Zero round trip time

ACCE . . . . . Authenticated and confidential channel establishment

AEAD . . . . . Authenticated encryption with associated Data

AES . . . . . Advanced Encryption Standard

AKC . . . . . Actor key compromise

AKE . . . . . Authenticated key exchange

ASICS . . . . . AKE Security Incorporating Certification Systems

CA . . . . . Certificate Authority

CBC . . . . . Cipher block chaining

CDH . . . . . Computational Diffie–Hellman

CK model . . . . . Canetti-Krawczyk model

DDH . . . . . Decisional Diffie–Hellman

DH . . . . . Diffie–Hellman

DHE . . . . . Diffie–Hellman ephemeral

Dual EC DRBG . Dual Elliptic Curve Deterministic Random Bit Generator

eCK model . . . . Extended Canetti-Krawczyk model

EdDSA . . . . . Edwards-curve Digital Signature Algorithm

ES . . . . . Entropy smoothing

EUF-CMA . . . . Existentially unforgeable under (adaptive) chosen-message attacks

GDH . . . . . Gap Diffie–Hellman

HKDF . . . . . HMAC-based key derivation function

HMAC . . . . . Keyed-hash message authentication code

HSM . . . . . Hardware security module

IACR . . . . . International Association for Cryptologic Research

IBE . . . . . Identity-based encryption
IEEE . . . . . Institute of Electrical and Electronics Engineers
IETF . . . . . Internet Engineering Task Force
IKE . . . . . Internet Key Exchange
IPsec . . . . . Internet Protocol security
IS . . . . . Intermediate secret
KCI . . . . . Key compromise impersonation
KDF . . . . . Key derivation function
MAC . . . . . Message authentication code
MPC . . . . . Multi-party computation
OTR . . . . . Off-the-Record Messaging
PCS . . . . . Post-compromise security
PGP . . . . . Pretty Good Privacy
PKCS . . . . . Public Key Cryptography Standards
PKI . . . . . Public key infrastructure
QR code . . . . . Quick response code
PFS . . . . . Perfect forward secrecy
PKG . . . . . Private key generator
PRF . . . . . Pseudorandom function
PRNG . . . . . Pseudorandom number generator
QUIC . . . . . Quick UDP Internet Connections
RSA . . . . . Rivest–Shamir–Adleman
SCIMP . . . . . Silent Circle Instant Message Protocol
SHA . . . . . Secure Hash Algorithm
TLS . . . . . Transport Layer Security
TPM . . . . . Trusted platform module
UC . . . . . Universal composability
UDP . . . . . User Datagram Protocol
UKS . . . . . Unknown key-share
XOR . . . . . Exclusive OR
ZRTP . . . . . Zimmermann Real-Time Transport Protocol

We can only see a short distance ahead, but we can see plenty there that needs to be done.

— Alan Turing, Computing Machinery and Intelligence, 1950

1 Introduction

Contents

1.1 Motivation

Cryptography is the study of techniques for secure communication in the presence of adversaries. People have used cryptography for centuries to protect their secrets. Obvious candidates to use cryptography include intelligence agencies and political activists living in oppressive regimes. However, modern-day life has seen the proliferation of online services, where the average person uses cryptography on a regular basis to protect their sensitive data as well. Examples include online banking, shopping and instant messaging.

The security of a cryptosystem is typically defined by more than just confidentiality of messages; protocols are designed to fulfil additional criteria such as integrity, authentication, non-repudiation, deniability or anonymity. This raises important natural questions such as:

(i) What, precisely, does one mean by the terms “confidentiality”, ‘integrity”, etc.?

(ii) What are the capabilities of the adversaries?

(iii) How does one know if a cryptosystem is secure?

Even with a seemingly strong cryptosystem, it is often difficult to answer (iii), particularly if (i) and (ii) are ill-defined. A famous example from history is before and during the Second World War, when rotor machines were extensively used to transmit secret radio messages. Most famously, the Nazi forces used the Enigma machine for naval communications. There was no guarantee that the Enigma was secure, except that it had an unfathomably large key space (with some configurations having over 158 quintillion possible settings) and that many intelligent people had tried to break it and failed. Initial attempts to break the Enigma started as early as 1932 by Polish researchers. However, the confidence the Nazis had in the Enigma was misplaced. Using the earlier work of the Polish, Alan Turing and his team in Hut 8 at Bletchley Park were eventually able to break the Enigma cipher, potentially shortening the war by years and saving millions of lives.

One way to help ensure security is to keep the cryptosystem itself secret (known as security through obscurity), but this is not feasible in the real world. In the real world, people are bribed, blackmailed or threatened into revealing how a cryptosystem works. In the case of the Enigma, a German traitor named Hans-Thilo Schmidt sold Engima secrets to the French in the 1930s, and in 1941, the crew of the HMS Bulldog were able to capture an intact Enigma machine and codebooks from the German U-boat U-110. In the modern-day context, the inner workings of software can be reverse engineered with memory dumps and debuggers, while hardware can be reverse engineered with tools such as microscopes to read computer chips. The assumption that the adversary knows how the cryptosystem works is known as Kerckhoffs’ principle. It is a necessary assumption in the modern world of cryptography.

Instead, one can formalise and answer all three questions above using mathematics. A branch of cryptography known as provable security aims to prove that cryptosystems meet precisely defined standards such as confidentiality, integrity, etc., in the same way

that a mathematical proof aims to prove that a theorem is true. First, an adversarial model is defined as a game between an adversary and a challenger. The game should capture the security properties of the cryptosystem: if the adversary can easily win the game, then the cryptosystem is insecure. Next, a reduction argument is made, where it is shown that if the adversary can somehow win the game, then either this happens with acceptably low probability or one can use the adversary to solve an underlying problem that is assumed to be hard, such as the discrete logarithm problem. This proof technique provides the guarantee that if the assumptions in the model are correct, then the cryptosystem is secure no matter how ingenious the adversary.[1]

But before we even begin to analyse if a cryptosystem is secure, the parties involved need to know how to communicate. The Enigma and modern-day alternatives use *symmetric* key cryptography, where it is assumed that the communicating parties already share a secret key. Communicating parties use their shared cryptosystem together with their shared secret key to communicate securely over an insecure channel such as radio or the internet. But how do the parties establish a shared secret key to begin with?

One of the most important problems in cryptography is the key distribution problem. The Nazis dealt with it by manually delivering codebooks to their outposts with Engima settings for each day of the month. More generally, a secure courier can be used to deliver a key. Of course, in the modern-day context, where billions of people communicate multiple times to other people and institutions all over the world, such a solution is most often impractical. The solution is *asymmetric* cryptography, where parties do not share any secrets to begin with, but they use each other's public keys and their own corresponding secret keys to communicate.

Unfortunately, asymmetric cryptography is orders of magnitude slower than symmetric cryptography. Therefore, one of the most important areas of cryptography is the study of authenticated key exchange (AKE) protocols. This is where parties use

[1]Real-life attacks are sometimes not captured by the model at all, so the assumptions in the model are important. For example, provable security rarely considers severe side-channel attacks or flawed implementations: the mathematics behind a protocol may be secure, yet the protocol will be insecure if its implementation is insecure.

asymmetric cryptography to authenticate each other and establish a shared session key that only they know. The session key is a symmetric key that is then used to efficiently communicate in a secure authenticated channel. Authenticated key exchange protocols are ubiquitous across the internet [33]. Notable examples include Secure Sockets Layer (SSL), also known as Transport Layer Security (TLS), which is used to establish an encrypted authenticated connection between a client and server; Secure Shell (SSH), which is used to securely access a computer remotely; Internet Key Exchange (IKE or IKEv2), which is used to establish a security association in the Internet Protocol Security (IPsec) suite; and Signal, which is used for end-to-end encryption of text messages between users.

Like the Enigma, modern AKE protocols often lack proofs of security. Vulnerabilities are found on a regular basis [46, 107]. The continued design and refinement of AKE protocols has been largely driven by newly discovered attacks [21, 23, 88, 95]. The design philosophy of continually patching vulnerabilities yields increasingly complicated protocols with unclear security guarantees.

On the other hand, a proof of security in a strong model provides significant confidence in the design of a protocol. Unfortunately, most state-of-the-art protocols with provable security guarantees [51, 88, 95] are not used in practice. Furthermore, there needs to be confidence in the security proof itself; sadly, security proofs are often lengthy, esoteric and difficult to follow. Errors can go unnoticed or it can even be unclear what concrete security guarantees some proofs actually provide [83–85, 121].

One reason for such complex proofs in the literature is because the security models are very intricate. This can sometimes be a good thing: the models attempt to capture many different realistic security failures, such as long-term keys, session keys or random number generators being compromised. A good protocol should be secure even if some compromise occurs, just like a well-designed skyscraper should stand tall even in the unfortunate event of an earthquake.

Thus, academic research contains a cornucopia of strong security models and theoretical protocols that satisfy them, often with security proofs that are hard to follow. Unfortunately, these strong security properties are rarely proven of real-world protocols.

Among the reasons we find are that real-world protocols typically (i) are substantially more complex than the academic protocols, and therefore harder to formally prove correct, (ii) were not designed to meet the newer requirements (e.g. resilience against weak random number generators) and (iii) do not meet the formal security definitions for technical reasons, even though this does not imply attacks in the real world.

Thus, there is a large gap between theory and practice in the current state of the art of AKE protocols. This thesis aims to bridge this gap. The three high-level goals of this thesis are to develop the state of the art for realistic, strong and provable key exchange security.

1.2 Research questions

This thesis investigates advancing the state of the art for realistic, strong and provable key exchange security. We want to investigate real-world protocols and either find attacks or show that they have strong, provable security guarantees. However, without the framework to analyse such protocols, little progress can be made. It therefore makes sense to tackle these three themes in reverse: to first build a framework for real-world provable security, to then use it to investigate and understand strong security properties and then apply all of the new understanding to real-world protocols.

Provable

In the first part of this thesis, we address the problems of the complexity and length of security proofs. As mentioned in the previous section, AKE models are often based on games, and therefore proofs of security in these models are reduced to arguments about games. The original attack game defines the environment, the powers of the adversary and what it means to say that a protocol is secure.

A game hopping proof is a proof technique where one considers ("hops" between) similar games that eventually lead from the original game to a game where the protocol is certainly secure. If one bounds the difference in success probabilities of the adversary between successive games in the sequence, then one can prove the security of a protocol in the original game. Unfortunately, game hopping proofs are often long, complex

and contain repetitive steps and unorthodox assumptions [68, 78]. This leads us to the following natural research question:

Research question. *What generic proofs and methodologies exist to reduce the length and increase the clarity of game hopping proofs?*

Strong

In the second part of this thesis, we apply our provable security knowledge to model strong security properties for state-of-the-art AKE protocols. Our ultimate aim is apply our understanding to the analysis of realistic protocols. As such, we draw our inspiration from the behaviour of some complex real-world AKE protocols. These protocols have substantially different behaviour to any academic protocol, and thus they potentially have novel security properties we can formally model.

Particular examples of protocols that have been notoriously hard to formally analyse are Off-the-Record Messaging (OTR) [30] and Signal [113]. These protocols are unlike most AKE protocols in that they are used for instant messaging conversations between users, and, rather that a simple AKE protocol that derives a single session key, these protocols continually send and update new keying material to derive per-message encryption keys. Because new keying material is continually being added, the participants, Alice and Bob, are sharing state. By contrast, most of the academic literature on AKE protocols only considers stateless protocols; that is, ones where state is not even shared across the sessions of a single party, let alone synchronised between Alice and Bob.

Signal, OTR and others like them may then provide a kind of "self-healing" property, where a participant can "recover" from a compromised state by introducing new keying material as the conversation goes on. In other contexts, similar informal language like "backward secrecy", "future secrecy" or "key continuity" has been used. In all cases, the intuition is clear, but no formal modelling or security guarantees have actually been investigated. It has always been cryptographic folklore that Alice cannot have a formal security guarantee about future communication with Bob if Bob's long-term key is already compromised, yet these protocols seem to defy this.

Research question. *Is it possible to formally model and prove a strong security property that has the intuition of "self-healing"?*

Research question. *What practical "self-healing" security properties are achievable and what mechanisms provide them?*

Realistic

In the third part of this thesis, we apply our new understanding to model stronger security guarantees for real-world state-of-the-art AKE protocols. In particular, we investigate the provable security guarantees of Signal.

Signal is a security protocol and accompanying app from Open Whisper Systems that provides end-to-end encryption for both text messages and phone calls. The core protocol has recently been adopted by WhatsApp, Facebook Messenger, Google Allo, Skype, CryptoCat and Viber amongst others. At the time of writing, the first two of these alone have over 1 billion active users. Yet, despite the enormous uptake and increasing popularity of Signal, there have been no publications analysing the formal security properties of the protocol. The reason for this, as with many other realistic protocols, is that it is a lot more complex than academic counterparts. Signal in particular has synchronised state between Alice and Bob, and each party has over ten different types of key.

Research question. *What provable security guarantees does Signal provide?*

1.3 Thesis overview

In this section, we provide an overview of the structure of the thesis and a synopsis of each chapter. Where applicable, we will give citations to full publications and then outline the main contributions from the author where the work had co-authors.

Chapter 2: Background

Here we define all the cryptographic notation and assumptions we will need for the proofs and analyses in the rest of the thesis.

Chapter 3: Game Hopping

The synopsis of this chapter is as follows:

1. First, we take a high-level view of the game hopping proof methodology. We explain in detail the different kinds of game hops and how they are used.

2. Next, we demystify the technique by giving a thorough example of how knowing what combination of game hops to do in a proof can be intuitive. After this, we provide general rules of thumb for game hopping proofs.

3. We then define a generic AKE model and provide a series of generic game hoping theorems. These theorems are widely applicable and can be used to simplify game hopping proofs so that one may focus on the crux of the proof, rather than the tedious, repetitive steps. We will use this work in later chapters of this thesis.

This work was featured as a solo term paper during the course of the author's DPhil. The theorems, examples and rules of thumb are all unique contributions, but the overall research theme came from DPhil supervisor Cas Cremers.

Chapter 4: Post-Compromise Security

The synopsis of this chapter is as follows:

1. We introduce the high-level concept of *post-compromise security* which formally captures the intuitive property of a protocol self-healing. This notion has wide applicability, and we use it to unify the potential security guarantees offered by hardware security modules, trusted platform modules and by regular key rotation as in Signal and OTR.

2. We instantiate post-compromise security in the context of authenticated key exchange protocols, advancing the state of the art in what provable security guarantees are achievable. Our models formally incorporate post-compromise security, and they are strictly stronger than the best known game-based definitions.

3. We give two specific approaches to achieve post-compromise security. The first allows certain computations with the long-term key but does not reveal it fully. This is similar to compromising an hardware security module but not the secret key within. We prove that a form of post-compromise security can be achieved in this case by relatively standard protocols.

4. The second approach permits the adversary to learn the long-term key of an agent before attempting an impersonation attack. We show that no stateless protocol can achieve this type of security, but we provide a generic protocol transformation to show that stateful protocols can achieve it. Our transformation produces protocols that are robust to an unreliable network and have denial of service resistance, which turns out to be non-trivial.

5. We supplement our work with case studies showing the practical applicability of post-compromise security. We show with examples of OPTLS (a recent TLS 1.3 proposal) and TextSecure/Axolotl (the predecessor to Signal), that practical protocols can fail to achieve post-compromise security even when they seemingly employ the right mechanisms.

6. Our generic protocol transformation gives rise to a hierarchy of strong AKE models and protocols that meet them. We formally reason about the relations between these models and additionally show that our transformation can provide strong per-peer psuedorandom number generation security guarantees.

7. We pave the way for the future analyses of stateful, realistic, state-of-the-art protocols such as Signal in Chapter 5.

This work was featured as a full publication at the Computer Security Foundations (CSF) Symposium in 2016 [44], with co-authors Cas Cremers and Katriel-Cohn-Gordon. The main individual contributions from the author were the formal modelling, security proofs and evaluations of the security model hierarchies.

Chapter 5: Signal Protocol

The synopsis of this chapter is as follows:

1. As a first step, we provide substantial documentation for how Signal actually works, as at the time of our work there was no documentation so the only option was to look through the source code of Signal to understand how it functioned.

2. We define the first security model that actually describes Signal; the protocol employs a novel and unstudied design, involving over ten different types of keys and a complex update process which leads to various "chains" of related keys. It therefore does not directly fit into existing security models. Moreover, the post-compromise security properties of Signal were not even formalised until the work of Chapter 4. Our model is a multi-stage key exchange model that can also be easily adapted for other protocols with similar mechanism to Signal.

3. We then provide the first in-depth formal security analysis of the cryptographic core of the Signal messaging protocol. This is the first ever proof of security of Signal as well as the first ever proof of post-compromise security for a real-world protocol. We provide both a proof in the random oracle model and a proof using a PRF-ODH assumption.

This work was featured as a full publication at the European Symposium on Security and Privacy (EuroS&P) in 2017 and as an extended abstract at Real World Crypto 2017 [43]. A journal version is also currently under submission. The co-authors are Cas Cremers, Katriel Cohn-Gordon, Ben Dowling and Douglas Stebila.The main individual contributions from the author were the formal modelling and security proof.

Chapter 6: Related Work

Here we summarise the related work to this thesis. The related work is analysed and generally partitioned logically into sections relating to the other chapters of this thesis.

Chapter 7: Conclusions

Here we provide a high-level conclusion of the work in this thesis and outline possible future work.

Appendices

Many of the longer proofs can be found in the appendices.

If you wish to make an apple pie from scratch, you must first invent the universe.

— Carl Sagan, Cosmos: The Lives of the Stars, 1980

2 Background

Contents

In this chapter we define our notation and recall some standard cryptographic definitions and assumptions that we will use throughout this thesis.

2.1 Notation

We write $v \leftarrow x$ to denote the assignment of x to the variable v and $x \leftarrow_{\$} X$ to denote that the variable x is assigned a value randomly chosen according to the distribution X. If S is a finite set, we write $x \leftarrow_{\$} S$ to mean that x is assigned a value chosen uniformly at random from S. We write $\oplus$ to denote the bitwise exclusive OR (XOR) operation. The XOR operation takes two bit strings of equal length and performs the logical exclusive OR operation on each pair of corresponding bits to output a bit string of the

same length. The result in each position is 1 if only the first bit is 1 or only the second bit is 1, but will be 0 if both are 0 or both are 1. For example, $1100 \oplus 1010 = 0110$.

We say that an algorithm is *efficient* if the running time is polynomial in the length of its input. We say that a function $f(k)$ is *negligible* if for all $c > 0$ there exists a k_0 such that for all $k > k_0$, $|f(k)| < k^{-c}$. In the context of security protocols, when we say functions and bounds are negligible, we mean negligible with respect to the security parameter(s) of a protocol. We denote the security parameter by λ. We use $\mathcal{D}$ to denote distinguisher algorithms and $\mathcal{A}$ to denote an adversary. We use hat notation such as $\hat{A}$, $\hat{B}$, $\hat{C}$, to denote actors Alice, Bob, Charlie, etc. We denote the long-term asymmetric public and private key pair of an AKE protocol by pk and sk respectively. We similarly use esk to denote an ephemeral secret key. We use subscripts to denote ownership, e.g. $\hat{A}$'s private key is $\mathsf{sk}_{\hat{A}}$. Sometimes we will just use (a, x, b, y) instead of $(\mathsf{sk}_{\hat{A}}, \mathsf{esk}_{\hat{A}}, \mathsf{sk}_{\hat{B}}, \mathsf{esk}_{\hat{B}})$, but this will always be clear from context. We use the notation $\mathsf{Enc}(\mathsf{pk}, m)$ and $\mathsf{Enc}(k, m)$ to denote encryption of message m either asymmetrically with public key pk or symmetrically under key k. We similarly use Dec for decryption. We write $\mathsf{Sig}_{\mathsf{sk}}(m)$ to mean the signature of message m under key sk and similarly $\mathsf{MAC}(m, k)$ to mean the MAC of message m using key k.

2.2 Diffie–Hellman assumptions

The proofs of security in this thesis rely on standard cryptographic hardness assumptions related to Diffie–Hellman key exchange [57]. Let $G = \langle g \rangle$ be a cyclic group of order q generated by g, let $x, y, z \leftarrow_{\$} \mathbb{Z}_q$ and let $\mathcal{O}_{\text{DDH}}$ be an efficient black box algorithm (oracle) that can distinguish (g^x, g^y, g^{xy}) from (g^x, g^y, g^z). For any algorithm $\mathcal{D}$ let

$$\begin{aligned}
\epsilon_{\text{DDH}}(\mathcal{D}) &\coloneqq \Big| \Pr\left(x, y \leftarrow_{\$} \mathbb{Z}_q : \mathcal{D}(G, q, g, g^x, g^y, g^{xy}) = 1\right) \\
&\qquad - \Pr\left(x, y, z \leftarrow_{\$} \mathbb{Z}_q : \mathcal{D}(G, q, g, g^x, g^y, g^z) = 1\right) \Big| \\
\epsilon_{\text{CDH}}(\mathcal{D}) &\coloneqq \Pr\left(x, y \leftarrow_{\$} \mathbb{Z}_q : \mathcal{D}(G, q, g, g^x, g^y) = g^{xy}\right) \\
\epsilon_{\text{GDH}}(\mathcal{D}) &\coloneqq \Pr(x, y \leftarrow_{\$} \mathbb{Z}_q : \mathcal{D}^{\mathcal{O}_{\text{DDH}}}(G, q, g, g^x, g^y) = g^{xy})
\end{aligned}$$

We make use of the following cryptographic hardness assumptions:

1. Decisional Diffie–Hellman (DDH): it is hard to distinguish (g^x, g^y, g^{xy}) from (g^x, g^y, g^z), i.e. $\epsilon_{\mathrm{DDH}}(\mathcal{D})$ is negligible in q for any efficient $\mathcal{D}$.

2. Computational Diffie–Hellman (CDH): it is hard to compute the value g^{xy} from (g^x, g^y), i.e. $\epsilon_{\mathrm{CDH}}(\mathcal{D})$ is negligible in q for any efficient $\mathcal{D}$.

3. Gap Diffie–Hellman (GDH): it is hard to compute the value g^{xy} from (g^x, g^y) even when given black box access to an oracle that can answer the DDH problem, i.e. $\epsilon_{\mathrm{GDH}}(\mathcal{D})$ is negligible in q for any efficient $\mathcal{D}$.

The relative difficulty of these problems is thought to be dependent on the group. For example, in $\mathbb{Z}_p^\star$ for prime p the CDH problem is believed to be hard, while the DDH assumption does not hold because given (g^x, g^y), one can efficiently compute the Legendre symbol of g^{xy}. This yields an efficient method to distinguish g^{xy} from a random group element with non-negligible probability.

2.3 Signature schemes

A signature scheme is a triple $(\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$. KGen is a probabilistic algorithm that takes as input the security parameter λ and outputs a public signature verification key pk and secret signing key sk. Sig is a (possibly probabilistic) signing algorithm that generates a signature σ for message m using secret key sk. Vf is a deterministic signature verification algorithm that, given input (pk, σ, m), outputs 1 if σ is a valid signature of m under key pk and 0 otherwise. It is required that for every k, every $(\mathsf{sk}, \mathsf{pk})$ output by $\mathsf{KGen}(\lambda)$ and every message m, it holds that

$$\mathsf{Vf}(m, \mathsf{Sig}_{\mathsf{sk}}(m)) = 1$$

Next, we define a signature scheme to be existentially unforgeable under (adaptive) chosen-message attacks (EUF-CMA). Consider the following game between a challenger and a polynomial time adversary:

1. The challenger generates a public/private key pair $(\mathsf{pk}, \mathsf{sk})$ using KGen and gives the adversary $\mathcal{A}$ the public key pk.

2. $\mathcal{A}$ is allowed to query adaptively chosen messages $m_1, \ldots, m_q$ for some $q \in \mathbb{N}$ to the challenger. The challenger responses to each query with $\sigma_i = \mathsf{Sig}(\mathsf{sk}, m_i)$.

3. After the adversary asks all of her queries, she outputs a pair (m, σ).

Definition 2.1 (EUF-CMA security)**.** *A signature scheme* $(\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$ *with security parameter* λ *is EUF-CMA-secure if for all efficient adversaries* $\mathcal{A}$ *it holds that*

$$\Pr\left((m, \sigma) \leftarrow_{\$} \mathcal{A} : \mathsf{Vf}(\mathsf{pk}, m, \sigma) = 1 \wedge m \notin \{m_1, \ldots, m_q\}\right) \coloneqq \epsilon_{SIG}$$

is negligible in the security parameter.

Sometimes we will write $|\Pr(S_0) - \Pr(S_1)| \leq \epsilon_{\mathrm{SIG}}$ to mean that Game 0 and Game 1 are indistinguishable under the assumption that a signature scheme is EUF-CMA-secure.

2.4 Pseudorandom functions

A pseudorandom function is an algorithm PRF. This algorithm implements a deterministic function $z = \mathsf{PRF}(k, x)$, taking as input a key k and some bit string x, and returning a string $z \in \{0,1\}^{\mu}$. To define a PRF, consider the following game:

1. The challenger selects a key k.

2. The adversary $\mathcal{A}$ is allowed to adaptively query the challenger $x_1, \ldots, x_q$ and the challenger responds to query x_i with $z_i = \mathsf{PRF}(k, x_i)$.

3. Eventually the adversary must output a value x. The challenger computes $z_0 \coloneqq \mathsf{PRF}(k, x)$ and selects $z_1 \leftarrow_{\$} \{0,1\}^{\mu}$. The challenger then selects $b \leftarrow_{\$} \{0,1\}$ and sends z_b back to the adversary.

4. The adversary then outputs a guess $\hat{b} \in \{0,1\}$ for b.

Definition 2.2 (pseudorandom function)**.** *We say a function* PRF *is pseudorandom if for all efficient adversaries* $\mathcal{A}$ *it holds that*

$$|\Pr(\hat{b} = b) - 1/2| \coloneqq \epsilon_{\mathsf{PRF}}$$

is negligible.

Sometimes we will write $|\Pr(S_0) - \Pr(S_1)| \leq \epsilon_{\mathsf{PRF}}$ to mean that Game 0 and Game 1 are indistinguishable under the assumption that a certain function is pseudorandom.

2.5 Semantic security

An encryption scheme takes as input a security parameter λ which is the secret key size. The intuition behind semantic security is that no efficient adversary should be able to distinguish between the encryption of two messages of her own choosing [69]. Semantic Security is formally defined in terms of a game between an adversary and a challenger as follows:

1. The challenger computes $(\mathsf{pk}, \mathsf{sk}) \leftarrow_{\$} \mathsf{KGen}$ and gives pk to the adversary.
2. The adversary $\mathcal{A}$ chooses two messages $m_0, m_1 \in M$ of equal length and gives these to the challenger.
3. The challenger computes

   $$b \leftarrow_{\$} \{0,1\},\ \psi \leftarrow_{\$} \mathsf{Enc}(\mathsf{pk}, m_b)$$

   and gives the target ciphertext ψ to the adversary.
4. The adversary outputs a guess $\hat{b} \in \{0,1\}$.

Definition 2.3 (Semantic security)**.** *We say that an encryption scheme with security parameter* λ *is semantically secure if for all efficient (polynomial time) adversaries* $\mathcal{A}$ *it holds that*

$$|\Pr(\hat{b} = b) - 1/2| \coloneqq \epsilon_{SS}$$

is negligible in the security parameter.

Sometimes we will write $|\Pr(S_0) - \Pr(S_1)| \leq \epsilon_{\text{SS}}$ to mean that Game 0 and Game 1 are indistinguishable under the assumption that an encryption scheme is semantically secure.

2.6 Entropy smoothing family of hash functions

Let G be a group. Informally, we say that a family $\mathcal{H} = \{H_k\}_{k \in K}$ of hash functions, where each H_k is a function from G to $\{0,1\}^l$, is entropy smoothing (ES) when it is hard to distinguish $(k, H_k(\delta))$ from (k, h), where $k \leftarrow_\$ K$, $\delta \leftarrow_\$ G$ and $h \leftarrow_\$ \{0,1\}^l$. More formally, let $\mathcal{D}$ be an algorithm that takes as input an element of K and an element of $\{0,1\}^l$, and outputs a bit. We say that the family of hash functions $\mathcal{H} = \{H_k\}_{k \in K}$ is entropy smoothing if for all efficient (polynomial time) adversaries $\mathcal{D}$, it holds that ϵ_{ES} is negligible in the security parameter(s), where ϵ_{ES} is defined as

$$|\Pr(k \leftarrow_\$ K,\ \delta \leftarrow_\$ G : \mathcal{D}(k, H_k(\delta)) = 1) - \Pr(k \leftarrow_\$ K,\ h \leftarrow_\$ \{0,1\}^l : \mathcal{D}(k, h) = 1)|$$

Sometimes we will write $|\Pr(S_0) - \Pr(S_1)| \leq \epsilon_{\text{ES}}$ to mean that Game 0 and Game 1 are indistinguishable under the assumption that a family of hash functions are entropy smoothing.

2.7 Collision-resistant hash functions

We say that a hash function $\mathcal{H}$ is collision-resistant if, for all efficient (polynomial time) adversaries $\mathcal{A}$ it holds that

$$\Pr(\mathcal{A}(\mathcal{H}) = (m, m') : m \neq m' \wedge \mathcal{H}(m) = \mathcal{H}(m')) \coloneqq \epsilon_{\mathcal{H}}$$

is negligible for any probabilistic, polynomial time adversary.

We will write $|\Pr(S_0) - \Pr(S_1)| \leq \epsilon_{\mathcal{H}}$ to mean that Game 0 and Game 1 are indistinguishable under the assumption that a hash function is collision-resistant.

2.8 Random oracle model

A random oracle is an oracle (a theoretical black box) that responds to every unique query with a response chosen uniformly at random from its output domain. Additionally, if a query is repeated, the random oracle always responds the same way. The only way to determine the random oracle output of any input is to query the random oracle. The random oracle model [15] is the modelling assumption that hash functions, key derivation functions, psuedorandom functions, etc., are random oracles.

The random oracle model has received some criticism as an assumption in the cryptographic community. It has even been shown [12, 40] that it is possible to construct cryptographic schemes that are secure in the random oracle model but insecure when any concrete primitive is substituted in its place. However, all known counterexamples to the validity of the random oracle model are pathological, and there are examples where deliberate avoidance of its use have actually caused more significant security failures [86]. Despite some criticism, the random oracle model remains an intuitive assumption and powerful tool to allow researchers to prove statements about cryptographic protocols.

2.9 Authenticated key exchange security models

Throughout this thesis, we instantiate our analysis of AKE protocols using game-based computational models. We refer the reader to [37, 51, 95] for more complete definitions.

We work in the tradition of Bellare and Rogaway [16], modelling agents (Alice, Bob, ...) as a collection of oracles $\Pi^s_{i,j}$ representing agent i talking to intended communication partner j in its s^{th} session. The set of all parties is denoted by $\mathcal{P}$. We denote the number of parties $|\mathcal{P}|$ by $\mathsf{n_P}$. The adversary, Eve, is a probabilistic polynomial time Turing machine with access to these oracles. The oracles only ever communicate via Eve and never directly with each other. As well as relaying messages, Eve has access

to certain other powers that are formally implemented as queries. Many different queries have been studied [23, 41, 95] and used to model various security properties. A protocol in this context succeeds if (i) it is correct, in the sense that the parties who intend to talk to each other (and only they) derive matching communication keys and (ii) it is secure, meaning that Eve cannot distinguish the so-called Test session key from a string chosen uniformly at random. Properties (i) and (ii) are captured formally in Definition 2.9 and Definition 2.10 respectively.

We use notation similar to [51]. Let π denote an AKE protocol of $h \geq 2$ messages. Each agent can execute any arbitrary number of instances of π, called sessions. We also assume that there is a fixed upper limit to the maximum possible number of sessions any agent can have in the game. We denote this upper limit by $\mathsf{n_S}$. Sessions can uniquely identified by their actor and the order in which they are created (i.e. by the pair $\hat{P}, i \in \mathcal{P} \in \mathbb{N}$). In any session, the session actor takes on the role of either initiator ($\mathcal{I}$) or responder ($\mathcal{R}$). Sessions of an actor are implemented by a probabilistic polynomial time Turing machine with memory contents both session-specific memory (denoted st_s for each session s) and long-term user memory (denoted $st_{\hat{P}}$ for party $\hat{P}$) that is shared across all sessions.

The contents of session memory st_s is described in Table 2.1 and is always local to a specific session s. In contrast, user memory $st_{\hat{P}}$ is shared across all sessions of the user $\hat{P}$ and consists of $\hat{P}$'s public/private key pair $(\mathsf{pk}_{\hat{P}}, \mathsf{sk}_{\hat{P}})$, the identities and corresponding public keys of all other users, as well as any other additional information required by the protocol. We always assume throughout this thesis that the protocol algorithm runs only one step of a session at a time, though many sessions may execute concurrently.

A protocol π is specified by an algorithm to generate the public/private key pair and a protocol algorithm executed by each session oracle. The protocol algorithm takes as input the current session, user state and an incoming message, and returns an outgoing message as well as new session and user state values.

To define protocol security, a game is played between a challenger and an adversary. The challenger implements all of the parties communicating. A setup algorithm first generates user identifiers and key pairs, sets all session-specific variables to an initial

Name	Description
s_{actor}	The identity of the agent executing session s, where $s_{\text{actor}} \in \mathcal{P}$.
s_{role}	The role (initiator $\mathcal{I}$ or responder $\mathcal{R}$) played by s_{actor} in s.
s_{peer}	The intended communication partner of s_{actor}, where $s_{\text{peer}} \in \mathcal{P}$.
s_{key}	The session key established during the session.
s_{rand}	The randomness used in the session.
s_{step}	The next step to be executed in the protocol.
s_{status}	The session status, where $s_{\text{status}} \in \{\perp, \texttt{active}, \texttt{accepted}, \texttt{rejected}\}$.
$s.m_i$	The i^{th} message sent.
s_{sent}	All messages sent by s_{actor} during the session.
s_{recv}	All messages received by s_{actor} during the session.

Table 2.1: Generic contents of session memory for session s of AKE protocol π.

Query	Description
$\mathsf{create}(\hat{A}, r, \hat{B})$	Create a new session oracle at $\hat{A}$ with role r and peer $\hat{B}$; randomness is sampled for s_{rand} and the protocol algorithm is executed to update the state and return a message.
$\mathsf{send}(\hat{A}, i, \hat{m})$	Execute the protocol on the i^{th} session oracle of $\hat{A}$, update the state and return the result.
$\mathsf{session\text{-}key}(\hat{A}, i)$	Reveal s_{key}, where $s = (\hat{A}, i)$.
$\mathsf{test}(s)$	Flip a coin $b \leftarrow_{\$} \{0,1\}$ and return k_b, where $k_1 \coloneqq s_{\text{key}}$ and k_0 is defined as a string chosen uniformly at random from the set of all possible session keys.
$\mathsf{guess}(b')$	End the game. The adversary wins if $b' = b$ and loses otherwise.

Table 2.2: An example set of queries available to $\mathcal{A}$ in a basic AKE model.

value $\perp$, initialises the user memory of each user including distributing all public keys, and initialises all other variables to $\perp$. We choose not to model dynamic key registration for simplicity and because we do not anticipate it leading to any problems unique to the topics addressed in this thesis. The adversary then plays the game with access to some set of queries in the particular AKE model (e.g. the queries of Table 2.2). The goal of the adversary is to attack the so-called Test session that it can choose with the $\mathsf{test}(s)$ query, receive as reply k_b for unknown b, and then correctly guess b with the $\mathsf{guess}(b')$ query. In other words, the adversary aims to win the game by correctly distinguishing the Test session key from a string chosen uniformly at random. Since unrestricted use of queries in the AKE models we consider

can trivially break any session (e.g. revealing the Test session key directly with a session-key query), we impose some restrictions.

Definition 2.4 (Freshness predicate)**.** *A* freshness predicate *is a Boolean predicate that takes as input a session of some protocol π and a sequence of queries (including arguments and results).*

Definition 2.5 (Authenticated key exchange security model)**.** *For AKE protocol π, let Q denote $\mathcal{A}$'s set of queries, and let F be a freshness predicate. We call (Q, F) an authenticated key exchange security model.*

The boolean freshness predicates exist to explicitly rule out trivial attacks from the adversary. The less stringent the conditions for the Test session to be defined as fresh, the stronger the adversary. We are now ready to formally define our security game.

Definition 2.6 (Security game)**.** *Let π be an AKE protocol and $X = (Q, F)$ be an AKE security model. We define the experiment $W(X)$ as follows:*

(i) The setup algorithm is executed.

(ii) The adversary $\mathcal{A}$ learns all public information (e.g. user identifiers and public keys) and then computes arbitrarily, performing a sequence of queries from the set $Q \setminus \{\mathsf{test}, \mathsf{guess}\}$.

(iii) At some point during $W(X)$, the query $\mathsf{test}(s)$ is issued on some session s that has accepted and satisfies F at the time the query is issued.

(iv) The adversary may continue issuing queries from Q under the condition that s continues to satisfy F.

(v) The adversary $\mathcal{A}$ outputs a bit b' via the query $\mathsf{guess}(b')$ as the guess for the bit b chosen by the challenger during the test *query. The game ends, and $\mathcal{A}$ wins if and only if $b = b'$.*

Following [36, 37], we split our analysis of an AKE protocol security into two parts: correctness (matching is correctly implemented and implies agreement on the derived key) and secrecy (the adversary cannot distinguish the derived key from random). For this, and for later definitions of specific freshness predicates, the following definitions of types of matching will be useful:

Definition 2.7 (Matching and partial matching)**.** *Let π be a protocol of $h \geq 2$ total messages, where, if h is even, the number of messages sent from both sides is equal and if h is odd, the initiator sends one extra message. Let s denote a session of π with $s_{status} =$* `accepted`*. We say that session s partially matches session s' in status $s'_{status} \neq \perp$ if the following conditions hold, where $s_{send}[1 \dots l]$ denotes the concatenation of the first l messages sent by session s and $s_{recv}[1 \dots l]$ denotes the concatenation of the first l messages received by s:*

- $s_{role} \neq s'_{role} \wedge s_{actor} = s'_{peer} \wedge s_{peer} = s'_{actor}$ *and either*
- $s_{role} = \mathcal{I} \wedge s_{send}[1 \dots m] = s'_{recv}[1 \dots m] \wedge s_{recv}[1 \dots m] = s'_{send}[1 \dots m]$ *with* $m = \frac{h}{2}$ *if h is even and* $m = \frac{h-1}{2}$ *if h is odd, or*
- $s_{role} = \mathcal{R} \wedge s_{send}[1 \dots (m-1)] = s'_{recv}[1 \dots (m-1)] \wedge s_{recv}[1 \dots m] = s'_{send}[1 \dots m]$ *with* $m = \frac{h}{2}$ *if h is even and* $m = \frac{h+1}{2}$ *if h is odd.*

In addition, if $(s_{status}, s_{sent}, s_{recv}) = (s'_{status}, s'_{recv}, s'_{sent})$, then we say that s matches *s'.*

Definition 2.8 (Origin-session)**.** *We say that a session s' with $s'_{status} \neq \perp$ is an origin-session for a session s with $s_{status} =$* `accepted` *if $s'_{send} = s_{recv}$.*

Definition 2.9 (AKE correctness)**.** *A protocol π is said to be* correct *in the security model (Q, F) if, for all probabilistic polynomial time adversaries $\mathcal{A}$, it holds that:*

(i) If completed sessions s and s' match each other, then $s_{key} = s'_{key}$.

(ii) $\Pr(\mathsf{Multiple\text{-}Match}^{\pi,\mathcal{A}}_{W(X)}(k))$ *is negligible, where* Multiple-Match *denotes the event that there exists a session with at least two matching sessions.*

Definition 2.10 (Authenticated key exchange security)**.** *An AKE protocol π is said to be* secure *in the security model (Q, F) if, for all probabilistic polynomial time adversaries $\mathcal{A}$, the $W(X)$-advantage*

$$\mathsf{Adv}^{\pi,\mathcal{A}}_{W(X)}(k) \coloneqq |\Pr(b = b') - 1/2|$$

of $\mathcal{A}$ is negligible in the security parameter.

Definition 2.11 (Protocol classes form [51])**.** *We define AKE as the class of all protocols of $h \geq 2$ total messages for which the probability that two sessions of the same user output in all protocol steps identical messages is negligible.*

The subclass ISM of AKE (for "initial state modification") consists of all protocols in AKE that only access and update user memory upon creation of sessions. The subclass SL of ISM consists of all stateless protocols; that is, those for which $st_{\hat{p}} = proj_3(\Psi(\lambda, st_s, st_{\hat{p}}, m))$ for all $(k, st_s, st_{\hat{P}}, m)$, where $proj_3$ is projection onto the third coordinate.

Definition 2.12 (Ranking of security models from [51])**.** *Let $secure(M, \pi)$ be a predicate that is true if and only if the protocol π is secure in security model M, and let Π be a class of AKE protocols. We say that a security model M' is* at least as strong *as a security model M with respect to Π, denoted by $M \leq^{\Pi}_{Sec} M'$, if*

$$\forall \pi \in \Pi, secure(M', \pi) \implies secure(M, \pi)$$

We say that M and M' are incomparable *if $M \not\leq^{\Pi}_{Sec} M' \wedge M' \not\leq^{\Pi}_{Sec} M$.*

State reveals

The traditional way to model compromise of ephemeral keys (e.g. x in a Diffie–Hellman (DH) key exchange that sends g^x) is to give the adversary a specific query to reveal them. This is natural, but it requires designers to explicitly specify which values can possibly be compromised. Moreover, the definition of security critically depends on this choice. For example, the NAXOS protocol uses session randomness x and sends Diffie–Hellman key $g^{H(\mathsf{sk},x)}$. Is x or $H(\mathsf{sk}, x)$ the "ephemeral key" in this case? To avoid this specification problem, more recent models have focused on reveal queries

that reveal the outputs of the random number generator, from which any intermediate value can be computed. So in this case there is a query to reveal x and a separate query to reveal the long-term key sk. By combining both of these values, $H(\mathsf{sk}, x)$ can be computed. We follow this trend, and use a $\mathsf{randomness}(s)$ query to reveal the session randomness x of session s, instead of a specific ephemeral key reveal.

Long-term secrets and short-term secrets

Security models usually distinguish between long-term secrets (e.g. asymmetric keys sk used to identify agents) and short-term secrets (e.g. ephemeral keys esk that are used once and then subsequently erased). Some argue that this distinction is rarely necessary; often an adversary will compromise a device and learn both types of secrets at the same time. However, there are practical scenarios where the adversary may learn one type of secret but not the other. In practice, the independent compromise of secrets can occur when random number generation comes from a vulnerable hardware component or software library. Notable examples include the recent Infineon security chips vulnerability [108], which affected the generation of long-term RSA keys; the NIST Dual EC DRBG [19] pseudorandom number generator (PRNG), which potentially has a backdoor for those who know the relationship between two specific points on the elliptic curve; and the Debian OpenSSL PRNG vulnerability [134], which resulted in only a very small number of possible seed values being used in the PRNG. In these cases, the long-term or ephemeral keys are compromised but not the whole state of a device. Further discussion can be found in [10]. Therefore, in this thesis, we follow the academic tradition of distinguishing between $\mathsf{randomness}$ and $\mathsf{corrupt}$ queries, where $\mathsf{randomness}(s)$ queries reveal the session randomness s_{rand} used in a particular session s, while $\mathsf{corrupt}(\hat{A})$ queries reveal the long-term key $\mathsf{sk}_{\hat{A}}$ of the targeted party $\hat{A}$. In all of the threat models we consider in this thesis, we will be sure to explicitly define what queries are available to the adversary and what the freshness predicate is.

Part I

Provable

In general, finding a good strategy for how to modify games, and the order in which to modify them, etc., is a bit of a "black art".

— *Victor Shoup, Sequences of Games: A Tool for Taming Complexity in Security Proofs, 2004 [122]*

3 Game Hopping

Contents

To prove an AKE protocol secure, one first defines a model and a game between a challenger and an adversary. The game should capture the security properties one wishes to prove; if the adversary cannot win this game with non-negligible advantage, then the protocol fulfils these security properties. The standard "game hopping" proof technique is where one considers a sequence of similar games. The idea is to bound the success probability of the adversary in each successive game in terms of the next one, until one reaches a game that the adversary clearly cannot win with non-negligible advantage. If the bounds between each game are good enough, this should prove that the adversary cannot win the original game with non-negligible advantage. Thus, the AKE protocol is proven to be secure.

Game hopping is a powerful technique because it allows for proofs of strong statements about the security of a cryptosystem. Game hopping is ubiquitous across

the academic literature [2, 29, 35, 38, 48, 63, 91, 94]. However, the technique comes with awkward downsides. Namely, the proofs are frequently esoteric, repetitive and lengthy. This makes them difficult to verify and prone to error. Unfortunately, this defeats the whole purpose of having a mathematical proof: to provide certainty in the security of a cryptosystem. Therefore, before we begin to address the complex protocols and models in this thesis, we first turn our attention to the generic problems facing this proof technique. Recall our earlier research question:

Research question. *What generic proofs and methodologies exist to reduce the length and increase the clarity of game hopping proofs?*

In this chapter, we answer this question and demystify the game hopping proof technique in the following ways. First, we recall and organise the different game hopping techniques that are spread across the academic literature. Then we show how it can be intuitive to know how to modify games and in which order so that it need not be a "black art" [122]. We then go on to extend these principles in a generic methodology framework. Finally, we prove generic theorems about how to hop between games. These generic principles and theorems will provide the foundation for security proofs in later chapters of this thesis.

Chapter overview

In Section 3.1, we recall and organise all of the known game hopping techniques that have been applied in security proofs across the literature. We partition the types of hops into four categories. In Section 3.2, we show with use of a minimal possible example that the order of game hops is not commutative. We then use this as a motivating example to show how to find the correct order of game hops. In Section 3.3, we provide a generic heuristic for game hopping proofs of AKE protocols, which we will apply in later chapters of this thesis. In Section 3.4, we provide generic theorems for how to hop between games, which we will also use in later chapters of this thesis.

3.1 Game hopping techniques

The security of a protocol can be proven by slowly altering the attack environment through a series of game hops until the success probability of the adversary can be computed as some negligible function of the security parameter(s). We also bound the differences in the success probability of the adversary in each game hop by a negligible function. This proves that the protocol in the original attack environment is secure. There are techniques used throughout the academic literature to do this, which we recall and organise here.

Throughout this thesis, the original security game is defined to be Game 0. It will always be made clear precisely which game this is.

3.1.1 The Difference Lemma

The following well-known known mathematical fact, known as the Difference Lemma, is frequently used in game hopping proofs to bound the success probability of the adversary between successive games. An early example of its use in cryptography can be found in [81].

Lemma 3.1 (Difference Lemma)**.** *Let A, B and F be events defined in some probability distribution. If $A \wedge \neg F \iff B \wedge \neg F$ then*

$$|\Pr(A) - \Pr(B)| \leq P(F)$$

Proof. Since $\Pr(A \wedge \neg F) = \Pr(B \wedge \neg F)$, it follows that

$$\begin{aligned}
|\Pr(A) - \Pr(B)| &= |\Pr(A \wedge F) + \Pr(A \wedge \neg F) - \Pr(B \wedge F) - \Pr(B \wedge \neg F)| \\
&= |\Pr(A \wedge F) - \Pr(B \wedge F)| \\
&\leq |\Pr(F) - \Pr(B \wedge F)| \text{ or } |\Pr(F) - \Pr(A \wedge F)| \\
&\leq \Pr(F)
\end{aligned}$$

□

3.1.2 The four types of game hop

There are four known types of game hop used in security proofs. Often a combination is used in a full proof, but rarely if ever are the type of hops made explicit. Here we list them and explain how they work. We will refer back to these types of hops throughout this thesis.

Let S_i denote the event that the adversary wins Game i (so we are always looking to bound $\Pr(S_0)$). One hops sequentially from Game i to Game $i+1$. The four known types of game hop are as follows:

- **Bridging.** Here the game is just reformulated in a totally equivalent way. The change is purely conceptual; consequently, we have the following equality:

  $$\Pr(S_i) = \Pr(S_{i+1})$$

  In principle, a bridging step may seem unnecessary, but without it, a proof can be much harder to follow. Generally, bridging hops lay the foundations for the other hops.

  An example of a bridging hop is thinking of a random function f as an oracle. Instead of evaluating f, we ask the oracle. The oracle keeps a table of previous input/output pairs. If a query is made that matches one of the previous inputs, the corresponding output is returned. Otherwise, an output value is chosen at random and a new input/output pair is added to the table.

- **Indistinguishability.** An early example of this technique can be found in [67]. If two distributions P_0 and P_1 are indistinguishable (either statistically or computationally), a game hop between Game i and Game $i+1$ based on indistinguishability works by considering a "hybrid game" between the two. The hybrid game, when run on distribution P_0, runs Game i, and when run on distribution P_1, runs Game $i+1$. The idea is that, to prove $|\Pr(S_i) - \Pr(S_{i+1})|$ is negligible, one defines a distinguishing algorithm $\mathcal{D}$ that interpolates between Game i and Game $i+1$. The distinguishing algorithm $\mathcal{D}$ receives an unknown

input, either from P_0 or P_1, and must determine which distribution it belongs to by running the hybrid game. It outputs 0 to signal it guesses the input belongs to distribution P_0 and 1 to signal it guesses the input belongs to distribution P_1. The two distributions are indistinguishable. This means that

$$|\Pr(x \leftarrow\!\!\$\, P_0; \mathcal{D}(x) = 1) - \Pr(x \leftarrow\!\!\$\, P_1; \mathcal{D}(x) = 1)|$$

is negligible. The distinguishing algorithm $\mathcal{D}$ receives input and runs the hybrid game of Game i and Game $i+1$. If the input to $\mathcal{D}$ is from P_0, $\mathcal{D}$ runs as a challenger in Game i. If the input is from P_1, $\mathcal{D}$ runs as a challenger in Game $i+1$. The algorithm $\mathcal{D}$ does not "know" which game it is running, but it uses the output of the adversary in whichever game it is running to help inform the decision it needs to make; if $\mathcal{D}$ receives input from P_0 (not knowing if it is from P_0 or P_1), it will run as a challenger in Game i and output 1 with probability $\Pr(S_i)$. Similarly, if the input is from P_1 it will run as a challenger for Game $i+1$ and output 1 with probability $\Pr(S_{i+1})$. As P_0 and P_1 are indistinguishable, this proves that $|P(S_i) - \Pr(S_{i+1})|$ is negligible.

An example of a game hop based on indistinguishability is when a group triple (g^x, g^y, g^{xy}) in Game i is replaced by (g^x, g^y, g^z) in Game $i+1$, for x, y and z chosen uniformly at random. If the adversary exhibits significantly different behaviour in each game, we can use this to create an algorithm to win the DDH game with corresponding advantage. Of course, one must show that $\mathcal{D}$ is able to simulate the hybrid game correctly.

- **Small failure events.** Here we use the Difference Lemma. Game $i+1$ proceeds identically to Game i unless a negligible failure event F occurs. When F occurs, Game $i+1$ is aborted by the challenger (and the adversary automatically loses). Therefore, $S_i \wedge \neg F \iff S_{i+1} \wedge \neg F$, so by the Difference Lemma we have

$$|\Pr(S_i) - \Pr(S_{i+1})| \leq \Pr(F)$$

for negligible event F.

An example of a hop based on a failure event is when Game $i+1$ is identical to Game i except that it halts if two honestly generated nonces are the same. This is a rare event that usually complicates matters, so a failure event here is helpful to make the analysis easier.

- **Large failure events.** This hop was formally introduced by Dent in [55]. Let F be an event that can occur during the execution of Game i such that $\Pr(\neg F)$ is non-negligible and $\Pr(S_i \wedge F) = \Pr(S_i)P(F)$. If Game $i+1$ is identical to Game i unless F occurs, in which case Game $i+1$ halts (and the adversary automatically loses), then

$$\Pr(S_{i+1}) = \Pr(S_i \wedge \neg F) = \Pr(S_i)\Pr(\neg F)$$

Thus, $\Pr(S_i)$ is non-negligible if and only if $\Pr(S_{i+1})$ is.

An example of a hop based on a large failure event is when Game i tasks the adversary, given q equally strong public keys, to forge a valid signature for one of the keys (with only one attempt). Game $i+1$ is identical to Game i except the challenger chooses a particular public key $i^* \xleftarrow{\$} \{1, 2 \ldots q\}$ unbeknownst to the adversary, and the adversary can only win if it happens to output a forgery for the public key i^*. Here the failure event F is the adversary providing an attempted forgery for a signature of a key other than i^*. In this case $\Pr(S_{i+1}) = Pr(S_i)/q$.

3.2 How to hop between games

Hopefully it can be intuitive, when looking at a protocol, what kind of game hops will be necessary in a formal security proof. For instance, if a protocol involves group elements (g^x, g^y, g^{xy}), a hash function and a pseudorandom function, it is reasonable to think that a game hopping proof will use hops exploiting the group triple (e.g. the computational, decisional or gap Diffie–Hellman assumption), as well as other hops involving the hash and pseudorandom function. Choosing the order of these hops correctly is crucial for the proof to work.

In this section, we provide a minimal possible example to explain why the utility of a game hopping proof depends on the order of the hops. Afterwards, we will explain how it is possible to know in advance the correct order of game hops to choose when compositing a proof, thus dispelling some mysticism behind the technique.

Minimal example: ElGamal encryption

ElGamal encryption is an asymmetric encryption algorithm based on Diffie–Hellman key exchange. It was first described by Taher Elgamal in 1985 [60]. ElGamal encryption is used in the free GNU Privacy Guard software, recent versions of PGP [137] and other cryptosystems. For our example, we will describe a variant of ElGamal encryption exploiting an entropy smoothing family of hash functions (Section 2.6).

Let $\mathcal{H} = \{H_k\}_{k \in K}$ denote a family of entropy smoothing keyed hash functions where each H_k is a function $H_k : G \to \{0,1\}^l$. For our Hashed ElGamal scheme, we assume all parties know the same large prime q, $G = \mathbb{Z}_q$ and generator g.

The key generation algorithm works as follows:

$$x \leftarrow_\$ \mathbb{Z}_q,\ k \leftarrow_\$ K,\ \alpha \leftarrow g^x,\ \mathsf{pk} \leftarrow (\alpha, k),\ \mathsf{sk} \leftarrow x$$

For a plaintext message m, the ciphertext ψ is computed as follows:

$$y \leftarrow_\$ \mathbb{Z}_q,\ \beta \leftarrow g^y,\ \delta \leftarrow \alpha^y,\ h \leftarrow H_k(\delta),\ v \leftarrow h \oplus m,\ \psi \leftarrow (\beta, v)$$

To decrypt a ciphertext $\psi = (\beta, v)$, the computation is as follows:

$$m \leftarrow H_k(\beta^x) \oplus v$$

We now proceed with a game hopping proof of the semantic security of Hashed ElGamal in the correct way. Afterwards we will show that reversing the order of game hops is impossible. We will then explain how it is possible to know which order of hops will work in advance.

Theorem 3.2. *Hashed ElGamal is semantically secure under the DDH and ES assumptions.*

Proof. Recall that S_i denotes the event of the adversary winning Game i. In the context of the semantic security challenge, this occurs when the adversary outputs the correct guess $\hat{b} = b$ in Game i.

Game 0: This is the original attack game. The actions are as follows:

$$x \leftarrow_\$ \mathbb{Z}_q,\ k \leftarrow_\$ K,\ \alpha \leftarrow g^x$$
$$r \leftarrow_\$ R,\ (m_0, m_1) \leftarrow \mathcal{A}(r, \alpha, k)$$
$$b \leftarrow_\$ \{0,1\},\ y \leftarrow_\$ \mathbb{Z}_q,\ \beta \leftarrow g^y,\ \delta \leftarrow \alpha^y,\ h \leftarrow H_k(\delta),\ v \leftarrow h \oplus m_b$$
$$\hat{b} \leftarrow \mathcal{A}(r, \alpha, k, \beta, v)$$

Game 1: This is a hop based on DDH indistinguishability. Game 1 is identical to Game 0 except we compute $\delta = g^z$ for $z \leftarrow_\$ \mathbb{Z}_q$. We describe the game formally as follows (with the box highlighting the change from Game 0):

$$x \leftarrow_\$ \mathbb{Z}_q,\ k \leftarrow_\$ K,\ \alpha \leftarrow g^x$$
$$r \leftarrow_\$ R,\ (m_0, m_1) \leftarrow \mathcal{A}(r, \alpha, k)$$
$$b \leftarrow_\$ \{0,1\},\ y \leftarrow_\$ \mathbb{Z}_q,\ \beta \leftarrow g^y,\ \boxed{z \leftarrow_\$ \mathbb{Z}_q,\ \delta \leftarrow g^z},\ h \leftarrow H_k(\delta),\ v \leftarrow h \oplus m_b$$
$$\hat{b} \leftarrow \mathcal{A}(r, \alpha, k, \beta, v)$$

We claim that

$$|P(S_0) - P(S_1)| \leq \epsilon_{\text{DDH}} \tag{3.1}$$

To prove (3.1), consider the following algorithm $\mathcal{D}$, which interpolates between Game 0 and Game 1:

Algorithm $\mathcal{D}(\alpha, \beta, \delta)$:

$k \leftarrow_\$ K$

$r \leftarrow_\$ R,\ (m_0, m_1) \leftarrow \mathcal{A}(r, \alpha, k)$

$b \leftarrow_\$ \{0,1\},\ h \leftarrow H_k(\delta),\ v \leftarrow h \oplus m_b$

$\hat{b} \leftarrow \mathcal{A}(r, \alpha, k, \beta, v)$

if $b = \hat{b}$ then output 1 else output 0

That is, $\mathcal{D}$ receives a group triple (α, β, δ) and then chooses some k uniformly at random. The distinguishing algorithm $\mathcal{D}$ then uses these four values to act as a challenger to an adversary $\mathcal{A}$ in the semantic security game. If the group triple is of the form (g^x, g^y, g^{xy}), then this game is exactly Game 0 from the perspective of $\mathcal{A}$. If the group triple is of the form (g^x, g^y, g^z), then this game is exactly Game 1 from the perspective of $\mathcal{A}$. The algorithm $\mathcal{D}$ does not "know" which game it is running since it does not know the form of the group triple, but it uses the output of $\mathcal{A}$ to determine the output to produce. Therefore, if the winning probabilities of $\mathcal{A}$ in Game 0 and Game 1 are significantly different, $\mathcal{D}$ will be able to translate this into a way of solving the DDH problem. We see that

$$P(S_0) = \Pr\left(x, y \leftarrow_{\$} \mathbb{Z}_q : \mathcal{D}(g^x, g^y, g^{xy}) = 1\right)$$

and

$$P(S_1) = \Pr\left(x, y, z \leftarrow_{\$} \mathbb{Z}_q : \mathcal{D}(g^x, g^y, g^z) = 1\right)$$

So, by the DDH assumption, (3.1) follows.

Game 2: This is a hop based on the ES hash family indistinguishability. Game 2 is the same as Game 1 except we compute h by choosing it at random, rather than as a hash of g^z. We describe the game formally as follows (with the box highlighting the difference between Game 2 and Game 1):

$$x \leftarrow_{\$} \mathbb{Z}_q,\ k \leftarrow_{\$} K,\ \alpha \leftarrow g^x$$
$$r \leftarrow_{\$} R,\ (m_0, m_1) \leftarrow \mathcal{A}(r, \alpha, k)$$
$$b \leftarrow_{\$} \{0,1\},\ y \leftarrow_{\$} \mathbb{Z}_q,\ \beta \leftarrow g^y,\ z \leftarrow_{\$} \mathbb{Z}_q,\ \delta \leftarrow g^z,\ \boxed{h \leftarrow_{\$} \{0,1\}^l},\ v \leftarrow h \oplus m_b$$
$$\hat{b} \leftarrow \mathcal{A}(r, \alpha, k, \beta, v)$$

Notice that δ plays no meaningful role in Game 2 as the adversary does not even see it or interact with it at all. The challenger computes δ for no reason at all.

We claim that

$$|P(S_1) - P(S_2)| \leq \epsilon_{\text{ES}} \tag{3.2}$$

To prove (3.2), consider the following algorithm $\hat{\mathcal{D}}$, which interpolates between Game 1 and Game 2:

Algorithm $\hat{\mathcal{D}}(k, h)$:

$x \leftarrow_\$ \mathbb{Z}_q,\ \alpha \leftarrow g^x$

$r \leftarrow_\$ R,\ (m_0, m_1) \leftarrow A(r, \alpha, k)$

$b \leftarrow_\$ \{0,1\},\ y \leftarrow_\$ \mathbb{Z}_q,\ \beta \leftarrow g^y,\ v \leftarrow h \oplus m_b$

$\hat{b} \leftarrow \mathcal{A}(r, \alpha, k, \beta, v)$

if $b = \hat{b}$ then output 1 else output 0

As before, $\hat{\mathcal{D}}$ does not "know" which game it is running. From the perspective of the adversary, $\hat{\mathcal{D}}$ runs as the challenger for either Game 1 or Game 2 depending on the input it receives. We see that

$$\Pr(S_1) = \Pr(k \leftarrow_\$ K,\ \delta \leftarrow_\$ G : \hat{\mathcal{D}}(k, H_k(\delta)) = 1)$$

since, because g is a generator, selecting $z \leftarrow_\$ \mathbb{Z}_q$ and then computing $\delta = g^z$ is equivalent to just selecting $\delta \leftarrow_\$ \mathbb{Z}_q$ in the first place from the perspective of the adversary. Similarly,

$$\Pr(S_2) = \Pr(k \leftarrow_\$ K,\ h \leftarrow_\$ \{0,1\}^l : \hat{\mathcal{D}}(k, h) = 1)$$

So (3.2) follows.

Here h acts as a one-time pad so $\Pr(S_2) = 1/2$ Therefore, by the triangle inequality, $|\Pr(S_0) - 1/2| \leq \epsilon_{\text{DDH}} + \epsilon_{\text{ES}}$ as required. □

In the above proof, the first hop was DDH and the second was ES. Could we have reserved the order? Consider the following game: Let Game* be the game that would occur if we tried the ES hop first, without the DDH hop, straight after Game 0.

$x \leftarrow_\$ \mathbb{Z}_q,\ k \leftarrow_\$ K,\ \alpha \leftarrow g^x$

$r \leftarrow_\$ R,\ (m_0, m_1) \leftarrow \mathcal{A}(r, \alpha, k)$

$b \leftarrow_\$ \{0,1\},\ y \leftarrow_\$ \mathbb{Z}_q,\ \beta \leftarrow g^y,\ \boxed{\delta \leftarrow \alpha^y,\ h \leftarrow_\$ \{0,1\}^l},\ v \leftarrow h \oplus m_b$

$\hat{b} \leftarrow \mathcal{A}(r, \alpha, k, \beta, v)$

Here we now run into a problem. Any distinguisher algorithm $\mathcal{D}'$ that interpolates between Game 0 and Game* will need to simulate the hybrid of both games with a subroutine adversary $\mathcal{A}$. No efficient algorithm will be able to do this.

To see this, note that such a $\mathcal{D}'$ is given input $k \in K$ so is not allowed to choose a different k. This is not a problem. The problem is that in both Game 0 and Game* (so in the hybrid game), the subroutine adversary $\mathcal{A}$ needs input of a public key $g^x = \alpha$. The only option is for $\mathcal{D}'$ to select some x for this. Now, with random coins $r \leftarrow_\$ R$, $\mathcal{A}(r, \alpha, k)$ will yield two messages (m_0, m_1) to send to $\mathcal{D}'$ as in the semantic security game. The algorithm $\mathcal{D}'$ chooses $b \leftarrow_\$ \{0,1\}$ and decides to return the ciphertext corresponding to m_b. Now, if we are in Game*, then there is no problem. This means that $\mathcal{D}'$ received input $h \leftarrow_\$ \{0,1\}^l$. It can simulate Game* perfectly well by merely computing $\beta \leftarrow g^y$, then (rather unnecessarily) computing $\delta \leftarrow \alpha^y$, then, using the input h, computing $v = h \oplus m_b$ and sending (β, v) to $\mathcal{A}$. It can then use $\mathcal{A}$'s response to help decide the output to produce on the ES problem (e.g. output 1 if $\hat{b} = b$ as before). On the other hand—and this is where the problem occurs—if the input to $\mathcal{D}'$ is (k, h) where $h = H_k(\delta)$ for $\delta \leftarrow_\$ G$, then $\mathcal{D}'$ is unable to simulate Game 0 because it will not know which y to select so that $\psi = (\beta, h)$ is consistent as in Game 0. In other words, $\mathcal{D}'$ will not know which y to choose such that $\alpha^y = \delta$. In fact, $\mathcal{D}'$ does not even know the value of δ that produced $h = h_k(\delta)$ without inverting h, which is presumably difficult.

In short, $\mathcal{D}'$ can simulate Game* but not Game 0. Of course, if $\mathcal{D}'$ inefficiently brute forced the correct y, this would work but would be an inefficient way of solving the ES problem, so the ES bound would not apply and the game hop would not be useful.

3.3 A generic game hopping methodology

In his guide to game hopping proofs [122], Shoup even refers to guessing the correct order of game hops as a "black art". Here we argue that it does not always have to be.

Looking at the Hashed ElGamal example from the previous part, we see that the ciphertext of m_b is $\psi = (g^y, H_k(g^{xy}) \oplus m_b) = (\beta, H_k(\delta) \oplus m_b)$, which links β and δ. Now, the ES assumption regards (the hash of) $\delta \leftarrow_\$ G$, which has nothing at all to do with $g^y = \beta$. It is therefore (hopefully) intuitive that one would want to first break

the link between $g^y = \beta$ and $g^{xy} = \delta$ by substituting $\delta = g^{xy}$ with g^z, using the DDH assumption, before trying the ES hop. Note that a subtle but important fact here is that g must be a generator if selecting a group element uniformly at random is to be equivalent to selecting a z uniformly at random and then computing g^z.

More generally, terms inside a nested function $f_1(f_2(\ldots(f_n())\ldots))$ are most often changed via hopping first, followed by hops involving f_n, then f_{n-1} and so on. This is because hopping in another order would most likely run into problems when the inputs to the functions are linked to other parts of the game, i.e. "the outside world" of the hybrid game. Hopping from the inside out carefully and gradually breaks these awkward links. The links to the rest of the game need to be broken so the distinguisher algorithms can successfully and consistently simulate the hybrid games.

Here we isolate general patterns in game hopping proofs and list some advice.

1. **Hop from the inside out in nested functions.** See above.

2. **Make nonces unique.** Nonces are, by definition, numbers that are used once. They are supposed to be reasonably long such that seeing two identical nonces is a negligible event. Therefore, it makes sense to save some trouble by making all nonces unique via a small failure event hop. There is no obvious reason why this cannot be the one of the first hops.

3. **Hash collisions.** It is often useful to have a hop based on a small failure event to stop the rare case of a hash collision.

4. **Pseudorandom to random.** Similarly to the above, if a protocol involves a pseudorandom function that is supposed to be indistinguishable from a random function, then it is likely a hop based on the indistinguishablity of the pseudorandom function to a truly random function will be useful. Moreover, the hop is only likely to work if the input to the PRF is random and not linked to anything else. For example, if the protocol evaluates PRF(g^{xy}) for a DH key exchange involving g^x and g^y, it may make sense to try swapping g^{xy} for g^z via a DDH hop first.

5. **Unforgeable signatures.** Signatures are always supposed to be hard to forge, so hops based on the signatures being unforgeable are likely to appear early on in a proof. Note that in the EUF-CMA game, the adversary only has access to pk and oracle queries of signatures, so if a hop based on the EUF-CMA assumption is to occur, any adversary constructed in the proof must never know anything about sk. This can usually be enforced by some freshness predicates as we will see later.

6. **Make a target:** In AKE models, almost always the notion of security revolves around a Test oracle that meets some freshness condition. It is often useful to target a particular Test oracle via a large failure event hop early on, as then the security proof is focused on a particular oracle. That is, hop to a new game that aborts unless the challenger has guessed the Test oracle in advance. One can then try to isolate the partner oracle to the Test. Note that guessing a target in advance will induce a linear factor in the overall security reduction. However, in [7] it is shown that it is impossible to derive a tight security in some circumstances.

7. **Diffie–Hellman.** One uses the DDH assumption to substitute g^{xy} with g^z only when g^x and g^y cannot be combined with another unknown DH key g^w. For example, if Alice and Bob do an ephemeral key exchange, the g^x and g^y ephemeral keys can be swapped for DDH challenge values because there is no way Alice or Bob will need to use the g^x and g^y keys again. Even if the adversary chooses a malicious DH key g^w, there is no burden on the challenger to simulate g^{xw} or g^{yw} for unknown x, y and w.

   By contrast, the GDH assumption is precisely for these situations, so long as the simulator only has to simulate $\mathrm{KDF}(g^{xw})$ or $\mathrm{KDF}(g^{yw})$, and not g^{xw} or g^{yw} directly. For example, this can occur when Alice's long-term key g^a and Bob's ephemeral key g^y are swapped for GDH challenge values. Now, the adversary may send a malicious g^w to Alice in another session, requiring Alice to compute $\mathrm{KDF}(g^{aw}, \dots)$. But this is okay because the simulator can merely make up a

random oracle answer to this query. The simulation works because a random oracle just returns random values anyway, so any random value will satisfy the simulation. Moreover, the simulator can use the DDH oracle inside the GDH game to check if an adversary random oracle query would detect this simulation and change the random oracle response accordingly.

3.4 Game hopping theorems

Often many game hops are similar and predictable, while the crux of the proof relies on some new assumption or argument, such as the (at the time unorthodox) PRF-ODH assumption used in the proof of TLS_DHE in [78]. In this part, we alleviate this problem by proving a series of game hopping theorems, applicable to a wide range of AKE protocols. We follow the context of AKE protocols from Chapter 2.

The following hash collision theorem says that we need not ever consider the case that a collision is found in a hash function. We can apply this generic game hop at any stage of a game hopping proof.

Theorem 3.3 (Hash collision hop)**.** *If Game* 1 *is identical to Game* 0 *except the challenger aborts if, during the game, two inputs* $\hat{m} \neq m'$ *are ever computed by any party (including the adversary) such that* $\mathcal{H}(\hat{m}) = \mathcal{H}(m')$*, then*

$$|\Pr(S_0) - \Pr(S_1)| \leq \epsilon_{\mathcal{H}}$$

Proof. This is a small failure event hop. We see that $S_0 \wedge \neg F \iff S_1 \wedge \neg F$ where F is the event that Game 1 aborts. All parties including the adversary are probabilistic polynomial time Turing machines, so if any of them can cause F to occur, then there exists a probabilistic polynomial time Turing machine that can find a hash collision. □

Next we prove a theorem that generalises an argument that is frequently seen in different contexts, though the bound is often an overestimate [78]. All this theorem requires us to do is count the maximum possible number of nonces in a game. For example, in the NAXOS protocol [95] this is $\mathsf{n_P n_S}$ because each agent sends one message per session. If we do not care about an exact security bound in the security proof,

then it suffices to check that the basic property that the maximum number of possible nonces in the game is negligible compared to the size of the set they are sampled from.

Theorem 3.4 (Nonce hop)**.** *Let π be an AKE protocol where each nonce is generated by sampling from a set of size q uniformly at random. Let m denote the maximum possible possible number of nonces in the game. If Game 1 is identical to Game 0 except the challenger aborts if any honestly generated nonces in the game are not unique, then*

$$|\Pr(S_0) - \Pr(S_1)| \leq \frac{\binom{m}{2}}{q}$$

Proof. This is a hop based on a small failure event. The event being that two nonces are ever produced that happen to be identical. There are a total of m such values in the game, so $\binom{m}{2}$ possible collisions. Any two values collide with probability $1/q$; therefore, the probability of the game having two identical values is bounded above by the sum, $\frac{\binom{m}{2}}{q}$. □

The following theorem also generalises an argument that is frequently presented in different contexts [43, 50, 95]: If the session key is derived using a KDF, then under normal conditions, the only feasible attack is if the adversary somehow determines the input to the KDF.

Theorem 3.5 (KDF hop)**.** *Let π is an AKE protocol with security parameter λ that computes session keys as* $\mathrm{KDF}(\sigma)$ *for some pre-master secret σ and collision-resistant* KDF *(assume collisions never by Theorem 3.3). Let distinct non-matching sessions of π result in the same σ only with negligible probability.*[1] *Let π never use a session key in the computation of any other value. Let the freshness predicate of the AKE model rule out the trivial attack of issuing a* session-key *query on the Test session or a matching session to the Test. If Game 1 is identical to Game 0 except the challenger aborts if the adversary queries* KDF *on $\sigma^\star$, where $\sigma^\star$ is the pre-master secret of the Test session, then*

[1] This is easily achieved, for instance, when an ephemeral key or nonce is an input to the KDF. We can even rule out this altogether by using applying Theorem 3.4.

$$\Pr(S_0) \leq \Pr(S_1) + \mathrm{negl}(\lambda)$$

for negligible function $\mathrm{negl}(\lambda)$.

Note that in the theorem statement we have to be careful to avoid pathological AKE protocols such as the one that computes an initial session key k, then the next session key as $\mathrm{KDF}(k)$. In this case, the adversary could make the first session the Test and receive key k_b. It could then issue a session-key query on the next session and compare the revealed key to $\mathrm{KDF}(k_b)$. If the keys are the same, then $b = 0$, otherwise $b = 1$. The use of the session key in other computations of the protocol is the reason why key indistinguishability is not the appropriate notion to prove security of a protocol such as TLS that has a handshake phase that establishes a session key and a record layer phase that uses it [78].

Proof. The adversary must determine the bit b from the test query with non-negligible advantage. Clearly, without being allowed to reveal the Test session key directly, the adversary needs to use some information value(s) from the game (e.g. another session key, some session state values, a message, etc.,) to help inform its guess.

We can assume the worst possible case, that the adversary knows all values in the game that are used to derive the Test session key. But without being allowed to query the random oracle, these alone are clearly of no use. We also know by the theorem assumption that no values in the game exist that are derived using the Test session key. In addition, clearly no value that is derived independently of the Test session key or any value used to derive the Test session key can be of any use to the adversary. The final remaining category of values in the game is that of values that are in part derived by values that are used to derive the Test session key. Given session state memory, the only one the adversary could use is another session key value. In other words, a key replication attack. But distinct non-matching sessions have distinct pre-master secrets except with negligible probability by the theorem assumption. Therefore, the only hope is for a KDF collision with distinct inputs. However, by Theorem 3.3 we can assume this never happens. □

The following theorem makes precise the methodology of the previous section of "choosing a target". It is helpful because it allows the rest of the proof argument to centre around a known Test session.

Theorem 3.6 (Test hop)**.** *Let π be an AKE model. Let t denote the maximum possible number of oracles the adversary could select to be the Test in the game. Let Game* 1 *be identical to Game* 0 *except the challenger guesses uniformly at random the Test oracle and aborts if the guess is wrong. Let neither game allow nonce collisions. Then*

$$\Pr(S_0) \leq t \Pr(S_1)$$

For protocols with only one stage, t is equal to the number of possible $\Pi^s_{i,j}$, which is $\mathsf{n_P n_S}$. As we shall see in this thesis, some protocols have multiple stages too, where each stage of an agent's session could be the Test. In this case, $t = \mathsf{n_P n_S n_s}$ where $\mathsf{n_s}$ is the maximum possible number of stages.

Proof. This is a game hop based on a large failure event. The event is the challenger not correctly guessing the Test oracle in advance. The challenger selects one of the t possible oracles uniformly at random in advance and aborts the game if the guess happens to be wrong. □

Similarly, with the following theorem, one can ensure that there are not multiple peers matching the Test in advance. Note that it might be the case that no such peer with a matching session exists, but this theorem ensures that if such a peer does exist, it is unique and known in advance to the challenger.

Theorem 3.7 (Peer hop)**.** *Let π be an AKE protocol, where before an actor completes a session s with $s_{status} =$ `accepted`, a message is sent that contains uniquely identifying information on the peer, and when receiving such a message, it rejects the session if the identity is not theirs.*[2] *Let Game* 1 *be identical to Game* 0 *except the challenger guesses in advance a $\hat{B} \in \mathcal{P}$ uniformly at random and aborts if there exist another party $\hat{C} \neq \hat{B}$ that has a matching session to the Test. Then*

[2]For example, their identity string from $\mathcal{P}$, or their public key when modelling honest public key registration.

$$\Pr(S_0) \leq \mathsf{n_P} \Pr(S_1)$$

Proof. We must show that there can be at most one identity $\hat{B}$ that can match the Test session (noting that $\hat{B}$ may have multiple sessions that match. Note that the Test session must accept, and the transcript of messages contains some uniquely identifying peer value. If any other peer $\hat{C} \neq \hat{B}$ matches, then it would have received this value and would have rejected. If it rejects, it could not have matched. Hence, this is just a game hop based on a large failure event. □

It is possible to expand upon this theorem to guess the peer's matching session to Test (if one exists), if the peer additionally contributes freshness to the session.

Theorem 3.8 (Matching Test hop)**.** *Let π be an AKE protocol, where before an actor completes a session s with $s_{status} =$ `accepted`, a message is sent that contains uniquely identifying information on the peer, and when receiving such a message, it rejects the session if the identity is not theirs. Let π additionally include ephemeral nonces in each oracle session that are sent from both parties. Let Game 1 be identical to Game 0 except the challenger guesses in advance a $\hat{B} \in \mathcal{P}$ uniformly at random and aborts if there exist another party $\hat{C} \neq \hat{B}$ that has a matching session to the Test. Let neither game allow nonce collisions. Then*

$$\Pr(S_0) \leq \mathsf{n_P n_S} \Pr(s_1)$$

Proof. We apply Theorem 3.7. Additionally, we note that the unique nonces are part of the matching conversation, and since no nonces collide, if a matching session from a unique party $\hat{B}$ exists, it must be unique since another matching session from $\hat{B}$ would imply a nonce collision. □

We can now combine all of the above theorems into one game hopping theorem that remove the tedious game hops seen in many game hopping proofs.

Theorem 3.9 (Removing repetitive game hops)**.** *Let π is an AKE protocol with security parameter λ that computes session keys as* $\mathrm{KDF}(\sigma)$ *for some pre-master secret σ and collision-resistant* KDF*. Let π generate each nonce by sampling from a set of size q uniformly at random. Let t denote the maximum possible number of oracles the adversary could select to be the Test in the game. Let π include ephemeral nonces in the* KDF*. Let π additionally be such that, before an actor completes a session s with $s_{status} = \texttt{accepted}$, a message is sent that contains uniquely identifying information on the peer, and when receiving such a message, it rejects the session if the identity is not theirs. Let Game* 1 *be identical to Game* 0 *except the challenger guesses in advance the Test oracle and a $\hat{B} \in \mathcal{P}$ uniformly at random, and aborts if (i) a hash collision is ever computed, (ii) any two honestly generated nonces ever collide, (iii) the challenger's guess of the Test oracle is wrong or there exist another party $\hat{C} \neq \hat{B}$ that has a matching session to the Test, then*

$$\Pr(S_0) \leq \epsilon_{\mathcal{H}} + \frac{\binom{m}{2}}{q} + \mathrm{negl}(\lambda) + t\mathsf{n}_{\mathsf{P}} \Pr(S_1)$$

for negligible function $\mathrm{negl}(\lambda)$*.*

With our new understanding of game hopping proofs, we are now ready to tackle the other research questions in this thesis.

Part II

Strong

One of the reasons for the delay in writing this is that I find myself in an embarrassing position; having written ref 1 ([131]) I have come to doubt the whole theory of non-secret encryption. The trouble is that I have no proof that the method of ref 1 is genuinely secure, in other words that it has a guaranteed work-factor involved in breaking it. This may be no more serious than the analogous fact that there is no proof that any of our ordinary encryption methods are genuinely secure but the fact does still worry me. So the whole of what follows should be prefaced by "if the method is sound then...".

— *Malcolm Williamson on the classified invention of Diffie–Hellman key exchange, Thoughts on Cheaper Non-Secret Encryption, 1976 [132]*

4 Post-Compromise Security

Contents

Is it possible to establish secure communication with a party whose secrets have already been compromised? Intuitively it seems impossible to provide any kind of security guarantee in this case; if Alice wants to talk to her bank but an adversary already knows her bank's long-term private key, how can Alice be sure she is really talking to her bank and not the adversary?

This intuition is *mostly* correct: if an adversary learns the long-term key of Bob, it can run the same computation that Bob would perform, using Bob's compromised long-term key and generating new random numbers if necessary. The adversary essentially does everything that Bob would do in any honest session with Alice. Thus, it should be impossible to distinguish between Bob and the adversary impersonating Bob.

This impossibility argument applies broadly to protocols with different security models, ranging from classical AKE protocols to instant messaging, and yet these "post-compromise" adversarial scenarios are a pressing practical concern; modern secure systems are designed assuming that some compromise might eventually occur, and aim instead to limit its scope. For example, devices frequently fall temporarily under adversarial control: later-removed malware, short-term theft, and confiscation at a border crossing are all potential scenarios where an adversary might have temporary access to a device that is then returned to the original owner. In all cases, traditional formal definitions of protocol security explicitly rule out post-compromise adversaries as indefensible, even though it is not always the case. In this chapter, we aim to address this gap in the theoretical literature.

Definition (informal)**.** *A protocol between Alice and Bob provides post-compromise security (PCS) if Alice has a security guarantee about communication with Bob, even if Bob's secrets have already been compromised.*

Although the impossibility result above is correct, under some conditions, practically relevant security guarantees can still be achieved. The argument above assumes a stateless protocol, so one potential solution is for Alice and Bob to share and continually update state. With an ever-evolving shared symmetric key that continually updates with new keying material, Alice and Bob's shared key may at some point be compromised but could potentially "heal" after compromise once fresh keying material is added from the genuine Alice and Bob. If this shared state is used as authentication, then such a mechanism could perhaps "lock out" the adversary to future sessions and provide a type of post-compromise security.

Some real-world protocols already maintain "key continuity" in this fashion, but the academic literature has been unable to make any formal statements about their post-compromise security properties until now. A prominent example today is the Signal Protocol [113], as used in WhatsApp, Facebook Messanger and Google Allo amongst others. Indeed, we will use the work in this chapter to analyse Signal in Chapter 5. Other important protocols include SSH, the telephony protocol ZRTP [136]

and the online instant messaging protocol OTR [30]. These protocols all implement a key rotation scheme, whereby each communication key is used to generate a successor and then deleted. Yet despite their prominence and their similar mechanisms, there have been no formal modelling or proofs of any post-compromise security guarantees. Thus, recall the following research questions:

Research question. *Is it possible to formally model and prove a strong security property that has the intuition of "self-healing"?*

Research question. *What practical "self-healing" security properties are achievable and what mechanisms provide them?*

We go about answering these questions in this chapter. We will answer the first question in the affirmative by defining the notion of post-compromise security, which captures the intuition of a "self-healing" protocol. We will then define some protocols with particular mechanisms and prove that they do indeed achieve post-compromise security. The work in this chapter has practical implications as we will use it for the first ever analysis and proof of security of the real-world Signal protocol in Chapter 5.

4.1 Overview

In this chapter, we formally model such post-compromise security guarantees and solutions, which capture the intuition of a "self-healing" process. We show how PCS can be achieved in several scenarios and consider the practical implications of our work. We distinguish between two forms of long-term key compromise: *weak* and *total*.

Weak compromise

Weak compromise corresponds to temporary adversarial control of long-term key operations, without actual theft of the long-term key. Such a situation can occur when an adversary has temporary access to a party's hardware security module (HSM) or Trusted Platform Module (TPM). Both are physical computing devices that safeguard long-term keys, providing cryptographic operations such as signing without allowing direct access to the key. The main difference is that a TPM is a hardware chip

embedded on a motherboard, while an HSM is an external device that can be easily added to a system. In what follows, our analysis applies to either an HSM or a TPM.

It is commonly believed that if an adversary has only temporary access to an HSM but does not learn the long-term key itself, then once this access is revoked, security can still be achieved. We concretely define this notion and show that such guarantees can indeed be proven formally. However, the application of an HSM to achieve PCS is non-trivial; we comment in particular that a recent TLS 1.3 proposal [92] does not have weak PCS.

Total compromise

By contrast, total compromise is more severe, and corresponds to an adversary actually learning the long-term keys as described earlier. For this case, we formally define a mechanism to synchronise and rotate keys between communicating parties, and show that it achieves PCS even after a full compromise of the long-term key. However, as we will show, there are subtleties involved where this can go wrong. We note that in the case of TextSecure (the predecessor to Signal), although key rotation was used, other usability features broke the PCS property, meaning compromise of a long-term key was sufficient for impersonation.

Practical implications

We instantiate our informal definitions in the setting of AKE protocols and develop two new strong security models for two different threat models. We show that both of these security models can be satisfied by proposing two concrete protocol constructions and proving they are secure in the models. Our work on PCS leads to crucial insights on how post-compromise security can (and cannot) be achieved, paving the way for applications in other domains. For example, in Chapter 5 we will use our theoretical understanding of PCS to formally analyse and prove the (post-compromise) security of the Signal Protocol in a strong computational game-based model.

Chapter overview

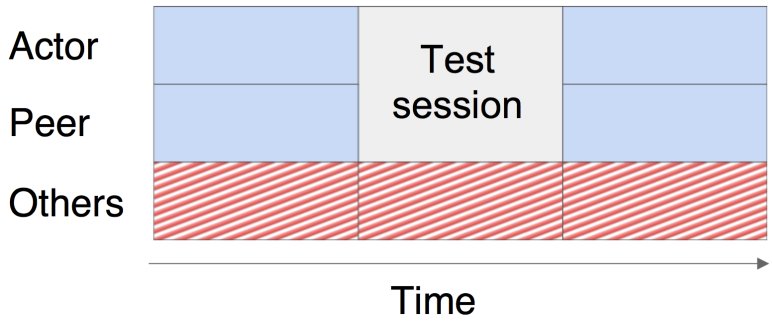
In Section 4.2 we analyse the high-level design of AKE security models and show how to incorporate PCS. In Section 4.3 we extend this intuitive view into formal definitions, constructions and security proofs. In Section 4.4 we formally define a hierarchy of different AKE models with protocols that meet them, and show how our PCS constructions can bring an additional advantage of per-peer stronger pseudorandom number generation security guarantees. We provide proofs for all our formal statements.

4.2 Intuition and the current state of the art

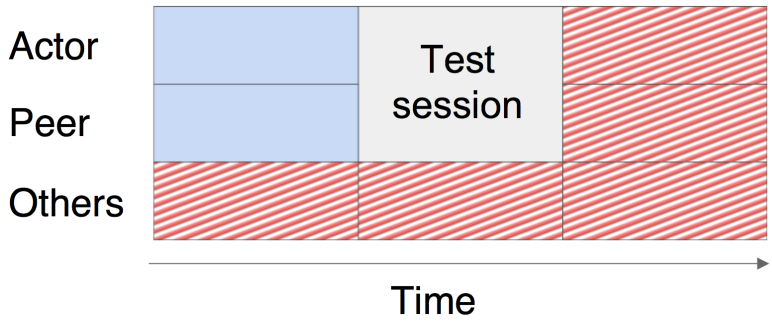
Recalling the first research question of this chapter, we turn our attention to the task of constructing a formal model for post-compromise security, which captures the intuition of self-healing. Before we do so, it makes sense to take a high-level view of the current state of the art of AKE security models. Once we have this view, we will better understand how PCS fits in.

The execution of a protocol can be abstractly depicted as in Figure 4.1. Each agent (represented by a row in the table) performs actions over time that may lead to the creation of multiple concurrent sessions. The adversary must choose a particular session of an agent to attack; recall from Chapter 2 that we call the target session the "Test" session and the particular agent of that session the "actor". During the Test session, the actor has a designated intended peer. The actor and the peer may have other sessions with each other before, after, or even during, the Test session. There may also be other agents in the security model, which can also engage in multiple concurrent sessions.

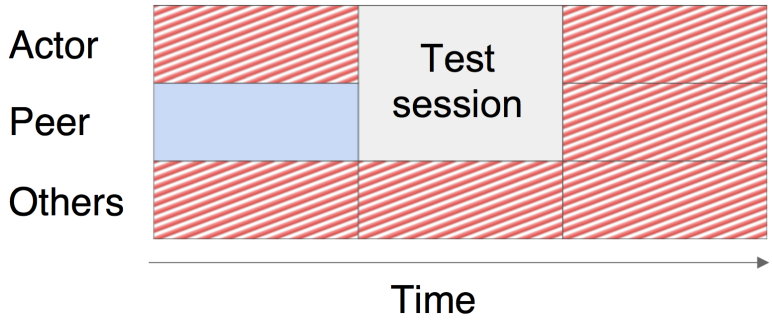
Note that the diagrams of Figure 4.1 are an abstract representation of the formal security models we define. They do not necessarily depict all of the technical details. In particular, they do not show (i) adversary actions during the Test session, though they are permitted to a limited extent based on freshness predicates, or (ii) the effects of concurrency, which mean that sessions can cross the vertical lines. We also do not distinguish in the diagrams between long-term key and ephemeral key compromise.



(a) Classical adversary model. An abstract representation of the capabilities of the adversary during an attack. If Alice is communicating with Bob, classical models allow the adversary to compromise the long-term keys of everyone except Alice and Bob. This ability to compromise others at any time is depicted in the red regions that are shaded diagonally.

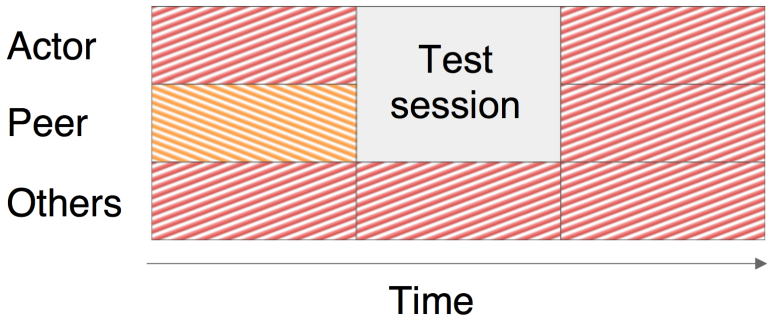


(b) Adversary model with forward secrecy. This diagram depicts the capabilities of an adversary that can also compromise all long-term private keys after the attacked Test session is complete. Protocols secure under such a threat model are said to also offer (perfect) forward secrecy.

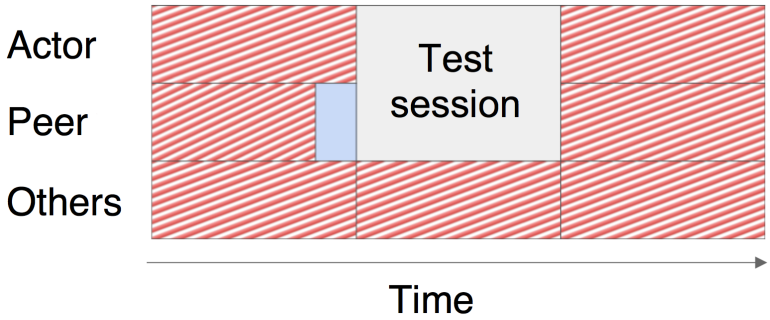


(c) Adversary model with actor key compromise. Later works also consider adversaries that can compromise the actor's long-term private key at any time. It has been shown that for some state-of-the-art protocols a secure Test session can still be achieved even in this scenario. In AKE literature, such protocols are said to offer resilience against actor key compromise or key compromise impersonation.

Figure 4.1: Existing adversary models for security protocols. Figure 4.1(a) represents a relatively weak model, while Figure 4.1(b) and Figure 4.1(c) extend it with forward secrecy and actor key compromise respectively.



(a) Adversary model with PCS through limited compromise. We represent with the orange box that is shaded reverse diagonally (the left side of the "peer" row), a limited form of compromise. This form of compromise allows the adversary some limited time to access long-term operations, yet does not reveal the long-term key itself. We call such a security property weak PCS.



(b) Adversary model with PCS through state. Another form of PCS can be achieved if the long-term key is compromised but there is at least one uncompromised session afterwards, represented by the small blue region that is not shaded diagonally (in the "peer' row just before the Test esssion). We call such a security property total PCS.

Figure 4.2: Our stronger adversary models for security protocols. Our models capture weak and total post-compromise security in Figure 4.2(a) and Figure 4.2(b) respectively.

Our high-level goal as cryptographers is to design protocols that achieve security even against a powerful adversary. So as standard we always assume that the network is fully untrusted, meaning that messages may be modified, dropped or inserted in transit. In addition, we allow the adversary access, in some circumstances, to the secret keys or random numbers of different parties. The more powers we grant to the adversary, the stronger the security model.

A classical adversary model, depicted in Figure 4.1(a), is to grant the adversary full access to any party not involved in the Test session. Security in such a model requires the protocol to limit the scope of compromise: if Charlie's key is revealed, Alice should still be able to communicate securely with Bob in a two-party conversation.

Although such a property is intuitive and seems trivial of most protocols, the famous Needham-Schroeder protocol was thought to be secure in this threat model for many years until proven otherwise by Gavin Lowe [100].

We can strengthen this model further by allowing the adversary to compromise all long-term private keys after the end of the attacked Test session [41], as depicted in Figure 4.1(b). This corresponds to (perfect) forward secrecy (PFS), whereby once a session is completed, even the long-term keys of all participants do not suffice to recover the session key.

We can also consider an adversary that compromises Alice's private key, as depicted in Figure 4.1(c). This corresponds to the notion of actor key compromise (AKC) [11]. The notion was first proposed in the AKE context as key compromise impersonation (KCI) [80], in which an adversary uses Alice's private key to impersonate as others *to Alice*.

Figure 4.1(c) depicts the current state of the art of adversarial models. It is clear that there is one more step one could take to further strengthen the model: to allow compromise in the final region. This corresponds to our concept of post-compromise security. In this chapter, we will show that it is possible to satisfy security against adversaries that have some powers even in this region. We have already seen the following assertion:

Claim (informal)**.** *Post-compromise security is not achievable after unrestricted compromise of the intended peer.*

The claim corresponds to the final region also being shaded red diagonally. The standard over-approximation is to rule out any compromise at all, as per Figure 4.1(c). To continue the colour representation, Figure 4.2(a) and Figure 4.2(b) show two ways in which a form of compromise can still be permitted without trivial wins by the adversary: we can either colour *all* of the final region *nearly red* (shaded orange reverse diagonally), or we can colour *nearly all* of the final region *red* (shaded diagonally), We develop both of these ideas and show that they naturally correspond to different kinds of PCS.

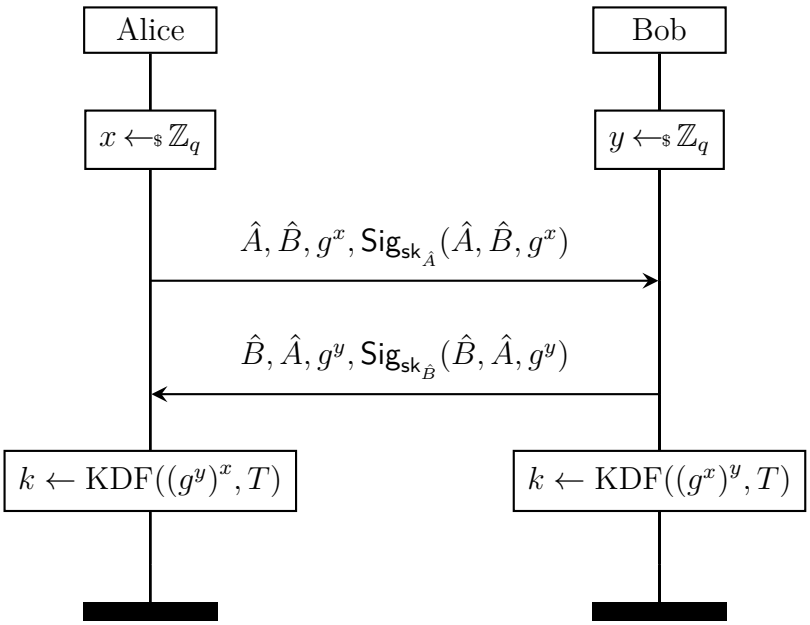


Figure 4.3: A simple signed DH protocol. The session key k is computed by applying a KDF to (g^{xy}, T), where T is the transcript of all messages sent and received during the session. This protocol has no challenge-response property to guarantee aliveness, so it does not achieve weak post-compromise security.

As a concrete running example, suppose Alice and Bob regularly authenticate each other by means of the simple signed Diffie–Hellman protocol shown in Figure 4.3.[1] Alice generates a random $x \coloneqq \mathsf{esk}_{\hat{A}}$ and sends $\mathsf{Sig}_{\mathsf{sk}_{\hat{A}}}(\hat{A}, \hat{B}, g^x)$ and Bob responds similarly. Alice's authentication in this protocol relies on sk_A not being compromised: if $\mathsf{sk}_{\hat{A}}$ is ever leaked, or even just used to sign a malicious DH value, then the authentication property is broken. Indeed, the adversary need only obtain a signature on $(\hat{A}, \hat{B}, g^z)$ for maliciously chosen z to impersonate as Alice to Bob. Post-compromise security is

[1]Note that the KDF includes the transcript T. This is to prevent the awkward "no-match" attacks of [98]. These are not attacks in the real world, but they technically constitute attacks in many security models, including our own. If the transcript was not included in the KDF, the attack would work as follows: the signature authenticates the data it signs, but nothing authenticates the signature; therefore, the adversary could change the signature in transit to still produce a valid but different signature on the same data. The sessions of Alice and Bob will now not match, but they will still compute the same session key. Since the sessions are not matching, the adversary is allowed to test Alice's session and reveal the Test session key with a session-key query on Bob's session. Requiring the signature scheme to be strongly unforgeable is not a satisfying solution to prevent this attack because in many models the adversary could have access to Alice's long-term key via a corrupt query This would allow the adversary to produce valid signatures anyway.

about in what circumstances these kinds of attacks can be prevented.

4.2.1 PCS via weak compromise

Figure 4.2(a) describes a limited form of compromise permissible against the peer of the Test session. The impersonation attack requires the adversary to use the long-term key during the Test session. If the security model allows some access to the key *without* revealing it to the adversary, and then revokes this access *during* the Test session, a PCS guarantee may still be achievable.

Claim (informal)**.** *If the adversary is not granted the key itself but is instead granted limited access to the key that is then revoked during the Test session, then a form of PCS can still be achieved.*

One pragmatic interpretation of this restriction on the adversary is temporary access to a form of trusted key storage, such as an HSM: the adversary may have temporary access to Bob's HSM at some point in time, but due to the nature of the HSM, the adversary should not have the ability to extract Bob's key. While the adversary has access to Bob's HSM, Bob is fully compromised, but if this access is then revoked, a well-designed protocol may regain its security guarantees. Another interpretation of limited compromise is a Bleichenbacher oracle [26]: Bleichenbacher's original attack requires sending numerous requests to a server and doing some computations with how the server responds. The attack allows the adversary to sign and decrypt arbitrary data with the server private key, without revealing the key itself.

Post-compromise security in this weak compromise scenario amounts to requiring that use of the long-term key depends on data from the Test session. In [79], Bleichenbacher attack is revived for TLS 1.3 and Google's Quick UDP Internet Connections (QUIC) protocol. The authors even remark that their attacks are more severe when deployed against QUIC precisely because it permits precomputation: in QUIC, signed values can be independent of the connection requests of a client.

Continuing with our running signed DH protocol example, suppose an adversary temporarily gains the ability to generate signatures with sk_A on arbitrary data. The

adversary can choose a random number z, compute a signature $\mathsf{Sig}_{\mathsf{sk}_{\hat{A}}}(\hat{A}, \hat{B}, g^z)$ and then store it for later use. At any point in the future, she can use this signature to impersonate Alice, even without the power to generate new signatures. Thus, our simple signed DH protocol does not possess this form of PCS. We shall see later (in Figure 4.4 and Theorem 4.4) that adding an appropriate nonce challenge-response to guarantee aliveness can prevent this attack. We will elaborate with formal details in Section 4.3.2.

4.2.2 PCS via state

A more subtle option, as shown in Figure 4.2(b), is to colour only some of the box; that is, to allow the long-term key to be revealed before the Test session as long as some uncompromised session has taken place. The intuition is as follows: suppose the Test session is between Alice and Bob and the adversary already knows Bob's long-term key. Suppose further that Alice and Bob have already completed a previous session, deriving some shared secret value v. If the actions in the Test session depend on v—a value from a previous session—then the adversary cannot necessarily impersonate as Bob without knowing v. Thus, one session can be used to augment the security of another.

Claim (informal)**.** *If the adversary can compromise all but one exchange of messages between Alice and Bob before the Test session, then PCS can still be achieved.*

This approach is *stateful*: data from one session is available in another. Without using a stateful protocol, even though Alice and Bob might establish a secret value unknown to the adversary, they could not use that value later for authentication purposes in the Test session. We discuss stateful protocols further in Section 4.3.3.

Continuing with our running example, we now have a scenario where the adversary knows sk_A and yet should still be prevented from impersonating Alice to Bob. To do so, we will modify the protocol so that it uses an additional "intermediate secret" to derive session keys. No longer will the long-term private key sk be sufficient to mount an impersonation attack. A secret value derived from the previous session will also be required. Now if, after a compromise, Alice and Bob manage to perform a single honest (non-compromised) session, the adversary will not have enough information to compute the subsequent session keys. We will elaborate with formal details in Section 4.3.3.

4.3 Capturing and proving post-compromise security

In this section we formalise the informal definitions and claims from Section 4.2 and provide proofs of PCS. We first define a basic security model that we will adapt throughout this chapter. Then we augment it to capture weak PCS. Finally, we adapt the model again to capture the full compromise case of total PCS.

4.3.1 Basic security model

We refer the reader to Section 2.9 for background details of AKE security models. We are now ready to specify a security model through a set of queries and a freshness predicate. The queries we consider in this chapter are in Table 4.1. The queries corrupt, randomness and cr-create are not new [51], but we include them so we can define a very strong AKE security model that not only captures current state-of-the-art security but also now PCS.

To first demonstrate the concept of AKE security models and proofs, we provide an example of a relatively weak model suitable for protocols that are not designed to resist compromised random number generators, such as the protocol of Figure 4.4. We will then extend this work to include PCS.

Definition 4.1 (Basic security model)**.** *The model X_{basic} is defined by $Q = \{$*create, send, corrupt, randomness, session-key, test, guess*$\}$ and $F = \bigwedge_{k=1}^{5} F_k$ where*

(F_1) No session-key(s) *query has been issued.*

(F_2) For all sessions $s^\star$ such that $s^\star$ matches s, no session-key$(s^\star)$ *query has been issued.*

(F_3) No randomness(s) *query has been issued.*

(F_4) For all sessions $s^\star$ such that $s^\star$ partially matches s, no randomness$(s^\star)$ *query has been issued.*

Query	Description
$\mathsf{create}(\hat{A}, r, \hat{B})$	Create a new session oracle at $\hat{A}$ with role r and peer $\hat{B}$; randomness is sampled for s_{rand} and the protocol algorithm is executed to update the state and return a message.
$\mathsf{send}(\hat{A}, i, \hat{m})$	Execute the protocol on the i^{th} session oracle of $\hat{A}$, update the state, and return the result.
$\mathsf{session\text{-}key}(\hat{A}, i)$	Reveal s_{key}, where $s = (\hat{A}, i)$.
$\mathsf{corrupt}(\hat{A})$	Reveal the long-term key of $\hat{A}$.
$\mathsf{randomness}(\hat{A}, i)$	Reveal s_{rand}, where $s = (\hat{A}, i)$.
$\mathsf{cr\text{-}create}(\hat{A}, r, \hat{B}, rnd)$	Create a session as with create, but set $s_{\text{rand}} = rnd$ instead of sampling it afresh.
$\mathsf{HSM}(\hat{A}, \dots)$	Defined in §4.3.2.
$\mathsf{test}(s)$	Flip a coin $b \leftarrow\!\!\$\ \{0,1\}$ and return k_b, where $k_1 := s_{\text{key}}$ and k_0 is defined as a string chosen uniformly at random from the set of all possible session keys.
$\mathsf{guess}(b')$	End the game. The adversary wins if $b' = b$ and loses otherwise.

Table 4.1: The set $\mathcal{Q}$ of queries available to $\mathcal{A}$ in our PCS models.

(F_5) If there exists no origin-session for session s, then no $\mathsf{corrupt}(s_{peer})$ *query has been issued before the* create *query creating session s or as long as* $s_{status} = \texttt{active}$ *and s_{recv} is empty.*

The following theorem is a relatively unsurprising result given the basic security model and protocol:

Theorem 4.2. *For an EUF-CMA signature scheme* $(\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$ *the signed DH protocol of Figure 4.4 is secure in the model X_{basic} under the random oracle and DDH assumptions.*

Proof. This will follow from Theorem 4.4. □

4.3.2 Weak compromise

To model limited access to long-term keys before the Test session, we add a new query $\mathsf{HSM}(\dots)$. This query will effectively act as a limited version of $\mathsf{corrupt}$, which the adversary can use to launch impersonation attacks. A model may additionally define a *long-term key interface* $\mathcal{I}$, which is simply a function. We write $(Q, F, \mathcal{I})$ for the model

(Q, F) with interface $\mathcal{I}$. We define the query $\mathsf{HSM}(A; \ldots)$ to return $\mathcal{I}(A, \mathsf{sk}_A, \ldots)$; that is, it invokes the long-term key interface with the secret key of its first argument, passing all other arguments through. For generality, we use the term "model" both when $\mathcal{I}$ is defined and where it is not (e.g. in other work). This is accomplished by setting $\mathcal{I} = id$ to be the identity function so that HSM is equivalent to $\mathsf{corrupt}$.

Definition 4.3. *Let* PCS-HSM *denote the trace predicate that for Test session s,*

(i) for all queries $\mathsf{HSM}(x, \ldots)$, $x = s_{peer}$*; and*

(ii) all queries $\mathsf{HSM}(x, \ldots)$ *were issued before the session s was created.*

This predicate captures that the adversary is allowed to query HSM only on the peer to the Test session, and then only before the creation of the Test session. This is a meaningful predicate: suppose we permitted HSM queries after the creation of the Test session. If security were still achievable in this case then the HSM query must not be capturing a worthwhile form of compromise, since even with access to it the adversary still cannot impersonate the peer. Moreover, permitting HSM queries against unrelated parties after the completion of the Test session does not add any additional adversarial power, since the strictly stronger $\mathsf{corrupt}$ query is also permitted on these occasions.

We can vary the security definition in our model by modifying the definition of the interface $\mathcal{I}$. Naturally, there is an impossibility result for stateless protocols in the case $\mathcal{I} = id$. A protocol is stateless when the inputs to each session are limited to the long-term key and public information, and values computed in one session are not available to others. In the case $\mathcal{I} = id$, the HSM query simply reveals the long-term key, so if a protocol carries no state across different sessions, the adversary can use the query $\mathsf{HSM}(\hat{B})$ to reveal the long-term key of the peer Bob and then simulate the computations of Bob for a successful impersonation attack.

Is there a meaningful interface $\mathcal{I}$ such that security in a model $(Q \cup \{\mathsf{HSM}\}, F \cap \text{PCS-HSM}, \mathcal{I})$ is achievable? Yes: a simple challenge-response signed DH protocol provides security in the model where $\mathcal{I}(\hat{X}, \mathsf{sk}_X, m) = \mathsf{Sig}_{\mathsf{sk}_{\hat{X}}}(m)$ is a signature with the long-term key of the target party, $\hat{X}$. That is, the adversary is given signatures

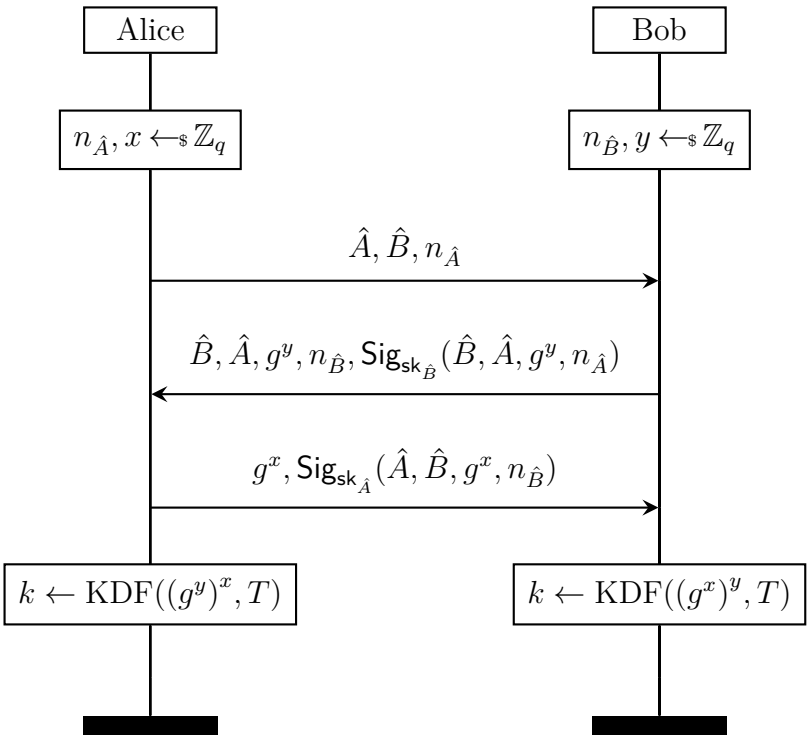


Figure 4.4: A simple signed challenge-response DH protocol. The session key k is computed by applying a KDF to (g^{xy}, T), where T is the transcript of all messages sent and received during the session. This protocol has a challenge-response property to guarantee aliveness, and it does achieve weak post-compromise security.

by Bob on its choice of messages up until creation of the Test session and yet is still unable to successfully impersonate as Bob.

Theorem 4.4. *Let* $\mathcal{I}(A, \mathsf{sk}_{\hat{A}}, m) \coloneqq \mathsf{Sig}_{\mathsf{sk}_{\hat{A}}}(m)$ *be the signature algorithm of an EUF-CMA signature scheme* $(\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$*. Let* π *be the protocol in Figure 4.4 and recall* $X_{basic} = (Q, F)$ *from Definition 4.1. The protocol* π *is secure in the model* $(Q \cup \{\mathsf{HSM}\}, F \cap \mathsf{PCS\text{-}HSM})$ *under the random oracle and DDH assumptions.*

Proof. The proof is given in Appendix A.1. □

A pragmatic interpretation of the result is as follows: if Alice's signing key is stored in an HSM and used only for the protocol in Figure 4.4, then to impersonate Alice, the adversary must have current access to the HSM (assuming the protocol primitives work correctly). Thus, the protocol of Figure 4.4 achieves weak PCS. By

contrast, the simpler signed DH protocol of Figure 4.3 is not secure in this model: the adversary only needs temporary access to the HSM to produce a signature on $(\hat{A}, \hat{B}, g^z)$ for maliciously chosen z. After this, the adversary can impersonate as Alice to Bob even after access to the HSM has been revoked.

TLS 1.3

Simply storing the long-term key in an HSM is not sufficient for PCS. For example, there have been recent discussions on the specification of the latest version of TLS: version 1.3. One proposal, OPTLS [92], has the server generate and sign temporary public keys that are then used for a signature-free handshake. Google's QUIC protocol also has this feature. After the client sends the initial Hello message and DH share, the server responds with its Hello message, a server DH share g^s, a signature on g^s, and a MAC computed over some messages. Because servers typically possess certified RSA signature keys, OPTLS allows for the signing of g^s with the signature key of the server. Thus, the server does not need to use a DH certificate. In one mode, the signature on g^s covers both g^s and a validity period through which a client can accept server authentication based on g^s. This mode saves the cost of a per-session signature, but it endows the server with the power of certification abilities. This means that a signed g^s is all that is required to impersonate the server. Thus, an adversary with temporary HSM access could impersonate as a server supporting TLS 1.3, even when the real server does not even support it.

The salient property of the protocol in Figure 4.4 that grants PCS is that access to the signing key before launching an attack on a session does not help the adversary. In OPTLS, this is not the case: these pre-signed keys effectively act as credentials for the server, so an adversary with temporary access to the signing key can issue fake ones that persist after access is revoked. This was briefly discussed on the TLS mailing list in the context of an adversary who can compromise only an ephemeral key [53, 90]. So while the existence of these credentials does have some clear advantages, they prevent OPTLS from satisfying PCS, even though OPTLS does provide forward secrecy.

4.3.3 Full compromise

Theorem 4.4 shows that a form of PCS can be achieved even when the adversary has restricted access to long-term keys. But what about the full access case? In the language of before, this is the case $\mathcal{I} = id$, or equivalently the case where the adversary is allowed corrupt queries to reveal long-term keys before the start of the Test session.

It is not hard to show that stateless protocols cannot meet this notion of security: an adversary can steal Bob's keys and then perform the same computations that Bob would perform. However, in this section we show that stateful protocols can resist such attacks and provide total post-compromise security. The intuition is as follows: every time Alice and Bob complete a session, they derive a session key and a intermediate secret token. The intermediate secret is then used in the key derivation for the *next* session completed between Alice and Bob. Thus, an adversary that wishes to simulate Bob's computation must have knowledge of the current intermediate secret in addition to all the other inputs used in the session key derivation. The current intermediate secret contributes to the key derivation of the next session key as well as the next intermediate secret. The shared intermediate secret continually "ratchets" forward in a chain. This means that if Alice and Bob are fully compromised, all they need is one uncompromised session to re-establish a fresh intermediate secret, and thus re-establish a security guarantee. We call such a design a *key refreshing protocol*. This is the core of our protocol transformation, which we will define. The generic transformation essentially takes a protocol as input and introduces this ratcheting mechanism. However, there are various subtleties involved in this that we will elaborate on when we introduce the transformation.

Modelling

The reasoning above is informal. To make it precise, we must define a formal security model that specifies exactly which actions the adversary is allowed to take. We proceed as follows: first, we define a minimal model, which we denote KE-PCS. The model KE-PCS *only* allows the adversary to make PCS attacks. We formally prove that KE-PCS is not satisfiable by stateless protocols. Next, we provide two examples of

adding PCS guarantees to existing models: first to a standard eCK-like definition (eCK$^{\omega}$ [50] to eCK$^{\omega}$-PCS), and second to a strong model, which we denote Ω_{AKE} for the class of all AKE protocols. Finally, we show that our models are achievable by constructing protocols that meet them.

We model full state compromise with corrupt, randomness and cr-create queries [51]. The corrupt query reveals the long-term key of a party while randomness queries reveal the randomness used in a session. Similarly, cr-create queries create a session with chosen randomness. To capture the intuitive property of an uncompromised session, which re-establishes a fresh intermediate secret, we make the following definition:

Definition 4.5 (Refreshed session)**.** *A session s is refreshed if there exists an intermediate session s' with matching session s'' such that*

(i) $s_{actor} = s'_{actor}$ *and* $s_{peer} = s'_{peer}$,

(ii) s' *and* s'' *both accept before the creation of* s,

(iii) at most one of $\mathsf{corrupt}(s'_{actor})$ *and* ($\mathsf{randomness}(s')$ *or* $\mathsf{cr\text{-}create}(s')$ *creating session* s'*) has been issued, and*

(iv) at most one of $\mathsf{corrupt}(s'_{peer})$ *and (*$\mathsf{randomness}(s'')$ *or* $\mathsf{cr\text{-}create}(s'')$ *creating session* s''*) has been issued.*

The third and fourth conditions of Definition 4.5 say that an adversary is allowed to compromise the long-term key or randomness, but it cannot compromise both. We choose this condition because in many strong protocols such as HMQV [88] or NAXOS [95] this is sufficient for the session to be uncompromised. Of course, many protocols are insecure if either is compromised, but we want to construct the most secure protocols possible.

Definition 4.6 (KE-PCS)**.** *The model* KE-PCS *is defined by* (Q, F) *where* $Q = \{\mathsf{create}, \mathsf{send}, \mathsf{corrupt}\}$ *and* $F = P_1 \wedge P_2$ *as follows:*

(P_1) *If the* corrupt *query is issued, it is against* s_{peer} *and before the creation of* s.

(P_2) *If there is no origin-session for s, and the* corrupt *query has been issued, then s is refreshed.*

The model KE-PCS captures precisely (and only) the concept of total PCS, as depicted in Figure 4.2(b).

Theorem 4.7. *No stateless protocol is secure in* KE-PCS.

Proof. The only secret inputs to the protocol algorithm of a stateless protocol are the session randomness and the long-term key. It follows that the simple corrupt-and-impersonate adversary has the same success probability as the corrupted agent. □

The KE-PCS model captures one particular aspect of protocol security. To give a strong, comprehensive model, we can extend existing state of the art AKE models to include PCS guarantees. For example, we can extend the eCK$^\omega$ model to yield a practical model for key exchange protocols that captures eCK security as well as PCS.

Definition 4.8 (eCK$^\omega$ (from [50]) and eCK$^\omega$-PCS (new))**.** *Consider the clauses defined in Definition 4.1 and Table 4.2. The models eCK$^\omega$ and eCK$^\omega$-PCS are defined by their freshness predicates as follows:*

$$F(eCK^\omega) \coloneqq F_1 \wedge F_2 \wedge F_3^{eCK^\omega} \wedge F_4^{eCK^\omega} \wedge F_5^{eCK^\omega}$$
$$F(eCK^\omega\text{-}PCS) \coloneqq F_1 \wedge F_2 \wedge F_3^{eCK^\omega} \wedge F_4^{eCK^\omega} \wedge F_5^{\mathsf{KR}}$$

Remark on strong models. The eCK and eCK$^\omega$ models differ only in the case that the adversary wishes to attack a Test session s admitting an origin-session $s^\star$ that is not a matching session for s. We use eCK$^\omega$ as our example since its presentation is closer to our own, but the same arguments apply to the more widely known eCK model. There are even stronger models than eCK$^\omega$, and we may naturally attempt to capture as many attacks as we can. In Section 4.4, we extend the work of [51] by defining strong models for whole classes of protocols. By considering PCS, we can

Name	Definition
F_1	No $\mathsf{session\text{-}key}(s)$ query has been issued.
F_2	For all sessions $s^\star$ such that $s^\star$ matches s, no $\mathsf{session\text{-}key}(s^\star)$ query has been issued.
$F_3^{\mathsf{eCK}^\omega}$	Not both queries $\mathsf{corrupt}(s_{\text{actor}})$ and ($\mathsf{randomness}(s)$ or $\mathsf{cr\text{-}create(.)}$ creating session s) have been issued.
$F_4^{\mathsf{eCK}^\omega}$	For all sessions s' such that s' is an origin-session for s, not both queries $\mathsf{corrupt}(s_{\text{peer}})$ and ($\mathsf{randomness}(s')$ or $\mathsf{cr\text{-}create(.)}$ creating session s') have been issued.
$F_5^{\mathsf{eCK}^\omega}$	If there exists no origin-session for session s, then no $\mathsf{corrupt}(s_{\text{peer}})$ query has been issued.
F_4^{SL}	For all sessions s' such that s is partially matching s', not both queries $\mathsf{corrupt}(s_{\text{peer}})$ and ($\mathsf{randomness}(s')$ or $\mathsf{cr\text{-}create(.)}$ creating session s') have been issued.
F_5^{SL}	If there exists no origin-session for session s, then no $\mathsf{corrupt}(s_{\text{peer}})$ query has been issued before the creation of session s via a (create or $\mathsf{cr\text{-}create}$) query or so long as as $s_{\text{status}} = \texttt{active}$ and $s_{\text{recv}} = \epsilon$
F_3^{ISM}	Not all queries $\mathsf{corrupt}(s_{\text{actor}})$ as well as ($\mathsf{randomness}$ or $\mathsf{cr\text{-}create}$) queries on all $\hat{s}$ with $\hat{s}_{\text{actor}} = s_{\text{actor}}$, where the query ($\mathsf{create}$ or $\mathsf{cr\text{-}create}$) creating session $\hat{s}$ occurred before or at the creation of session s, have been issued.
F_4^{ISM}	For all sessions s' where s partially matches s', not all queries $\mathsf{corrupt}(s_{\text{peer}})$ as well as ($\mathsf{randomness}$ or $\mathsf{cr\text{-}create}$) queries on all sessions $\hat{s}$ with $\hat{s}_{\text{actor}} = s'_{\text{actor}}$, where the query ($\mathsf{create}$ or $\mathsf{cr\text{-}create}$) creating session $\hat{s}$ occurred before or at the creation of session s', have been issued.
F_3^{KR}	If both queries $\mathsf{corrupt}(s_{\text{actor}})$ and ($\mathsf{randomness}(s)$ or $\mathsf{cr\text{-}create}$ creating session s) have been issued, then s is refreshed
F_4^{KR}	For all sessions s' such that s is partially matching s', if both queries $\mathsf{corrupt}(s_{\text{peer}})$ and ($\mathsf{randomness}(s')$ or $\mathsf{cr\text{-}create}$ creating session s') have been issued, then s' is refreshed.
F_5^{KR}	If there exists no origin-session for session s, and a $\mathsf{corrupt}(s_{\text{peer}})$ query has been issued before creation of session s via a query (create or $\mathsf{cr\text{-}create}$) or as long as $s_{\text{status}} = \texttt{active}$ and $s_{\text{recv}} = \epsilon$, then s is refreshed.

Table 4.2: Clauses of the freshness predicates for our AKE models. The predicates with superscript eCK^ω are from [50] while the predicates with superscript ISM and SL are from [51]. The new predicates are the ones with superscript KR. The predicate F_5^{KR} relates to post-compromise security while F_3^{KR} and F_4^{KR} relate to our generic protocol transformation potentially increasing resistance to bad randomness. We discuss how the predicates F_3^{KR} and F_4^{KR} interact with our generic PCS transformation in Appendix A.3.

construct a very strong model for the class of all AKE protocols.[2] We follow their naming scheme, whereby the model Ω_{X} is a strong model for the class X of protocols.

Definition 4.9 ($\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$ [51] and Ω_{AKE} (new))**.** *Consider the clauses defined in Definition 4.1 and Table 4.2. The models $\Omega_{AKE\cap ISM}$ and Ω_{AKE} are defined by their freshness predicates as follows:*

$$F(\Omega_{AKE\cap ISM}) := F_1 \wedge F_2 \wedge F_3^{ISM} \wedge F_4^{ISM} \wedge F_5^{SL}$$
$$F(\Omega_{AKE}) := F_1 \wedge F_2 \wedge F_3^{ISM} \wedge F_4^{ISM} \wedge F_5^{KR}$$

Here AKE denotes the class of all AKE protocols while ISM denotes the class of all protocols with initial state modification. That is, the class of all protocols that only access and update state upon creation of sessions. So $\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$ is a strong model for protocols in the class $\mathsf{AKE} \cap \mathsf{ISM}$. We show in this work that we can achieve even stronger security guarantees by introducing key ratcheting to provide PCS. In doing so, we consider protocols outside of the class ISM; thus, we define the strong model Ω_{AKE} for the class of all AKE protocols.

Rationale for the freshness conditions

The Ω models described in Definition 4.9 and Section 4.4 are constructed as follows: First, impossibility results are proven for a class of protocols, by constructing adversaries that provably succeed against all protocols in the class. Second, these adversaries are ruled out through careful choice of a freshness predicate.

For example, the predicate $F_3^{\mathrm{eCK}^\omega}$ rules out the class of adversaries that compromise both the randomness of the Test session and the long-term key of the actor, since such adversaries can always succeed against stateless protocols using these reveal queries. We take these strong models as our base models.

The new predicate F_5^{KR} encodes the PCS attacks illustrated in Figure 4.2(b), captured by $\mathsf{KE\text{-}PCS}$: if there is no origin-session for s (i.e. the adversary is attempting

[2]We are consistently careful to avoid the terminology "strongest model". Finding and proving that a model is truly the strongest possible practical model seems to be a hard problem; given a practical security model, it is usually trivial to define a stronger model that is identical in every way, except the adversary wins if the session key is a particular value, e.g. 7. Moreover, there might be more non-trivial queries that one has not thought of that could extend the model, such as $\mathsf{HSM}(A;\ldots)$.

to impersonate the peer) and the key of s_{peer} has been revealed, then there must have been a "blue area", or refreshing session, allowing security to be re-established.

The predicates P_1 and P_2 defined in KE-PCS are much more restrictive than the predicates F_j that we will use subsequently; the former rule out many more adversarial actions than is necessary for a reasonable protocol. Specifically, the P_1 and P_2 predicates limit the adversary to learning the key *only* of the peer to the Test session. This is by design: KE-PCS is not intended as a strong mode, but rather as a minimal demonstration of a model that captures PCS.

The other new predicates F_3^{KR} and F_4^{KR} capture a potential additional benefit of adding a PCS mechanism to a protocol. We will define such a mechanism next. These predicates roughly correspond to an additional protection against bad randomness. This is relatively unrelated to PCS itself but nevertheless an interesting result. Therefore, we elaborate on this in detail in Appendix A.3.

Satisfiability of the models

Defining strong security models is not worthwhile if they are not satisfiable: for a model X to be interesting we must be able to prove that some non-trivial protocol is secure in X. In Section 4.3.2 we defined a concrete protocol and proved it secure in a model that captures (weak) PCS. There, in the weak compromise case, the salient feature of the protocol construction was a challenge-response to guarantee aliveness so that old signatures cannot be used for arbitrary authentication in the future. Now, in the full compromise case, the protocol construction must ensure that state is synchronised between communicating parties and used correctly in the key derivation. This is a significantly more complex design. However, we prove that it can be achieved by a wide range of protocols by defining a generic protocol transformation $\pi \mapsto \pi^\dagger$.

Our generic transformation takes a three-message protocol π and converts it into a protocol that achieves PCS (and possibly more) by introducing a shared, evolving intermediate secret. Specifically in Theorem 4.13 we show that if π is secure in $\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$ then $\pi^\dagger$ is secure in Ω_{AKE}. We also show that if π is secure in eCK$^\omega$ then $\pi^\dagger$ is secure in eCK$^\omega$-PCS.

Let π be a protocol whose session keys are derived as $\mathrm{KDF}(\sigma)$ for some pre-master secret σ and random oracle KDF, where session keys are not used to compute any other variable. Let s denote the i^{th} completed session that has had actor $\hat{A}$ and peer $\hat{B}$, and let $IS_i^{\hat{A},\hat{B}}$ denote the i^{th} intermediate secret $\hat{A}$ has stored for $\hat{B}$ in $st_{\hat{A}}$. The protocol π is modified as follows:

- The session key $k \leftarrow \mathrm{KDF}(\sigma)$ that $\hat{A}$ computes in session s is replaced by $k \leftarrow \mathrm{KDF}(\sigma, IS_{i-1}^{\hat{A},\hat{B}}, 0)$.
- The next intermediate secret for peer $\hat{B}$ that $\hat{A}$ computes is $IS_i^{\hat{A},\hat{B}} \leftarrow \mathrm{KDF}(\sigma, IS_{i-1}^{\hat{A},\hat{B}}, 1)$.
- Upon session acceptance, $IS_i^{\hat{A},\hat{B}}$ is stored in $st_{\hat{A}}$, replacing $IS_{i-1}^{\hat{A},\hat{B}}$, which is erased.

Actor $\hat{B}$ computes intermediate secrets and session keys identically, and the initial intermediate secret $IS_0^{\hat{P}_i,\hat{P}_j}$ for any two parties $\hat{P}_i, \hat{P}_j$ is publicly set to 0.

Figure 4.5: Definition of the one-round PCS transform $\pi \mapsto \pi^*$.

As a stepping stone in defining our generic transformation, consider the following simpler transformation $\pi \to \pi^\star$ defined in Figure 4.5. It takes any protocol in ISM and merely adds an intermediate secret. The message flows stay the same. If $\hat{A}$ and $\hat{B}$ are synchronised—that is, they have the same intermediate secret for each other—then they will derive equal session keys from an honest session.

Theorem 4.10. *Let π be any AKE protocol in the class $\mathsf{AKE} \cap \mathsf{ISM}$.*

(i) If π is secure in eCK$^\omega$, then $\pi^\star$ is secure in eCK$^\omega$-PCS.

(ii) If π is secure in $\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$, then $\pi^\star$ is secure in Ω_{AKE}.

Proof. Note that this proof is only the Test session key being indistinguishable from a string chosen uniformly at random. The proof is in Appendix A.3. □

Unfortunately, the use of shared state for authentication has a number of subtleties. In particular, if Alice updates her state without explicitly authenticating her peer, she risks deleting a secret value that she needs to communicate with Bob; thus, stateful implicitly authenticated protocols with PCS can have a severe denial of service flaw.

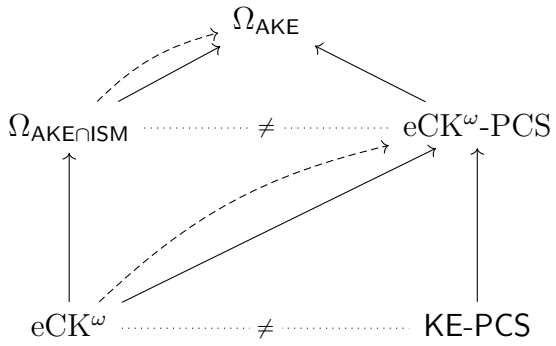


Figure 4.6: Relationship between the various models we define. The models eCK$^{\omega}$ and $\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$ are from [50] and [51] respectively. We propose the other three models in this thesis. Solid arrows denote inequality of security models ($\leq_{sec}$, Section 4.4), dotted arrows denote incomparability (neither model implies the other) and dashed arrows denote our transformation. For example, in this set Ω_{AKE} captures the strongest security guarantees.

We illustrate the situation in Figure 4.8. So blindly applying the transformation $\pi \to \pi^{\star}$ will not lead to a practical protocol.

Theorem 4.11. *No correct one-round protocol π that is secure in* KE-PCS *is post-network robust.*

Proof. In Appendix A.2. □

Theorem 4.11 says that *any* one-round protocol that achieves security in KE-PCS is vulnerable to a denial of service in which an adversary with temporary network control can prevent any two parties from ever communicating again. To deal with these denial of service issues, our final transformation $\pi \to \pi^{\dagger}$ is relatively complex. To approach this denial of service issue formally, we define the notion of *robustness* as follows:

Definition 4.12 (Robustness)**.** *Let $\mathcal{C}(\hat{A}, \hat{B})$ be the benign "correct execution" adversary that creates an initiator session at $\hat{A}$, relays its initial message to $\hat{B}$, returns the response to $\hat{A}$ and continues until $\hat{A}$ completes, and relays the final message from $\hat{A}$ to $\hat{B}$. A network adversary is any adversary that only issues* create *and* send *queries.*

A protocol π is robust *if for any $\hat{A}$ and $\hat{B}$, $\mathcal{C}$ causes a pair of matching sessions deriving the same key to be established at $\hat{A}$ with $\hat{B}$ and vice versa. It is* post-network robust *if $\mathcal{C}(\hat{A}, \hat{B})$ still induces a pair of matching sessions that derive the same session key when executed sequentially after any network adversary.*

Let π be a three-message protocol $\hat{A} \xrightarrow{m_1} \hat{B} \xrightarrow{m_2} \hat{A} \xrightarrow{m_3} \hat{B}$. Assume that public DH keys are distributed for all participants, and that π computes a pre-master secret from which session keys are derived, as in Figure 4.5. We define $\pi^\dagger$ as a modification of π as follows:

- For each agent $\hat{B}$ we define new global state variables $IS^{\hat{B},\hat{A}}$ and $st^{\mathcal{R}}_{\hat{B}}.\text{potential}_{\hat{A}}$ (for each communication partner $\hat{A}$). The first stores a single value and the second a possibly empty list of values. Initially, $IS^{\hat{A},\hat{B}}$ is set to g^{ab} for all $\hat{A}, \hat{B}$.
- In a session s between parties $\hat{A}$ and $\hat{B}$, each message $\hat{A} \xrightarrow{m_j} \hat{B}$ is replaced by $\langle m_j, \mu_j \rangle$ where $\mu_j = \mathsf{MAC}(m_j; IS^{\hat{A},\hat{B}})$, and each message $\hat{B} \xrightarrow{m'_j} \hat{A}$ is replaced by $\langle m'_j, \mu'_j \rangle$ where $\mu'_j = \mathsf{MAC}(m'_j; IS^{\hat{B},\hat{A}})$.
- Upon receiving $\langle m_1, \mu_1 \rangle$, $\hat{B}$ acts as follows:
  - If $\mu_1 = \mathsf{MAC}(m_1; IS^{\hat{B},\hat{A}})$, $\hat{B}$ continues to the next step.
  - If not, $\hat{B}$ checks for each value $is \in st^{\mathcal{R}}_{\hat{B}}.\text{potential}_{\hat{A}}$ whether $\mu_1 = \mathsf{MAC}(m_1; is)$. If this holds for some value is, $\hat{B}$ replaces $IS^{\hat{B},\hat{A}} \leftarrow is$, empties $st^{\mathcal{R}}_{\hat{B}}.\text{potential}_{\hat{A}} \leftarrow \emptyset$, and continues to the next step.
  - Otherwise, $\hat{B}$ `rejects`.
- The new value $IS^{\hat{B},\hat{A}}_{\text{new}} \leftarrow \text{KDF}(\sigma, IS^{\hat{B},\hat{A}}, 1)$ is appended to $st^{\mathcal{R}}_{\hat{B}}.\text{potential}_{\hat{A}}$.
- The pre-master secret $pms \leftarrow \text{KDF}(\sigma)$ is then replaced by $pms' \leftarrow \text{KDF}(\sigma, IS^{\hat{B},\hat{A}}, 0)$.
- $\hat{B}$ computes $\xrightarrow{m_2}$ and sends $\langle m_2, \mu_2 \rangle$
- $\hat{A}$ verifies μ_2 using its intermediate secret, and `rejects` if the verification fails. Otherwise, it updates $IS^{\hat{A},\hat{B}} \leftarrow IS^{\hat{A},\hat{B}}_{\text{new}} = \text{KDF}(\sigma, IS^{\hat{A},\hat{B}}, 1)$, and sends $\langle m_3, \mu_3 \rangle$.
- $\hat{B}$ verifies μ_3 against the potential value from the current session, and if the verification passes and it would `accept` then it first sets $IS^{\hat{B},\hat{A}} \leftarrow IS^{\hat{B},\hat{A}}_{\text{new}}$ and $st^{\mathcal{R}}_{\hat{B}}.\text{potential}_{\hat{A}} \leftarrow \emptyset$.

Figure 4.7: Definition of the PCS transform $\pi \mapsto \pi^\dagger$.

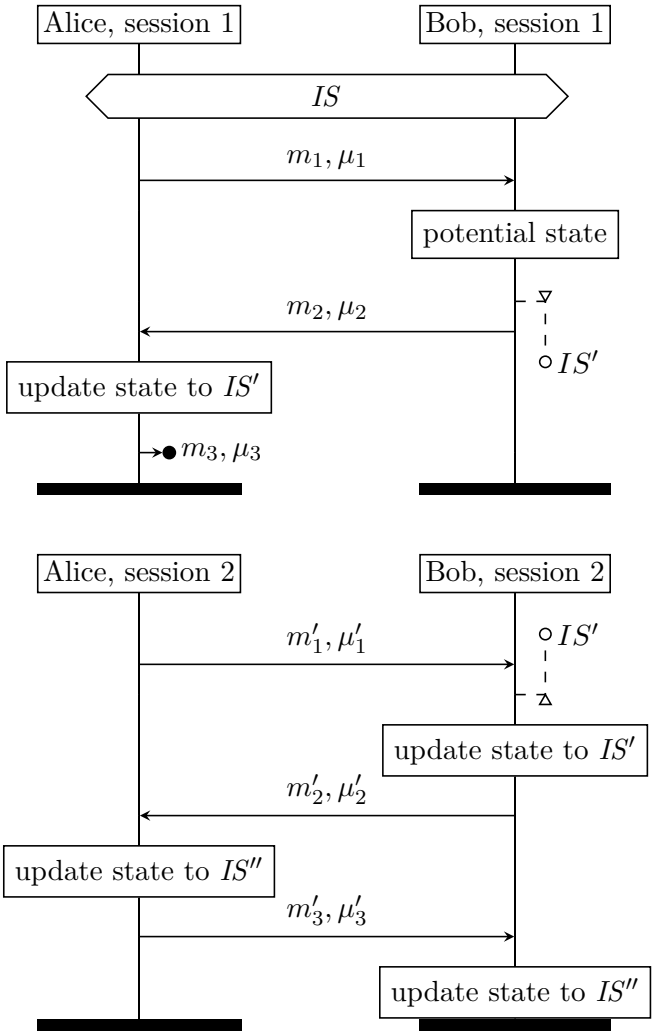


Figure 4.8: Example trace of a transformed protocol with PCS and robustness. We show two sessions between Alice and Bob: in the first, the final message is dropped so Bob does not update to IS'. In the second, Bob recalls IS' from earlier (represented by the dotted line) and performs the earlier missed update.

We give the final version of the protocol transform, which handles these and other issues, in Figure 4.7. In Figure 4.8 we show an example execution that illustrates the robustness mechanism. We refer the reader to Appendix A.3 for a full explanation of the issues and justification of the transform design. The intuition is that it adds explicit authentication (through a MAC applied to each message and requiring knowledge of the current IS), in such a way that Alice only updates an intermediate secret if she can be confident that Bob can derive the new value. The security of the modified protocol $\pi^{\dagger}$ follows from adding an intermediate secret in the KDF and

continuously updating it as sessions take place just as in $\pi^\star$.[3] Our construction additionally depends on the (weak) perfect forward secrecy of π. Because $\pi^\dagger$ also has this property, the attacker cannot compute the secrets added in the refreshed session, even if it has one of the long-term keys.

Theorem 4.13. *Let π be a stateless protocol.*

(i) If π is secure in eCK$^\omega$, then $\pi^\dagger$ is secure in eCK$^\omega$-PCS.

(ii) If π is secure in $\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$, then $\pi^\dagger$ is secure in Ω_{AKE}.

(iii) If π is robust, then $\pi^\dagger$ is post-network robust.

The proof proceeds in several stages and can be found in Appendix A.3. We first give a simpler transformation $\pi \mapsto \pi^*$ that replaces the session key derivation $K \leftarrow \mathrm{KDF}(pms)$ with $K \leftarrow \mathrm{KDF}(pms, IS, 0)$ and generates a new $IS \leftarrow \mathrm{KDF}(pms, IS, 1)$. Assuming it is hard to invert the KDF, we show that an adversary that can compute the IS output by a session can also compute the session key of that session. Next, we show that due to the chaining of IS values, it is hard for the adversary to derive the IS of a refreshed session. From these observations we can prove security of the simpler transformation in Ω_{AKE}.

Next, we consider an intermediate protocol π^*_{MAC}, that adds the MAC values μ_j to the messages but does not maintain the buffer of potential new state values. We show that the addition of the MACs does not weaken the security of the protocol, and therefore π^*_{MAC} is also secure in Ω_{AKE}. Finally, to complete the security proof we show that the transformed protocol $\pi^\dagger$ is at least as secure as π^*_{MAC}, by a direct reduction.

Results in the computational setting are generally negative, in the sense that they prove that the adversary cannot distinguish between certain values. Robustness is not a property of this form, and therefore the standard game hopping techniques are not immediately applicable. We instead construct a security argument to prove (iii), showing that certain syntactic invariants imply that parties must remain synchronised.

[3] In fact, protocols transformed from eCK$^\omega$ are secure in a stronger model than eCK$^\omega$-PCS: the intermediate secret acts as a weak entropy pool, sharing randomness across sessions. However, this weak pooling is not very practical since a stronger version is achievable with minimal overhead using pooling of randomness. This is explained further in Section 4.4.

Roughly, we prove an invariant that if one party updates the state $s \mapsto s'$ then the other party "knows" s', in the sense that s' is either its current state or present in its buffer of potential updates. By construction of the protocol algorithm, this suffices to show that the correct-execution adversary $\mathcal{C}$ indeed induces matching sessions.

Remark on detection. Of course, the fact that Alice and Bob continually evolve and update state means that if the adversary fully compromises Bob (including the current intermediate secret) and engages in a session with Alice, Bob will be permanently locked out. This ratcheting mechanism essentially forces the adversary to be a man in the middle indefinitely if it wants to avoid detection. In [105], the authors construct a variety of stateful protocols to automatically detect the misuse of secrets. The work of [105] focuses mainly on authentication through synchronisation, while PCS is about deriving strong keys. So while they are similar concepts, the work is somewhat orthogonal.

TextSecure

TextSecure is the predecessor to Signal, the protocol and app used to encrypt messages in WhatsApp, Facebook and Google Allo amongst others. TextSecure used a ratcheting mechanism very similar to the intermediate secrets mechanism above, called Axolotl. Axolotl (now renamed to the Double Ratchet) works by Alice and Bob sharing a dynamically changing "root key" rk, similar to our intermediate secrets. We describe the precise mechanism in detail in Chapter 5. Messages are encrypted with keys derived from the most recent root key. The root key synchronises state between Alice and Bob; as such, the design of Signal includes a mechanism to provide PCS.

The recently discontinued TextSecure messaging app used Axolotl; thus, one might expect it to achieve PCS. It did not, however, because of its implementation of multi-device messaging: TextSecure shared long-term keys across all devices but maintained a separate Axolotl chain for each, allowing an adversary to impersonate Bob by claiming to be a previously unseen device after a compromise.

In more detail, the design of the protocol was that each device on a given user's account stored a copy of the user's long-term key but otherwise did not share state.

For Alice to send a message to Bob she would send it separately to each of Bob's devices. Thus, an adversary that learned Bob's long-term key could register a new device on Bob's account, and Alice would accept the new device as valid since the correct identity key was used to set it up. Hence, if Bob's key is compromised then he could be impersonated to anybody. In particular, relating to the definition of PCS, the scenario would be the following: first, the adversary learns Bob's (long-term) identity key. Next, Alice and Bob have a refreshing (honest) session. If PCS were to be achieved, the adversary would now be locked out. However, the attacker can now register a new device for Bob using only the identity key. This enables the attacker to impersonate Bob again even though the session under attack has been refreshed. Thus, PCS is not achieved. As a consequence, it is not clear if the "ratcheting" mechanism in TextSecure provided any additional *security* guarantees above and beyond those of a strong AKE protocol, though it certainly provided message transport with fewer round-trips. Thus, we see that, like in the case of weak PCS for OPTLS and QUIC, although the right mechanism of ratcheting is applied in TextSecure, achieving total post-compromise security is non-trivial.

4.4 Strong models for protocol classes

In this section, we elaborate on our analysis of strong models for different classes of AKE protocols. This extends the work of [51] by incorporating PCS.

The work of [51] analyses the limits of protocols in $\mathsf{AKE} \cap \mathsf{SL}$ and (the superset) $\mathsf{AKE} \cap \mathsf{ISM}$, but it leaves out the analysis of protocols inside $\mathsf{AKE} \cap \neg\mathsf{ISM}$ as future work. Our protocol constructions that achieve PCS by synchronising state between two parties across sessions are in the class $\mathsf{AKE} \cap \neg\mathsf{ISM}$. As such, we will extend this work. Before we do, let us recap the work of [51].

Each of the defined AKE classes above admit different strong models. Attacks that can be prevented in a larger class may always work in a smaller one. For example, we have already seen the impossibility result that stateless protocols cannot achieve key indistinguishability if the long-term key of the peer to the Test session has been revealed (Theorem 4.7).

In [51, Theorem 2], five generic attacks are defined against protocols in $\mathsf{AKE} \cap \mathsf{SL}$. These attacks are then ruled out in the corresponding security model defined below (denoted $\Omega^{-}_{\mathsf{AKE}}$ in [51]). The first pair of attacks are clear: the adversary directly queries the session key of the Test session or its partner. The second pair of attacks correspond to an adversary that corrupts all secret inputs to a party and then simulates its computation. In the final attack, the adversary corrupts some party and then impersonates as them to the Test session.

Definition 4.14 ($\Omega_{\mathsf{AKE}\cap\mathsf{SL}}$ (denoted $\Omega^{-}_{\mathsf{AKE}}$ in [51]))**.** *Consider the clauses defined in Definition 4.1 and Table 4.2. The $\Omega_{AKE\cap SL}$ model is defined by freshness predicates as follows:*

$$F(\Omega_{AKE\cap SL}) \coloneqq F_1 \wedge F_2 \wedge F_3^{eCK^{\omega}} \wedge F_4^{SL} \wedge F_5^{SL}$$

A similar impossibility result, analogous to [51, Corollary 3], applies to protocols in $\mathsf{AKE} \cap \mathsf{ISM}$. Since in this class randomness can be pooled across sessions, it suffices for at least one previous session to have uncorrupted randomness. Thus, we can derive the following strong model for the class of protocols in $\mathsf{AKE} \cap \mathsf{ISM}$:

Definition 4.15 ($\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$ [51])**.** *Consider the clauses defined in Definition 4.1 and Table 4.2. The model $\Omega_{AKE\cap ISM}$ is defined by freshness predicates as follows:*

$$F(\Omega_{AKE\cap ISM}) \coloneqq F_1 \wedge F_2 \wedge F_3^{ISM} \wedge F_4^{ISM} \wedge F_5^{SL}$$

We remark that the translation from an impossibility result to a strong model is not unique. An impossibility result gives specific adversaries that can attack any protocol in a class, and a corresponding strong model rules out those adversaries but ideally as few others as possible. In particular, if an attack is not as generic as thought, the corresponding freshness predicate will rule out more adversaries than strictly necessary. For example, the PCS adversary only attacks the first session (by corrupting Bob and then attempting to impersonate as Bob to Alice in Alice's first session), but the final clause of the freshness predicate (i.e. F_5^{SL}) rules out any

compromise before the Test session. As we have seen, this is more restriction on the adversary than necessary for protocols that can achieve PCS.

Having recapped the work of [51], we are now ready to define some new strong models which we can integrate into the AKE model hierarchy of [51, Figure 4]. Note that the predicates F_3^{KR} and F_4^{KR} defined in Table 4.2.capture a bonus feature of applying our PCS transform: it can help achieve a strong form of resistance to bad randomness, although it is not as strong as pooling all randomness. We will explain this further soon.

Definition 4.16 ($\mathcal{S}_{\min}$ and $\mathcal{S}_{\mathsf{SL}}$)**.** *Consider the clauses defined in Definition 4.1 and Table 4.2. The models $\mathcal{S}_{\min}$ and $\mathcal{S}_{\mathsf{SL}}$ are defined by their freshness predicates as follows:*

$$F(\mathcal{S}_{\min}) := F_1 \wedge F_2 \wedge F_3^{\mathsf{eCK}^\omega} \wedge F_4^{\mathsf{SL}} \wedge F_5^{\mathsf{KR}}$$
$$F(\mathcal{S}_{\mathsf{SL}}) := F_1 \wedge F_2 \wedge F_3^{\mathsf{KR}} \wedge F_4^{\mathsf{KR}} \wedge F_5^{\mathsf{KR}}$$

Recall the definition of eCK$^\omega$-PCS from Definition 4.8. The difference between $F(\mathcal{S}_{\min})$ and F(eCK$^\omega$-PCS) is only in F_4: $F(\mathcal{S}_{\min})$ has F_4^{SL} while eCK$^\omega$-PCS has $F_4^{\mathrm{eCK}^\omega}$. The former restricts compromise only on sessions s' such that the Test session s *partially matches* s', while the latter restricts compromise on any s' that is an *origin-session* for s. Since a session s partially matching a session s' implies that s' is an origin-session for s, we see that $F(\text{eCK}^\omega\text{-PCS}) \implies F(\mathcal{S}_{\min})$. Likewise, one can see that $F(\mathcal{S}_{\min}) \implies F(\mathcal{S}_{\mathsf{SL}})$, since the latter again allows strictly more adversarial actions. Hence, we have the following result:

Theorem 4.17. *It holds that eCK$^\omega$-PCS $\leq_{sec} \mathcal{S}_{\min} \leq_{sec} \mathcal{S}_{\mathsf{SL}}$.*

Similarly, it is easy to see from the freshness predicates in the definition that Ω_{AKE} of Definition 4.9 is the strongest AKE model of all of this work and [51], because it has the least stringent adversarial freshness predicate in each of the five positions.

Pooling of randomness

One key difference between the above models is in the treatment of the third and fourth clauses of the freshness predicate. In $F_3^{\mathsf{eCK}^\omega}$, F_3^{SL}, $F_4^{\mathsf{eCK}^\omega}$ and F_4^{SL} these clauses describe the security of sessions where both the long-term key and local session randomness of

an agent are compromised. For example, the compromise of the long-term key and session randomness of the actor of the Test session is impossible to defend against in the class $\mathsf{AKE} \cap \mathsf{SL}$ since these are the only secret inputs to the session key computation. Thus, they are forbidden in the freshness predicates of Definition 4.14. There are, however, multiple ways to lift this restriction outside of stateless protocols.

A protocol such as NXPR [51] that uses the randomness of all past sessions (or some derived value, e.g. a hash or seeded PRNG) needs only one of these to not be compromised to maintain session key secrecy. This is the intuition behind $\Omega_{\mathsf{AKE} \cap \mathsf{ISM}}$, which uses the clauses F_3^{ISM} and F_4^{ISM}. However, the protocol only stores local secrets; the state does not help authenticate its peer, so the protocol cannot achieve total PCS.

A protocol that stores state synchronised with its peer can use that state for improved authentication. Because the state depends on previous random values, it can also provide some guarantees if the local randomness for a session is compromised. Here we are referring to $(F_3^{\mathsf{ISM}}, F_4^{\mathsf{ISM}})$ versus $(F_3^{\mathsf{KR}}, F_4^{\mathsf{KR}})$. The guarantees of just using synchronised state $(F_3^{\mathsf{KR}}, F_4^{\mathsf{KR}})$ are, conversely, weaker in two ways:

(i) Because the secret is per-peer, if the randomness of a session s with Bob is compromised, then only a previous session *with Bob* can possibly help provide secure randomness in s.

(ii) Because the peer also knows the secret, it could also be leaked *by them*.

This is the intuition behind the predicates F_3^{KR} and F_4^{KR}. Concretely, let $\mathcal{A}$ denote the adversary that honestly relays three sessions: s_1 : Alice $\leftrightarrow$ Bob, s_2 : Alice $\leftrightarrow$ Charlie and s_3 : Alice $\leftrightarrow$ Bob then queries $\mathsf{corrupt}(\text{Alice})$, $\mathsf{randomness}(s_1)$ and $\mathsf{randomness}(s_3)$. This adversary is allowed in $\Omega_{\mathsf{AKE} \cap \mathsf{ISM}}$ since there was at least one previous session with uncorrupted randomness but not allowed in $\mathcal{S}_{\mathsf{SL}}$ since that session was not with Bob. Conversely, the generic impersonation attack is valid in $\mathcal{S}_{\mathsf{SL}}$ but succeeds against any protocol in ISM.

In some sense, PCS can be seen as an orthogonal security property to pooling of randomness. This is easiest to see with the queries of Table 4.2. Of all the fifth predicates, the one that permits the most adversarial behaviour is F_5^{KR}, which is

associated with PCS. Similarly, the least stringent third and fourth predicates are F_3^{ISM} and F_4^{ISM}, which are associated with the pooling of randomness construction. So, given any model with weaker adversaries due to the third, fourth or fifth predicates, it can be extended with these ones. Moreover, there is a generic pooling of randomness transform in [51] that should be able to take a protocol that is secure in the original model and output a protocol secure in the new model with the modified F_3^{ISM} and F_4^{ISM} predicates. Similarly, our generic PCS transformation does the same for the F_5^{KR} predicate. Our transformation can possibly provide additional benefits via F_3^{KR} and F_4^{KR} if the third and fourth predicates were weaker in the original model; one can apply our PCS transformation from Figure 4.5 to additionally provide per-peer resistance to bad random number generation. We formally capture this in the following theorem:

Theorem 4.18. *Let π be any AKE protocol in the class $\mathsf{AKE} \cap \mathsf{ISM}$. If π is secure in $\Omega_{\mathsf{AKE} \cap \mathsf{SL}}$, then $\pi^\star$ is secure in $\mathcal{S}_{\mathsf{SL}}$.*

Proof. We will prove this in Appendix A.3 after the proof of Theorem 4.10 because the argument follows from the one used in the proof of Theorem 4.10. □

However, clearly per-peer resistance to bad random number generation is not as good as pooling of randomness across all sessions. This demonstrates that the models $\Omega_{\mathsf{AKE} \cap \mathsf{ISM}}$ and $\mathcal{S}_{\mathsf{SL}}$ are incomparable. On one hand, pooling of randomness gives stronger bad randomness resistance, while synchronising state helps guard against PCS impersonation attacks. But once state is shared between sessions, there is no obvious reason not to pool randomness as well as maintain a shared secret, giving stronger guarantees if the local PRNG is weak. This is how we can achieve security in our strongest model of all: Ω_{AKE}.

Part III

Realistic

You did a lot more with it than we did.

— *James Ellis to Whitfield Diffie in reference to the classified invention of public key cryptography, England, 1982 [97]*

5 Signal Protocol

Contents

In Chapter 3, we constructed new tools to help ease proofs of security. In Chapter 4, we introduced and formally defined the strong security notion of post-compromise security. With all of our new tools and understanding, we are now finally ready bring it all together to analyse the widely popular yet previously unstudied Signal messaging protocol. In doing so, we will further bridge the gap between theory and practice in the current state of the art of key exchange protocols. Recall our final research question:

Research question. *What provable security guarantees does Signal provide?*

For context and motivation, Signal is a security protocol and accompanying app from Open Whisper Systems that provides end-to-end encryption for text messaging. As of 2017, it is also being used to encrypt phone calls. The core protocol has recently been adopted by WhatsApp, Facebook Messenger and Google Allo amongst many others. WhatsApp and Facebook Messenger alone have over one billion active users. Signal shares state across sessions, uses over ten different types of keys and continually evolves state in a way that might provide (total) post-compromise security. Yet despite its importance and novelty, prior to this work there has been no academic analysis of the protocol at all.

In this chapter, we conduct the first ever security analysis of the key agreement and so-called "Double Ratchet" mechanism of Signal. We extract from the implementation a formal description of the core protocol and define a security model that can capture the ratcheting key update structure. We then prove the security of the core of Signal in our model. Our model captures several standard security properties in addition to PCS. We find no major flaws in the design and hope that our presentation and results can serve as a starting point for other analyses of this widely adopted protocol.

5.1 Overview

To achieve our goals, we first have to formally define what Signal actually is. This was more challenging than expected because at the time of this work there was no documentation for the protocol. As such, it was necessary to examine the source code directly to understand how the protocol worked.

At a high level, Signal employs a so-called Double Ratchet mechanism. This involves an *asymmetric ratchet* that is essentially Alice and Bob taking turns to send each other new Diffie–Hellman values. These DH values provide new keying material for Alice and Bob so they can continually update their shared symmetric "root key" rk. This is similar to the intermediate secrets mechanism of Chapter 4. The asymmetric ratchet was inspired by the Off-the-Record (OTR) messaging protocol [30]. The other

half of the Double Ratchet is the *symmetric ratchet*. This is essentially applying a one-way function to a "chain key" ck. Chain keys are used to derive the message keys that actually encrypt text messages. The symmetric ratchet ensures that both participants do not have to hold on to old keying material for long. The symmetric ratcheting mechanism was inspired by the Silent Circle Instant Messaging Protocol (SCIMP) [106]. The Double Ratchet is essentially a combination of both. The root key is used to derive the chain keys and the chain keys are used to derive the message keys. The intricate relations between the keys and the Double Ratchet facilitates asynchronous communication and strong security properties, which we will formally define.

Next, we provide the first formalisation of the security properties of Signal. We do this by developing a multi-stage key exchange security model with adversarial queries and freshness conditions. Compared to previous multi-stage key exchange models, which involve a single sequence of stages within each session, our model considers a *tree* of stages to model the various "chains" in Signal. Our model captures many security properties, including PCS. We subsequently prove that the cryptographic core of Signal is secure in our model, providing the first formal security guarantees for Signal.

Chapter overview

In Section 5.2 we explain in detail how Signal actually works. In Section 5.3 we informally specify our threat model, while in Section 5.4 we formally define it with adversarial queries and freshness predicates. In Section 5.5 we give our argument for security and a proof sketch. The full proof can be found in Appendix B.1.

5.2 The core of Signal

Motivation and scope

The Signal protocol uses an intricate design. Our focus is to study the existing protocol. Here we simply aim to report what Signal is, not why any of its design choices were made.

It is not entirely straightforward to pin down a precise definition of the intended usage and security properties of Signal. Our descriptions in this section were aided

by existing documentation, but the ultimate authority was the implementation itself from Open Whisper Systems [113].[1] After the time of writing, Open Whisper Systems published high-level specifications for the initial handshake in Signal (called X3DH [115]) and the Double Ratchet mechanism [114]. This helps to clarify many details; however, the codebase is still necessary to obtain a full definition. The specification also does not contain detailed definitions of the security goals of Signal.

Basic setup

The Signal protocol aims to send encrypted messages from one party to another. To do this, it assumes each party has a long-term public/private key pair, referred to as the identity key. However, since the parties might be offline at any point in time, standard authenticated key exchange solutions cannot be directly applied; using DH key exchange to achieve forward secrecy requires both parties to contribute new ephemeral DH keys, but the responding peer may be offline at the time the initiator sends a message. In the AKE literature we have considered so far, it is always assumed both parties are online at the same time to engage in sessions.

Instead of forcing users to wait until their peer is online, Signal implements an asynchronous transmission protocol by requiring potential recipients to pre-send batches of ephemeral public keys during registration or later. When the sender wishes to send a message, she obtains keys for the recipient from an intermediate server that is always online. The server is not trusted and is only used to forward messages. The sender then performs a key exchange protocol, using the long-term and ephemeral keys to compute a message encryption key.

This basic setup is then extended with the Double Ratchet mechanism for later messages in the session. New message keys are dependent on previously exchanged values between the parties. As such, "chains" of keys are formed, which, as we will see, provide interesting security properties such as PCS.

[1] The tagged releases of libsignal lag behind the current codebase. The commit hash of the state of the repository as of our reading is listed in the bibliography. Note that there are separate implementations in C, JavaScript and Java; the latter is used by Android mobile apps and is the one we have read most carefully.

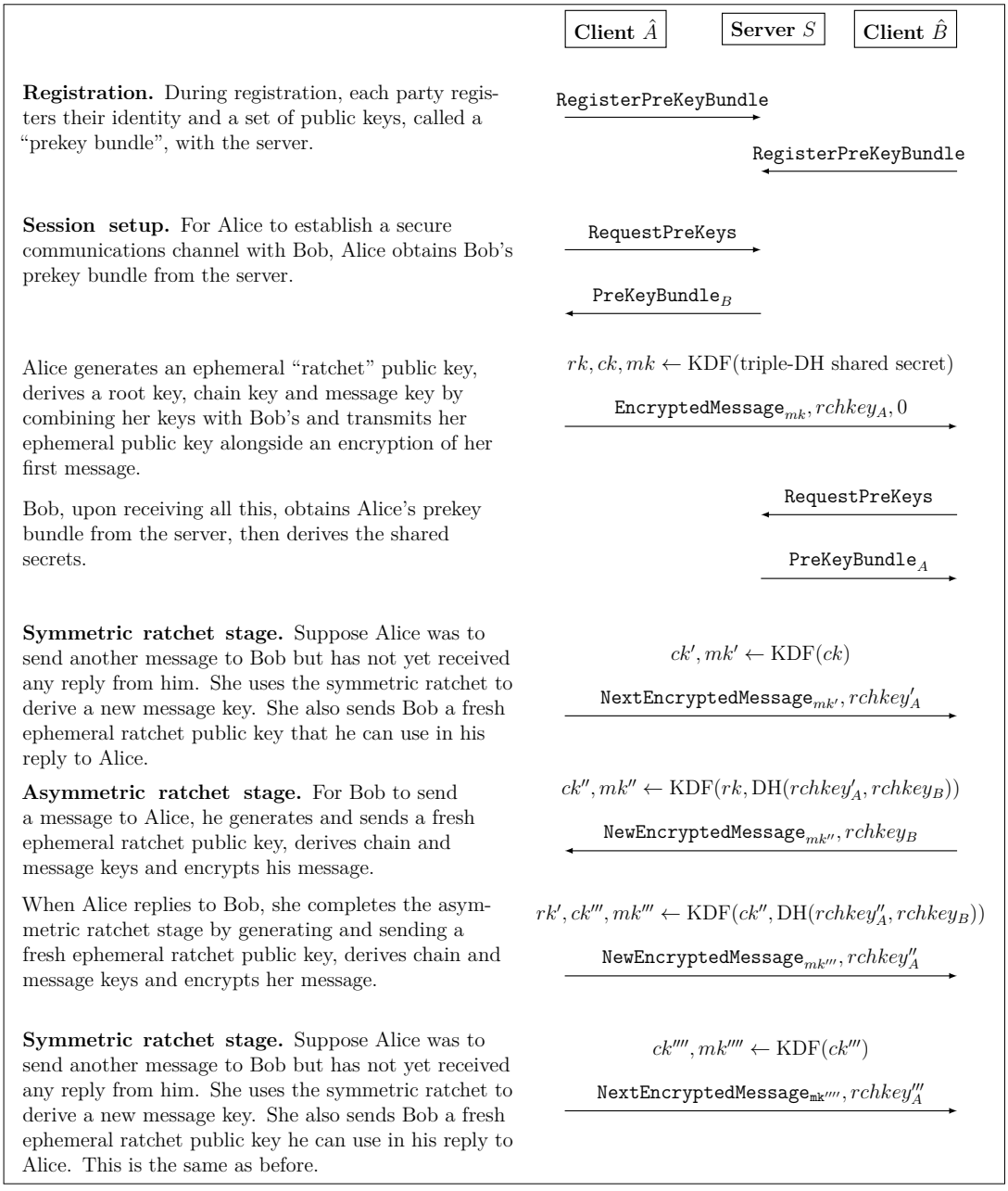


Figure 5.1: Message flow of an example Signal execution between two clients $\hat{A}$ and $\hat{B}$ via a server S. Notation and some operations have been simplified for clarity compared to later use.

5.2.1 Protocol overview

Figure 5.1 shows the message flow for an example execution of the Signal protocol. The notation in Figure 5.1 has been simplified for clarity compared to later protocol diagrams. Some operations have also been simplified. For example, several KDF applications have been collapsed to give the general idea of the protocol.

A party using Signal first registers their long-term key, as well as a medium-term key and some cached one-time keys with a key distribution server. The medium-term key is supposed to be replaced at regular intervals. A session begins with Alice requesting Bob's long-term and medium-term key as well as a single one-time key from Bob. Alice requests via a key distribution server because Bob may be offline at the time. This is what provides asynchronous communication in addition to forward secrecy. Alice then uses Bob's public keys in a proprietary key exchange protocol sometimes called the Signal Key Exchange, "TripleDH" or "X3DH". In the Signal app, WhatsApp, Facebook Messenger and other implementations of Signal, Alice and Bob have the option to (ideally over an authenticated channel), verify out-of-band that each other's keys were distributed honestly by the server. This is usually accomplished by scanning and verifying QR codes that encode session information.

The key exchange outputs a master secret. The master secret is then used to derive two symmetric keys: a "root key" and a "sending chain key". As messages are sent and received, these keys are frequently updated by passing them through a key derivation function.

When Alice wishes to encrypt a message for Bob, she advances her sending chain by one step, deriving a replacement sending chain key as well as a message encryption key. She can derive subsequent message encryption keys by repeating this process, advancing the sending chain once per message to derive a new key. Similarly, when she receives a message from Bob, she advances her receiving chain to generate a decryption key.

The root chain is advanced through a separate mechanism: when the session is initialised, Alice also generates an ephemeral DH key known as her "ratchet key". She attaches this to her messages, authenticated but not encrypted. When Bob replies to a message, he will send his own "ratchet public key". Upon receiving a new ratchet public key from Bob, Alice advances the root chain *twice*: first with the DH shared secret obtained using her old public key, and second with that using her new. The resulting two outputs of the chain initialise the new receiving and sending chains respectively, and the resulting root chain key replaces the original root chain key.

Figure 5.4 shows an example sequence of stages that one party might go through, with the message encryption keys derived in each stage. For the initiator (respectively responder), $mk^{\textbf{sym-ir}:x,y}$ denotes the y^{th} symmetric key on the x^{th} sending (respectively receiving) chain, and $mk^{\textbf{sym-ri}:x,y}$ denotes the y^{th} symmetric key on the x^{th} receiving (respectively sending) chain. We use the notation **ir** and **ri** for sending and receiving keys from the initiator of the session's perspective: **ir** is from initiator of the session to responder, so corresponds to a sending key for the initiator and receiving key for the responder, and vice versa for **ri**. The notation inherits its complexity from the underlying protocol, but it does allow us to distinctly name each session key that is generated. This notation will also allow us to make note of the subtly different properties of different keys. At a high level, we can separate the Signal protocol into four phases:

Registration. (Section 5.2.3)

At installation, Alice registers her identity with a key distribution server and uploads some cryptographic data.

Session setup. (Section 5.2.4)

Alice requests and receives cryptographic data from Bob (either from the central server or directly from Bob himself) and uses it to setup a long-lived messaging session and establish initial symmetric root and chain keys.

Symmetric ratchet communication. (Section 5.2.5)

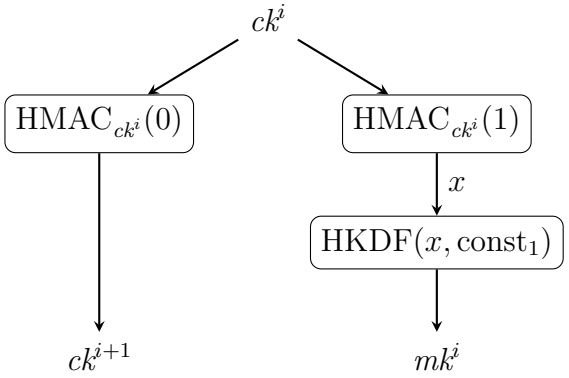
To send a message to Bob, Alice passes the current symmetric "chain key" of her session through a key derivation function to derive the next chain key as well as a message key. The message key is used to encrypt a message for Bob. This forms a KDF chain of chain keys, with a single message key branching off at each chain key in the chain. The idea behind this mechanism is that no message key can be used to derive any other message key. This means that if one message along a chain is late to arrive while subsequent messages ahead of it have already arrived, it is not so risky to hold on to the key for the late message for a long time; the chain key continues to ratchet forward.

asymmetric	$ipk_{\hat{A}}$	$ik_{\hat{A}}$	$\hat{A}$'s long-term identity key pair
	$prepk_{\hat{B}}$	$prek_{\hat{B}}$	$\hat{B}$'s medium-term (signed) prekey pair
	$eprepk_{\hat{B}}$	$eprek_{\hat{B}}$	$\hat{B}$'s ephemeral prekey pair
	$epk_{\hat{A}}$	$ek_{\hat{A}}$	$\hat{A}$'s ephemeral key pair
	$rchpk^{x}_{\hat{A}}$	$rchk^{a}_{\hat{A}}$	$\hat{A}$'s x^{th} ratchet key pair
symmetric		$ck^{\textbf{sym-ir}:x,y}_{\hat{A}}$	y^{th} chain key in $\hat{A}$'s x^{th} sending chain
		$ck^{\textbf{sym-ri}:x,y}_{\hat{A}}$	y^{th} chain key in $\hat{A}$'s x^{th} receiving chain
		$mk^{\textbf{sym-ir}:x,y}_{\hat{A}}$	y^{th} message key in $\hat{A}$'s x^{th} sending chain
		$mk^{\textbf{sym-ri}:x,y}_{\hat{A}}$	y^{th} message key in $\hat{A}$'s a^{th} receiving chain
		$rk^{a}_{\hat{A}}$	$\hat{A}$'s x^{th} root key

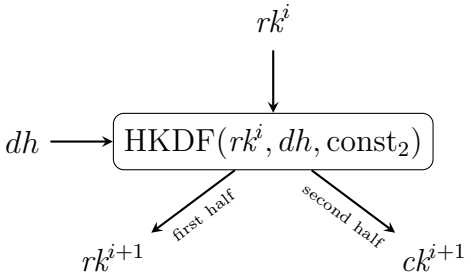
Table 5.1: Keys used in the Signal protocol. Asymmetric key pairs show public and private components.

Asymmetric ratchet updates. (Section 5.2.6)

Alice and Bob also have another chain of symmetric keys. This type of key is called the root key. Just as message keys branch off from chain keys, chains of chain keys branch off from the root key. Unlike with chain keys, the next root key is derived not only with the previous root key but also from new DH values. Alice exchanges new DH values with Bob during the session and uses the new keying material to begin new chains of chain keys. The DH shared secrets are combined with the previous root key and put through a KDF to derive the next root key. At every root key, there is a new chain of chain keys. The idea behind this mechanism is that Alice and Bob continually add new keying material to their session. This could provide a self-healing post-compromise security guarantee. In addition, the intricate tree of keys means that no chain key can be used to derive a chain key from another chain. This means that Alice and Bob can continue to advance state, and even if they hold on to the latest chain key of a particular chain just in case late messages arrive encrypted from that chain, the compromise of that chain key will only compromise message keys further down that particular chain.



(a) KDF_m, the KDF for message chain updates. Note that new chain keys are *not* computed using HKDF; instead, they use only a HMAC.



(b) KDF_r, the KDF for root chain updates. Here, dh denotes the DH secret derived from the ratchet keys used in this stage. The result is a new root key as well an output chain key.

Figure 5.2: Key derivation functions for root and chain keys in Signal: keys flow along edges and boxes apply their functions to their input. In our analysis, we treat these functions as black boxes instead of making specific assumptions. Iterating these functions produces the chains of keys in Signal.

Table 5.1 shows the different classes of keys used in the Signal protocol[2] while Figure 5.2 shows the key derivations. Figure 5.3 depicts the operations executed in all stages of the protocol in a pseudocode format.

5.2.2 Notation and primitives

Sessions. We denote $\hat{A}$'s i^{th} session by $\pi^i_{\hat{A}}$.

Stages. Within a session, Signal admits a tree of various different stages (Figure 5.4). We adopt a unified notation to refer to any of them. All stages are described using a term in [square brackets]; the initial stage is always [0]. Subsequent stages occur locally at Alice and Bob but correspond in the sense of generating matching keys.

Alice and Bob assign different roles to the stages they complete: Alice may consider some stage s as generating a sending key, while Bob considers his version of the same stage as generating a receiving key. To avoid persistent case distinctions, we adopt a *role-agnostic* naming scheme, describing stages as "**-ir**" if they are used for

[2]There are also MAC keys in Signal, but we do not consider them in our core modelling of underlying key exchange protocol; we use freshness predicates to encode if certain messages are authentic or not.

the initiator of the conversion to send to the responder, and as "**-ri**" if they are used for the responder to send to the initiator. This ensures that stages with the same name generate the same keys.

There are two types of asymmetric updates as part of the asymmetric ratchet. The first uses a received ratchet key to begin a receiving chain and the second generates a new ratchet key to begin 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, because the first asymmetric stage is of type **ri**.

Similarly, there are two types of symmetric updates: "**-ri**" and "**-ir**". The type depends on whether the chain to which they belong was created by a stage of type [**asym-ri**:x] or [**asym-ir**:x] respectively. 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].

Signal accommodates for the potential of out-of-order message delivery in the following way: for a fixed x, all symmetric stages in which a party generates sending keys in chain x occur before the asymmetric stage $x + 1$, but symmetric receiving ratchets in chain x can occur at any time after the parent node in the graph has been established. For example, the initiator performs all stages [**sym-ir**:x,1], [**sym-ir**:x,2], etc., before stage [**asym-ir**:$x + 1$] but may delay stages [**sym-ri**:x,y] as much as necessary. This means that Alice can retain any particular receiving chain for as long as she wants. Moreover, along any given chain, chain keys can be ratcheted forward to produce message keys in such a way that message keys are independent of each other, so retaining them while waiting for late messages to arrive should not compromise other messages. This means that along a receiving chain, Alice can produce message key for delayed message 2, while still symmetrically ratcheting forward to decrypt received messages 3, 4, 5, etc., safe in the knowledge that retaining message key 2 while waiting for the message to arrive should not endanger other message keys along the chain. The two key concepts of the root key producing chain keys, and the chain keys producing message keys, means that messages can arrive in arbitrary order while

Alice and Bob can continue to asymmetrically and symmetrically ratchet forward. It is not necessary for Alice or Bob to retain an old root key.

Keys. Signal distinguishes between at least ten different classes of key, depicted in Table 5.1. Again, for ease of reading, we adopt 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 $\hat{A}$ that generates a key could be ambiguous, we explicitly mark this with a subscript (e.g. $k_{\hat{A}}$). We omit the subscript where it is clear from context.

Every stage derives new keys. To identify these keys uniquely, we write the index of the stage deriving a key k in superscript. Thus, $rk^1_{\hat{A}}$ would be the root key derived by $\hat{A}$ in the initial stage, and $mk^{\textbf{sym-ri}:x,y}$ the message key derived in stage [**sym-ri**:x,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 if and only if 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 messages to send, while the responder would use the same key for decrypting received messages.

In our model, there are technically 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 5.4). 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.

Cryptographic functions. Signal uses one of two elliptic curves to implement X3DH: curve X25519 [17] or curve X448 [76]. The key derivation functions are depicted in Figure 5.2 and use either HMAC-SHA256 [13] or HKDF [89] using SHA256 as indicated.

AEAD denotes an authenticated encryption scheme with associated data [118]. In Signal, this is an encrypt-then-MAC scheme. Encryption is 256-bit AES in CBC mode with PKCS#5 padding and the MAC is HMAC-SHA256. In Signal, AEAD is used on messages. This is the same as originally used in TextSecure v3, which was shown by [62] to have standard authenticated encryption security properties. Finally, the Ed25519 signature scheme [18, 111] is used to sign the medium-term key with the long-term identity key. Since our focus is on the key exchange portion of Signal, we omit modelling of AEAD and signatures and instead capture authenticity and integrity via our freshness predicates.

5.2.3 Registration phase (Figure 5.3(a))

Upon installation, all agents generate a number of cryptographic keys and register themselves with a key distribution server. Specifically, each agent generates the following DH private keys:

(i) A long-term "identity" key *ik*.

(ii) A medium-term "signed prekey" *prek*.

(iii) Multiple "one-time prekeys" *eprek*.

The public keys corresponding to these values are then uploaded to the server, together with a signature on *prek* using *ik*. A new medium-term key and one-time prekeys are also regularly uploaded to the server. It is particularly important to always have a supply of one-time prekeys on the server, because otherwise, if they are all used up, only the peer will contribute an ephemeral key in the initial key exchange.

5.2.4 Session setup phase (Figure 5.3(b))

In the session setup phase, public keys are exchanged and used to initialise shared secrets in the session memory. The underlying key exchange protocol is a one-round

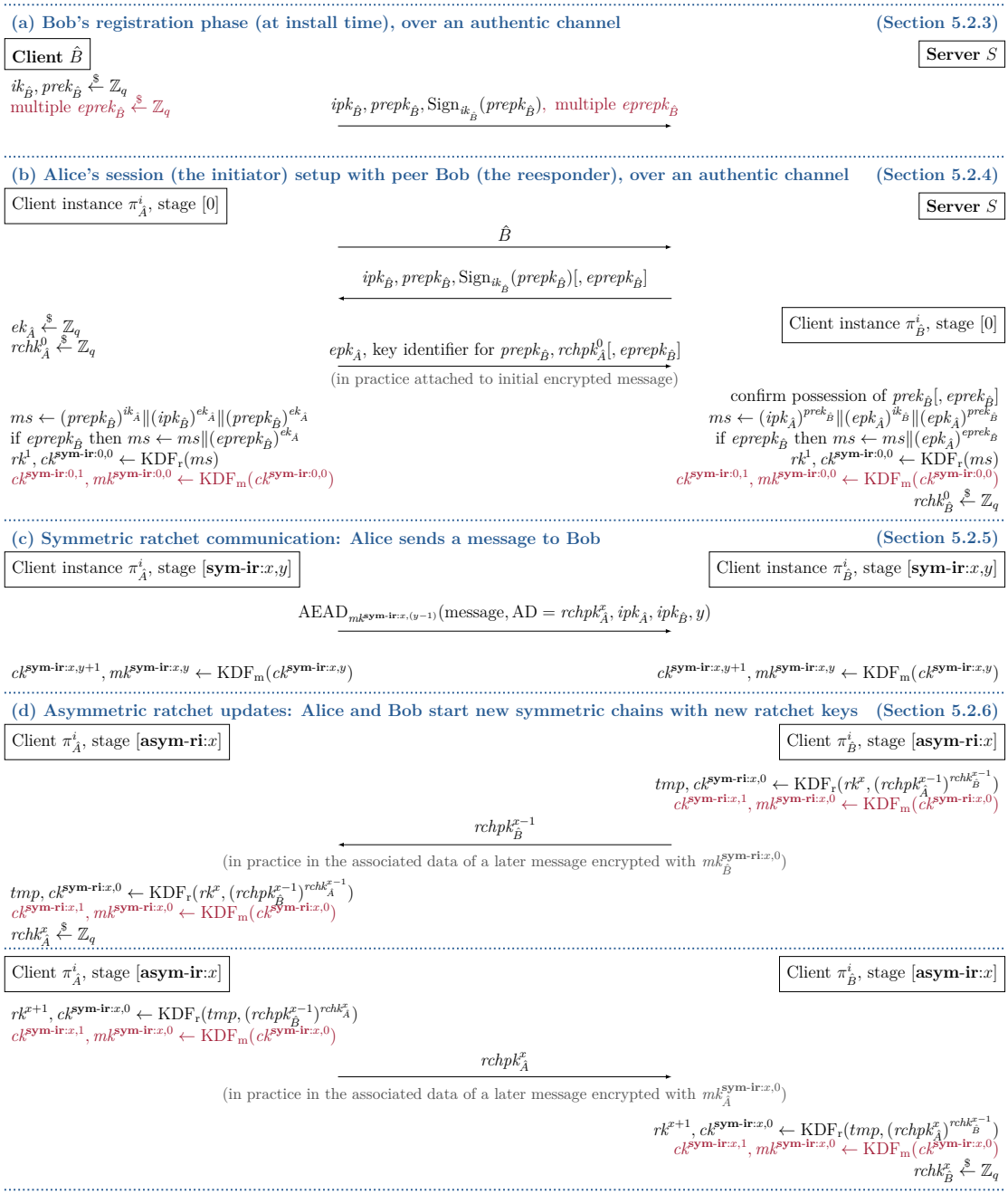


Figure 5.3: Signal protocol including preregistration of keys. Local actions are depicted in the left and right columns with message flow between them. We show only one step of the symmetric and asymmetric ratchets; they can be iterated arbitrarily. Variables storing keys are defined in Table 5.1, KDF_r and KDF_m in Figure 5.2 and session identifiers in Table 5.2. Dark red text indicates reordered actions in our model, as discussed in Section 5.5. 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.

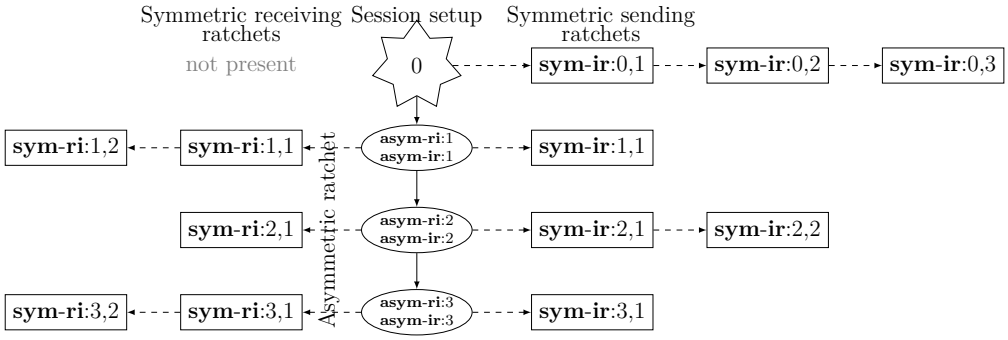


Figure 5.4: An example tree of *stages* that one party might use in one session of Signal. The content of each node is the stage name. We use **ir** to denotes stages deriving a key used to send from initiator to responder and vice versa for **ri**. In our notation, mk^s denotes the message key derived by stage s. For example, $mk^{\textbf{sym-ir}:x,y}$ denotes the y^{th} symmetric key on the x^{th} chain, used by the initiator to encrypt messages and by the responder to decrypt them. Each chain is derived by ratcheting as in Figure 5.2 with a root or chain key. For the first symmetric ratchets in a session, the initiator of the session only has a sending chain, while the responder only has a receiving chain.

DH protocol called the Signal Key Exchange, X3DH or TripleDH.[3] The key exchanges consists of an exchange of various DH public keys and computations of various DH shared secrets as shown in Figure 5.5, followed by an application of a key derivation function. While many possible variants of such protocols have been explored in depth in the literature (HMQV [88], Kudla-Paterson [93], NAXOS [95] amongst many others), the session key derivation used here is new and not based on one of these standard protocols, though it draws some inspiration from [93].

Recall that for asynchronicity, Signal uses prekeys: initial protocol messages that are stored at an intermediate server, allowing agents to establish a session with offline peers by retrieving one of their cached messages (in the form of a DH ephemeral public key). In addition to this ephemeral public key, agents also publish a "medium-term" key, that is used in multiple sessions and with different peers; the medium-term key is used in a similar way to the long-term identity key, except it is replaced on a regular basis.

[3]The key exchange protocol is sometimes referred to as TripleDH because there are always three DH shared secrets in the KDF, although in most configurations four shared secrets are actually used. Confusingly, the name QuadrupleDH has also been used for the variant that includes the long-term/long-term DH shared secret, not, as might be expected, the variant that includes the one-time ephemeral-prekey shared secret. This means that TripleDH usually has four, sometimes three, shared secrets, while QuadrupleDH usually has five, sometimes four, shared secrets, depending on whether or not the one-time ephemeral-preykey shared secret is in the KDF.

name	sid	situation
$sid[0]$	$(\texttt{triple} : ipk_i, ipk_r, prepk_r, epk_i)$	`triple`
	$(\texttt{triple+DHE} : ipk_i, ipk_r, prepk_r, epk_i, eprepk_r)$	`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 5.2: Definition of session identifiers $sid[s]$ for 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 Alice believes she is responding to Bob, then ipk_i denotes the identity public key of Bob and ipk_r denotes the identity public key of Alice. The initial asymmetric stage sid contains two ratchet keys (instead of one) since they are not used in the initial session key derivation; thus, they 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 an assymetric DH ratchet.

If the one-time ephemeral keys stored at the server are exhausted, the session will still go ahead, using only the long-term and medium-term key. This form of key reuse is studied in [103] and will be modelled in this thesis. Thus, it is important for participants to ensure that they always have a supply of fresh one-time prekeys on the Signal server.

To conclude, session setup in the Signal protocol consists of two steps: first, Alice obtains ephemeral values from Bob (usually via a key distribution server); second, Alice treats the received values as the first message of a Signal key exchange, and completes the exchange to derive a master secret.

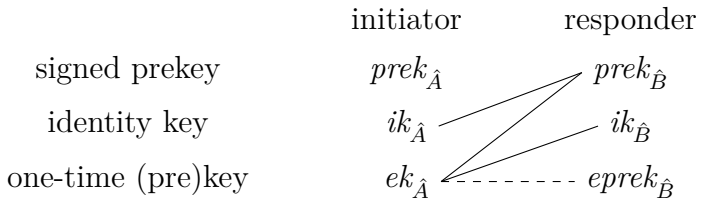


Figure 5.5: Diffie–Hellman private keys used in the Signal Key Exchange KDF. An edge between two private keys (e.g. ik_A and $prek_B$) indicates that their DH value ($g^{ik_A prek_{\hat{B}}}$) is included in the final KDF computation. The dashed line is optional: it is omitted from the session key derivation if $eprek_{\hat{B}}$ is not sent (this happens if all the uploaded prekeys on the server have been used up). The signed prekey is intended to be medium-term and used across sessions. The identity key is intended to be long-term and used across sessions. The one-time prekeys are intended to be used only once. Note the asymmetry: when Alice initiates a session with Bob, her signed prekey is not used at all. Our freshness conditions in Section 5.4.3 will be partially based on this graph.

Receiving ephemerals

The most common way for Alice to receive Bob's session-specific data is for her to query a semi-trusted server for pre-computed values (known as a PreKeyBundle). When Alice requests Bob's identity information, she receives his identity public key ipk_B, his current signed prekey $prepk_B$ and a one-time prekey $eprepk_B$ if there are any available. Signed prekeys are stored for the medium term, and therefore shared between everyone sending messages to Bob. One-time keys are deleted by the server upon transmission. Alice's initial message contains identifiers for the prekeys so that Bob can learn which were used.

Building a session

Once Alice has received the above values, she generates her own ephemeral key ek_A and computes a session key by performing three or four (three in the case that there is no one-time prekey to use) group exponentiations as depicted in Figure 5.5. She then concatenates the resulting shared secrets and passes them through a key derivation function ($\mathrm{KDF_r}$ Figure 5.2(b)) to derive an initial root key rk^1 and sending chain key $ck^{\mathbf{sym\text{-}ir}:0,0}$. For modelling purposes, we also have Alice generate her initial sending message key $mk^{\mathbf{sym\text{-}ir}:0,0}$, (which is the session key of this stage) and the

next sending chain key $ck^{\textbf{sym-ri}:0,0}$. Finally, she generates a new ephemeral DH key $rchk^0$, known as her ratchet key.

For Bob to complete[4] the key exchange, he must receive Alice's public ephemeral key epk_A. In the Signal protocol, Alice attaches this value to all messages that she sends (until she receives a message from Bob, since from such a message she can conclude that Bob received epk_A). To disentangle the stages of the model, we have Alice send epk_A in a separate message. Thus, once the session construction stage is complete, both Alice and Bob have derived their root and chain keys.

When Bob receives epk_A, he first checks that he currently knows the private keys corresponding to the identity, signed prekey and one-time prekey that Alice used. If so, he performs the receiver algorithm for the key exchange, deriving the same root key rk^1 and chain key, (which he records as $ck^{\textbf{sym-ir}:0,0}$). For modelling purposes, we also have Bob generate his initial receiving message key $mk^{\textbf{sym-ir}:0,0}$ (which is the session key output of this stage) and the next receiving chain key $ck^{\textbf{sym-ir}:0,1}$.

5.2.5 Symmetric ratchet phase (Figure 5.3(c))

Two sequences of symmetric keys will be derived using a PRF chain, one for sending and one for receiving. The symmetric chains (to the left and the right in Figure 5.4) may be advanced for one of two reasons: either Alice wishes to send a new message, or she wishes to decrypt a message she has just received.

If Alice wishes to send a new message, she takes her 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 $mk^{\textbf{sym-ir}:x,y}$. Alice uses the sending message key to encrypt her outgoing message, then deletes it and the old sending chain key. This process can be repeated arbitrarily.

When Alice receives an encrypted message, she checks the accompanying ratchet public key to confirm that she has not yet processed it, and if not she then performs an asymmetric ratchet update, described below. Regardless, she then reads the metadata

[4]If the initial message from Alice is invalid, Bob will in fact not complete a session. This does not affect our analysis, which considers only secrecy of session keys, but may become important if we were analysing a property such as deniability.

in the message header to determine the index of the message in the receiving chain, and advances the receiving chain as many steps as is necessary to derive the required receiving message key. By construction, Alice's receiving message keys equal Bob's sending keys. Unlike for the sending case, Alice cannot delete receiving message keys immediately; she must wait to receive a message encrypted under each one (otherwise, out-of-order messages would be impossible to decrypt because their keys would have been deleted). The open source implementation of Signal from Open Whisper Systems 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.

5.2.6 Asymmetric ratchet phase (Figure 5.3(d))

The final phase of Signal is the asymmetric ratchet update. In this phase, Alice and Bob take turns generating and sending new DH public keys and using them to derive new shared secrets. These are accumulated in the asymmetric ratchet chain, from which new (symmetric) message chains are initialised.

When Alice receives a message from Bob, it may be accompanied by a new (previously unseen) ratchet public key $rchpk_B^{x-1}$ if it is Bob's turn to ratchet. If so, this triggers Alice to enter her next asymmetric ratchet phase [**asym-ir**:x]. Note that Alice already has stored a previously generated private ratchet key $rchk_A^{x-1}$. Before decrypting the message, Alice updates her asymmetric ratchet as per Figure 5.3. This consists of two steps. In the first step, she computes a first DH shared secret $(rchpk_{\hat{B}}^{x-1})^{rchk_{\hat{A}}^{x-1}}$ (between the received ratchet public key and her old ratchet private key), and combines this with the root key rk^x to derive a new receiving chain key and receiving message key as well as a temporary tmp value. In the second step, she computes a second DH shared secret $(rchpk_{\hat{B}}^{x-1})^{rchk_{\hat{A}}^{x}}$ (between the received ratchet public key and her new ratchet private key), and combines this with the tmp value to derive a new sending chain key and sending message key, as well as new root key rk^{x+1} for the next asymmetric stage. The message keys in the first and second steps have slightly different security properties, so they are recorded in our model as belonging to distinct stages [**asym-ri**:x] and [**asym-ir**:x]. Alice then sends her

new ratchet public key $rchpk_A^x$ along with future messages to Bob. The ratcheting back and forth can continue indefinitely.

Bob does the corresponding operations shown in Figure 5.3 to compute the same DH shared secrets and the corresponding root, chain, and message keys. While symmetric updates can be triggered either by Alice or Bob at any time, asymmetric updates can only be triggered by receiving a previously unseen ratchet key.

5.2.7 Memory contents

There are a number of different variables in Alice's memory at any time. Alice's global state—shared between all of her sessions—contains four different collections of values: identity keys (Alice's own identity private key, and the identity public keys of all her peers), signed prekeys, ephemeral keys, and a list of all sessions.

Furthermore, session memory contains the keys used by the protocol in each session (until they are deleted). Specifically, Alice's session memory always has the current ratchet, root and chain keys. In addition, session memory may also have some number of receiving chain and message keys, corresponding to out-of-order messages not yet received from the peer.

5.3 Threat models

As with the other protocols in this thesis, we will analyse Signal in the context of an adversary that fully controls the network. The high-level properties we aim to prove are secrecy and authentication of message keys. Authentication will be implicit (only the intended party could compute the key) rather than explicit (proof that the intended party did compute the key). Forward secrecy and post-compromise security are not explicit goals; instead, derived session keys should remain secret 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) or if state is compromised and then an uncompromised asymmetric stage occurs before the Test (post-compromise security). We assume out-of-band verification of identity keys and medium-term keys,

and we do not consider side channel attacks. The formal details of our threat model are ultimately encoded in the freshness predicates, specified in Section 5.4.3.

On our choice of threat model

At the time of writing, Signal did not claim any formally specified security properties. As a result, part of our analysis included choosing a threat model. The README document accompanying the source code [113] states that Signal "is a ratcheting forward secrecy protocol that works in synchronous and asynchronous messaging environments". A separate GitHub wiki page [112] provides some more goals (forward and future secrecy,[5] 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 AEAD cipher modes.

We believe that the threat model we have chosen is realistic, although later we discuss some directions in which it could be strengthened. For example, the TLS 1.3 standard [117, Appendix D] discusses the properties we outline below (where the network is fully controlled by the adversary, and where the adversary may compromise the keys of some participants).

It is common in the authenticated key exchange literature to assume a trusted public key infrastructure (PKI), though some models allow the adversary more control [32]. In Signal, the PKI is combined with the network, in the sense that the same server distributes identity and ephemeral keys. Thus, in some scenarios, assuming a trusted PKI also restricts the control the adversary has over particular sessions.

Some claims have been made about privacy and deniability [127] in Signal, but these are relatively vague. Signatures are used but only for the signed prekey in the initial handshake, meaning that an observer can prove that Alice sent a message [56] to *someone* but perhaps not to *Bob* [49].

Additionally, one might consider a threat model that includes imperfect random number generators. Since no static-static DH shared secret is included in the KDF of Signal, an adversary that knows all ephemeral values can compute all secrets.

[5] Future secrecy is only informally defined, but it means "a leak of keys to a passive eavesdropper will be healed by introducing new DH ratchet keys" [112].

However, Signal continuously updates state with random numbers, so we capture in our threat model the fact that it is possible to make some security guarantees if some but not all random numbers are compromised.

The trust assumptions on the registration channel are not defined; Signal specifies a mandatory method for participants to verify each other's identity keys through an out-of-band channel, but most implementations do not require such verification to take place before messaging can occur. Without verification of identity keys, an untrusted key distribution server can impersonate any agent by deliberately distributing the incorrect keys.

The mechanisms of Signal suggest a lot of effort has been invested to protect against the loss of secrets used in specific communications. If the corresponding threat model is an adversary gaining (temporary) access to the device, it becomes crucial if certain previous secrets and decrypted messages can be accessed by the adversary or not. This, in turn, depends on whether *deletion* of messages and previous secrets has been effective. This is known to be a hard problem, especially on flash-based storage media [116], which are commonly used on mobile phones.

5.4 Security model

In this section, we present a security model for multi-stage key exchange. We will then use our model to analyse the initial key exchange and ratcheting mechanisms of Signal. As with other work, our model allows multiple parties to execute multiple, concurrent sessions. In addition, our model allows each session to have multiple *stages*. Other work exists on multi-stage key exchange where protocols such as QUIC have multiple stages [61]; however, the difference with our work is that stages can branch off in a tree as in Figure 5.4, rather than just appear linearly in a session.

For Signal, the session represents a single text chat between two parties. Each stage corresponds to a new message sent. Figure 5.3 depicts, roughly, a single session. There are three types of stage in Signal: the initial key exchange, asymmetric ratcheting and symmetric ratcheting. In addition, 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.

On the choice of model. As with the other work in this thesis, we choose to study the security of Signal in the traditional key exchange notion of key indistinguishability [16] (albeit a multi-stage variant), as opposed to a monolithic *secure channel* notion such as authenticated and confidential channel establishment (ACCE) [78].

It is often preferable to analyse the key exchange portion independently, and then compose this with authenticated encryption to establish a secure channel [41]; monolithic notions like ACCE are necessary for protocols that use the session key (or values derived from it) in the channel establishment itself. This prevents a clean separation between the handshake and record layers for composition. While Signal does use intermediate values between stages, the sending and receiving message keys $mk^{\textbf{sym-ir}:x,y}$, $mk^{\textbf{sym-ri}:x,y}$ are not used in subsequent key exchange operations except for MACs. We abstract out the idea of a MAC to guarantee authenticity and integrity with our freshness predicates; thus, the key indistinguishability notion of the key exchange aspect of Signal is achievable.

Another subtlety compared to the multi-stage key exchange model of [61] is that QUIC and TLS 1.3 demand a linear sequence of stages, whereas Signal has a tree of stages, as seen in Figure 5.4. This is possible because Signal does not use session keys in the channel establishment.

Finally, different implementations of Signal have different ways of handling users potentially having multiple devices. Therefore, in our model, we only consider each party having one device at a time. This means that the PCS attack on TextSecure at the end of Chapter 4 is not captured in our model.

Model notation. We use the following notational and typographic conventions: constants are denoted with `Monotype` text; serif text denotes algorithms and variables associated with the actual protocol (variables are in *italic*); 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; and we use

object-oriented notation to represent collections of variables. To denote the variable v in stage s of party u's i^{th} session, we write $\pi_u^i.v[s]$. Note that s is not a natural number; for Signal, [0] denotes the session setup stage; [**sym-ir**:x,y] and [**sym-ri**:x,y] denote the (initiator to responder) symmetric sending and receiving stages respectively; and [**asym-ri**:x] and [**asym-ir**:x] denote the first and second portions of the x^{th} asymmetric stages (from initiator to responder) respectively.

Generality of our model. Some aspects of our model are quite general, while others are very specific to Signal. Our formulation of a multi-stage key exchange protocol as a tuple of algorithms, as well as the main experiment and oracles in Figure 5.6, should be applicable to any multi-stage key exchange protocol that includes semi-static (medium-term) keys; however, our freshness definition is highly customised to Signal via our clean clauses, since we aim to precisely characterise the security properties of Signal.

The level of generality is an important decision when designing a security model for a protocol like Signal: on one hand, a general model allows analysis of and comparison to other protocols; on the other hand, it does not allow very fine-grained statements about the specific protocol under consideration. Our model is in the centre of this spectrum: we aim to keep the overall structure relatively independent of Signal, while the cleanness predicates Section 5.4.3 allow us to make fine-grained assertions about Signal specifically, which capture as much as possible.

Medium-term key authentication. The medium-term keys in Signal are signed by the same identity key used in Diffie–Hellman exponentiations. This violates the general cryptographic principle of key separation, whereby keys should only be used for one purpose only. Although there has been some analysis of this form of key reuse [54, 110], it significantly adds to the complexity of a proof.[6] We instead enforce authentication by fiat, allowing the adversary to select any of the medium-term keys

[6]The reason why in the case of Signal (Theorem 5.10) is that we use a gap Diffie–Hellman game hop where a fresh medium-term is swapped with a GDH challenge value. If the medium-term key is also used in a signature, the argument that doing this hop and answering random oracles queries in a certain way still perfectly simulates the former game falls through: the challenger does not know how to simulate the signature on the identity key. Similarly, we have an outline of a proof using a PRF-ODH assumption, which falls through for similar reasons.

owned by an agent but not to inject their own. In the game, this is implemented as an extra argument when the adversary creates a new session. Note that we do also allow the adversary to reveal medium-term keys in certain situations.

5.4.1 Multi-stage key exchange protocol

Definition 5.1 (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 *indicating the number of bits of randomness used in each session.*

The algorithms are:

- KeyGen() $\xmapsto{\$}$ (ipk, ik)*: A probabilistic long-term key generation algorithm that outputs a long-term public/secret key pair* (ipk, ik)*. In Signal, these are called "identity keys".*

- MedTermKeyGen(ik) $\xmapsto{\$}$ $(prepk, prek)$*: A probabilistic medium-term key generation algorithm that takes as input a long-term secret key ik and outputs a medium-term public/secret key pair* $(prepk, prek)$*. In Signal, these are called "signed prekeys" or "medium-term keys". In the key exchange literature, they are sometimes called "semi-static keys".*

- Activate($ik, prek, role$) $\to (\pi', m')$*: A probabilistic protocol activation algorithm that takes as input a long-term secret key ik, a medium-term secret key prek and a role* $\in \{\mathcal{I}, \mathcal{R}\}$*, and outputs a state* π' *and (possibly empty) outgoing message* m'*.*

- Run($ik, prek, \pi, m$) $\to (\pi', m')$*: A probabilistic* 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'*.*

Next, we define the state contents of a multi-stage key exchange protocol. We aim to be as generic as possible, so this will broadly follow the definition of state contents of an AKE protocol in Section 2.9. Additionally, some adaptations will be

made to accommodate a multi-stage protocol such as Signal. Recall that $\mathcal{P}$ denotes the set of parties in the model.

Definition 5.2 (Multi-stage key exchange protocol state)**.** *The session state of instance π of a multi-stage key exchange protocol is defined as a collection of the following variables:*

- $\pi.actor \in \mathcal{P}$*: the identity of the actor in the session instance.*
- $\pi.peer \in \mathcal{P}$*: the intended communication partner of the actor in the session instance.*
- $\pi.peeripk$*: the peer's long-term public key.*
- $\pi.peerprepk$*: the peer's medium-term public key.*
- $\pi.status[s] \in \{\perp, \texttt{active}, \texttt{accepted}, \texttt{rejected}\}$*: execution status for stage s. This is set to $\perp$ by default. It is set to* `active` *upon the start of a new stage. It is set to* `accepted` *or* `rejected` *upon computation of the ratchet key of the stage.*
- $\pi.k[s] \in \mathcal{K}$*: the session key output by stage s, where $\mathcal{K}$ is the key space of the protocol.*
- $\pi.st[s]$*: any additional protocol state values that a previous stage gives as input to stage s (defined as part of the protocol).*
- $\pi.sid[s]$*: the identifier of stage s of session. π. This is view the actor has of the session π in stage s, as defined in Table 5.2.*
- $\pi.type[s]$*: the type of freshness required for this stage to have security. For Signal, this is* `triple`*,* `triple+DHE`*,* `asym-ir`*,* `asym-ri`*,* `sym-ir` *or* `sym-ri`*. Freshness conditions are defined in Section 5.4.3.*

The state of an instance π in our experiment models "real" protocol state that an implementation would keep track of and use during protocol execution. We will supplement this in the experiment with additional variables that are artificially added for the experiment. These are administrative identifiers, used to formally reason about what is happening in our security experiment, e.g. to identity sessions and partners.

5.4.2 Key indistinguishability experiment

Having defined a multi-stage key exchange protocol, we can now set up the experiment for key indistinguishability, similarly to the other models in this thesis. As is typical in key exchange security models, the experiment establishes long-term keys and then allows the adversary to interact with the system. The adversary can direct parties to start sessions with particular medium-term keys, and can control the delivery of messages to parties (including modifying, dropping, delaying, and inserting messages). The adversary can learn various long-term or per-session secret information from parties via reveal queries, and at any point can choose a single stage of a single session to test. The adversary is then given either the real session (message) key from this stage, or a random key from the same key space, and asked to determine which was given. To be precise in our context of the multi-stage key exchange protocol Signal, we use Definition 5.3 below.

Before Definition 5.3, we need to recall and define the following notation: Recall that $\mathsf{n_P}$ and $\mathsf{n_S}$ denote the number of parties and maximum number of possible sessions per party respectively. Additionally, let $\mathsf{n_M}$ and $\mathsf{n_s}$ denote the maximum number of possible medium-term keys per party and the maximum number of possible stages per party respectively.

Definition 5.3 (Multi-stage key indistinguishability)**.** *Let* Π *be a multi-stage key exchange protocol. Let* $\mathcal{A}$ *be a probabilistic algorithm that runs in time polynomial in the security parameter. Define*

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

where the security experiment $\mathsf{Exp}^{\mathsf{ms\text{-}ind}}_{\Pi,\mathsf{n_P},\mathsf{n_M},\mathsf{n_S},\mathsf{n_s}}(\mathcal{A})$ *is as defined in Figure 5.6.*

Note that $\mathsf{Exp}^{\mathsf{ms\text{-}ind}}$ 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 been issued.

Furthermore, $\mathsf{Exp}^{\mathsf{ms\text{-}ind}}$ extends the per-session state π_u^i (user u's i^{th} session) with the following experiment variables:

- $\pi_u^i.\mathsf{rand}[s] \in \{0,1\}^\lambda$: random coins for π_u^i's s^{th} stage.
- $\pi_u^i.\mathsf{peerid} \in \{1, \ldots, \mathsf{n_P}\}$: the identifier of the alleged peer.
- $\pi_u^i.\mathsf{peerpreid} \in \{1, \ldots, \mathsf{n_M}\}$: the index of the alleged peer's medium-term key.
- $\pi_u^i.\mathsf{rev\_session}[s] \in \{\text{true}, \text{false}\}$: 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.

We are working in the post-specified peer model, where the peer's identity is learned by the actor during the execution of the protocol, by virtue of learning the peer's public key; and similarly for the peer's medium-term key. Certain aspects of the experiment require the administrative index of the corresponding key, and thus we assume that $\pi_u^i.\mathsf{peerid}$ is set to the corresponding index upon $\pi_u^i.\mathit{peeripk}$ being set, and similarly for the medium-term key index $\pi_u^i.\mathsf{peerpreid}$ upon $\pi_u^i.\mathit{peerprepk}$ being set.

$\mathsf{Exp}^{\text{ms-ind}}_{\Pi,\mathsf{np},\mathsf{nM},\mathsf{nS},\mathsf{ns}}(\mathcal{A})$:

1: $\mathsf{b} \xleftarrow{\$} \{0,1\}$
2: $\mathsf{tested} \leftarrow \perp$
3: *// generate long-term and semi-static keys*
4: **for** $u = 1$ **to** nP **do**
5: $\quad (ipk_u, ik_u) \xleftarrow{\$} \text{KeyGen}()$
6: $\quad$ **for** $\mathsf{preid} = 1$ **to** nM **do**
7: $\quad\quad (prepk_u^{\mathsf{preid}}, prek_u^{\mathsf{preid}}) \xleftarrow{\$} \text{MedTermKeyGen}(ik_u)$
8: $pubinfo \leftarrow (ipk_1, \ldots, ipk_{\mathsf{np}}, prepk_1^1, \ldots, prepk_{\mathsf{np}}^{\mathsf{nM}})$
9: $\mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\mathsf{Send},\mathsf{Rev}*,\mathsf{Test}}(pubinfo)$
10: **if** $(\mathsf{tested} \neq \perp) \wedge \mathsf{fresh}(\mathsf{tested}) \wedge \mathsf{b} = \mathsf{b}'$ **then**
11: $\quad$ **return** 1 *// the adversary wins*
12: **else**
13: $\quad$ **return** 0 *// the adversary loses*

$\mathsf{Send}(u, i, m)$:

1: **if** $\pi_u^i = \perp$ **then**
2: $\quad$ *// start new session and record intended peer*
3: $\quad$ parse m as $(\pi_u^i.\mathsf{preid}, role)$
4: $\quad \pi_u^i.\vec{\mathsf{rand}} \xleftarrow{\$} \{0,1\}^{\mathsf{ns} \times \lambda}$
5: $\quad (\pi_u^i, m') \leftarrow \text{Activate}(ik_u, prek_u^{\pi_u^i.\mathsf{preid}}, role; \pi_u^i.\mathsf{rand}[0])$
6: $\quad$ **return** m'
7: **else**
8: $\quad s \leftarrow \pi_u^i.stage$
9: $\quad (\pi_u^i, m') \leftarrow \text{Run}(ik_u, prek_u^{\pi_u^i.\mathsf{preid}}, \pi_u^i, m; \pi_u^i.\mathsf{rand}[s])$
10: $\quad$ **return** m'

$\mathsf{RevSessKey}(u, i, s)$:

1: $\pi_u^i.\mathsf{rev\_session}[s] \leftarrow \text{true}$
2: **return** $\pi_u^i.k[s]$

$\mathsf{RevLongTermKey}(u)$:

1: $\mathsf{rev\_ltk}_u \leftarrow \text{true}$
2: **return** ik_u

$\mathsf{RevMedTermKey}(u, \mathsf{preid})$:

1: $\mathsf{rev\_mtk}_u^{\mathsf{preid}} \leftarrow \text{true}$
2: **return** $prek_u^{\mathsf{preid}}$

$\mathsf{RevRand}(u, i, s)$:

1: $\pi_u^i.\mathsf{rev\_random}[s] \leftarrow \text{true}$
2: **return** $\pi_u^i.\mathsf{rand}[s]$

$\mathsf{RevState}(u, i, s)$:

1: $\pi_u^i.\mathsf{rev\_state}[s] \leftarrow \text{true}$
2: **return** $\pi_u^i.st[s]$

$\mathsf{Test}(u, i, s)$:

1: *// can only call* Test *once, and only on accepted stages*
2: **if** $(\mathsf{tested} \neq \perp)$ **or** $(\pi_u^i.status[s] \neq \mathtt{accept})$ **then**
3: $\quad$ **return** $\perp$
4: $\mathsf{tested} \leftarrow (u, i, s)$
5: *// return real or random key depending on b*
6: **if** $\mathsf{b} = 0$ **then**
7: $\quad$ **return** $\pi_u^i.k[s]$
8: **else**
9: $\quad k' \xleftarrow{\$} \mathcal{K}$
10: $\quad$ **return** k'

Figure 5.6: Security experiment for adversary $\mathcal{A}$ against multi-stage key indistinguishability security of protocol Π.

Session identifiers

We define the session identifiers $sid[s]$ for each stage $[s]$ of Signal in Table 5.2. It is important to note that these session identifiers only exist in our model, not in the protocol specification itself. We will use them to precisely define (via the freshness predicates of Section 5.4.3) the restrictions on the allowed behaviour of the adversary in our model. This allows us to make precise security statements. In this context, if two sessions have equal session identifiers we say that they are matching.

The precise components of the session identifiers are crucial to our definition of security: the more information is included in the session identifier, the more specific the restriction on the adversary, and hence the stronger the security model. In particular, we do *not* include identities, because they are not included in the key derivation of messages in Signal. This means that the unknown key-share (UKS) attack against TextSecure [62] is not considered an attack in our model: Alice's session with Eve will have the same session identifier as Bob's session with Alice.

5.4.3 Freshness predicates

From a key exchange perspective, the novelty of Signal is the different security guarantees of session keys depending on which stage they come from. The subtle differences between these security guarantees will be captured in our security model via the different freshness conditions we place on the Test session key. Having defined the queries of the adversary in Figure 5.6, we now define the freshness predicate fresh.

Our goal when defining fresh is to describe the best security guarantees that might be provable for each of the message keys of Signal, depending on which stage it comes from. Here, "best" is with respect to the maximal combinations of secrets learned by the adversary. That is, we use the structure of the protocol to infer what attacks cannot possibly be prevented, and rule them out by restricting the adversary accordingly.

We must define fresh separately for the initial stages and for subsequent ones, since additional secrets are introduced in later asymmetric stages. In the initial stages, our choices are based on Figure 5.5. In the 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. The intuition is that if the adversary cannot trivially query to learn the secret corresponding to one of these edges, it should not be able to distinguish the session key from random. 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 $\hat{A}$'s $ik_{\hat{A}}$ and $ek_{\hat{A}}$, it can derive the secrets corresponding to all edges. A similar observation can be made for the responder.

The vertex cover of a graph is a set of nodes incident to every edge. A vertex cover of Figure 5.5 gives a way for the adversary to compute the relevant session key directly. We can think of our freshness predicate as excluding all such vertex covers. If this graph was the complete bipartite graph $K_{3,3}$ (i.e. all nine possible DH secrets in the KDF) or even just additionally included the long-term shared secret g^{ab} of identity keys in the KDF, then the this would yield the standard eCK freshness predicate. In particular, an observation we can make here is that, because there is no long-term shared secret, Signal can make no security guarantee in our model if all ephemeral random numbers are compromised.

In stages after the initial ones, we define modified freshness conditions to capture the post-compromise security guarantees of Signal. These conditions are recursive: either the stage was fresh already, or it becomes fresh through the introduction of new secrets.

The freshness predicate fresh is defined via a variety of 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 specialised to Signal to capture the exact type of security guarantee for a message key depending on which stage it comes from.

Definition 5.4 (Validity and freshness)**.** *Let s be the i^{th} session of agent u and let $\tau \coloneqq \pi_u^i.type[s]$ be its type (e.g.* `triple`*,* `triple+DHE`*, etc.). We say that the session is* valid *if it has accepted and the adversary has not revealed either the session key or the session key of any session with the same identifier. We define this precisely as follows:*

$$\begin{aligned}\mathsf{valid}(u,i,s) \coloneqq\ & (\pi_u^i.status[s] = \texttt{accepted}) \wedge \neg \pi_u^i.\mathsf{rev\_session}[s] \\ & \wedge \Big(\forall\ j : \pi_u^i.sid[s] = \pi^j_{\pi_u^i.\mathsf{peerid}}.sid[s] \implies \neg \pi^j_{\pi_u^i.\mathsf{peerid}}.\mathsf{rev\_session}[s]\Big)\end{aligned}$$

Additionally, we say that the session is fresh *if it also satisfies cleanness* $\mathsf{clean}_\tau(u,i,s)$ *(defined in this section) where the condition depends on the stage τ. Precisely, we define the session as* fresh *as follows:*

$$\mathsf{fresh}(u,i,s) \coloneqq \mathsf{valid}(u,i,s) \wedge \mathsf{clean}_\tau(u,i,s)$$

Session Setup Stage [0]

The session key derived from a `triple` (respectively. `triple+DHE`) key exchange is derived by a applying a KDF to the concatenation of three (respectively. four) DH shared secrets. Thus, intuitively we should expect security of the session key so long as at least one of the inputs to the KDF is not revealed to the adversary. Thus, the cleanness predicate in this stage is the disjunction of three (respectively. four) predicates, each encoding the secrecy of one DH pair. Note that $\mathsf{clean}_{\texttt{triple}}$ and $\mathsf{clean}_{\texttt{triple+DHE}}$ only need to be defined for the initial key exchange, i.e. stage [0].

Definition 5.5 ($\mathsf{clean}_{\texttt{triple}}$)**.** *Within the context of Definition 5.4,*

$$\mathsf{clean}_{\texttt{triple}}(u, i, [0]) \coloneqq \mathsf{clean}_{\texttt{LM}}(u, i) \vee \mathsf{clean}_{\texttt{EL}}(u, i, [0]) \vee \mathsf{clean}_{\texttt{EM}}(u, i, [0])$$

Definition 5.6 ($\mathsf{clean}_{\texttt{triple+DHE}}$)**.** *Within the context of Definition 5.4, define*

$$\mathsf{clean}_{\texttt{triple+DHE}}(u, i, [0]) \coloneqq \mathsf{clean}_{\texttt{triple}}(u, i, [0]) \vee \mathsf{clean}_{\texttt{EE}}(u, i, [0], [0])$$

We now have to define the sub-clauses. For the sub-clause $\mathsf{clean}_{\texttt{XY}}$ in the above two definitions, our convention is that the key of the initiator is of type `X` and the key of the responder is of type `Y`, where the possible types are `L`, `M`, and `E`, denoting long-term (*ik*), medium-term (*prek*) and ephemeral (*ek*) keys respectively. Before we define these precisely, we make the following definition that captures (i) that an ephemeral key was honestly sent from a peer and (ii) it was not revealed by the adversary:

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

We can now define our various clean predicates. These are non-trivial restrictions on the adversary: For example, note that we rule out the attack where the adversary corrupts Bob's medium-term key and Alice's long-term key and then attempts to impersonate as Alice to Bob by generating ephemeral keys as Alice would in an honest session. Since there is no long-term shared secret in the KDF, such an

attack would succeed. Indeed, in later work by Bhargavan and Kobeissi [82] this was named as an explicit attack against Signal. In this work, we rule out such attacks via our freshness predicates so that we prove statements about what security guarantees Signal does actually achieve.

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

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

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

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

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

Excluded attacks. Recall that a vertex cover of Figure 5.5 gives an attack that we must rule out. Any cover must include one of $prek_{\hat{B}}$ and $ek_{\hat{A}}$ to meet the edge between them, so the only (minimal) vertex covers for a `triple` handshake are the full state of $\hat{B}$ ($prek_{\hat{B}}$, $ik_{\hat{B}}$), the full state of $\hat{A}$ ($ek_{\hat{A}}$, $ik_{\hat{A}}$), or the pair ($prek_{\hat{B}}$, $ek_{\hat{A}}$). The former two are trivial: an agent must be able to compute its own session key, so learning all of their secrets also allows the adversary to compute the session key too. The final pair exists because of the lack of an edge in Signal between $ik_{\hat{A}}$ and $ik_{\hat{B}}$. This means that an adversary who learns $prek_{\hat{B}}$ and $ek_{\hat{A}}$ can also learn the session key. In particular, since the ephemeral key is not authenticated, the adversary can generate their own $ek_{\hat{A}}$ and successfully impersonate $\hat{A}$. This is the key compromise impersonation attack of [82]. The attack is ruled out in our model because $\mathsf{clean}_{\mathtt{triple}}$ is false if $prek_{\hat{B}}$ and $ik_{\hat{A}}$ have been revealed and the ephemeral key of $\hat{A}$ is not honest. Similarly, 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 against

a `triple+DHE` handshake. Again, this could be fixed by including the long-term shared secret g^{ab} in the key derivation.

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

In Signal, keys are updated via either symmetric or asymmetric ratcheting. Asymmetric ratcheting introduces new DH shared secrets into the state, whereas symmetric ratcheting simply applies a KDF to existing state.

It will be helpful to have the following predicate:

Definition 5.7 ($\mathsf{clean}_{\mathsf{state}}$)**.** *In the context of Definition 5.4, define* $\mathsf{clean}_{\mathsf{state}}(u, i, s, s')$ *as*

$$\neg\pi_u^i.\mathsf{rev\_state}[s] \wedge \left(\forall\ j : \pi_u^i.sid[s] = \pi^j_{\pi_u^i.\mathsf{peerid}}.sid[s'] \implies \neg\pi^j_{\pi_u^i.\mathsf{peerid}}.\mathsf{rev\_state}[s']\right)$$

For brevity, we will write $\mathsf{clean}_{\mathsf{state}}(u, i, s)$ *as shorthand for* $\mathsf{clean}_{\mathsf{state}}(u, i, s, s)$.

A state reveal query on stage s reveals additional state information that a previous stage gives as input to stage s. For Signal, we define what it does as follows: For asymmetric stages, a state reveal query gives the root key used in the session key computation that was derived in the previous stage. For symmetric stages, a state reveal query gives the chain key derived in the previous stage.

During asymmetric ratcheting, there are actually two sub-stages, in which keys with slightly different properties are derived. In the first sub-stage, 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 the previous ratcheting keys of both parties. Keys from this sub-stage are marked with sid[**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 sub-stage, the parties apply a KDF to three pieces of keying material: the root key, a DH shared secret from the first sub-stage, and a DH shared secret derived from one the previous ratcheting public keys of one party and the new ratcheting public key of the other. Keys from this sub-stage are marked with sid[**asym-ir**:x] and should be secure if at least one of the three pieces is unrevealed, which is what `asym-ir` captures.

Definition 5.8 ($\mathsf{clean}_{\texttt{asym-ir}}, \mathsf{clean}_{\texttt{asym-ri}}$)**.** *Within the same context as Definition 5.4, let* $s_{ir} \coloneqq [\textbf{asym-ir}{:}x]$, $s_{ri} \coloneqq [\textbf{asym-ri}{:}x]$, $s'_{ir} \coloneqq [\textbf{asym-ir}{:}x{-}1]$ *and* $s'_{ri} \coloneqq [\textbf{asym-ri}{:}x{-}1]$. *Define* $\mathsf{clean}_{\texttt{asym-ri}}(u, i, s_{ri})$ *as*

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

Similarly, we define $\mathsf{clean}_{\texttt{asym-ir}}(u, i, s_{ir})$ *as*

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

These clauses capture the post-compromise security goal of Signal: if a device had been compromised at some prior time, 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), so the session is fresh. Similarly, the session can be fresh even if the current ephemeral exchange is compromised, just so long as the previous prior secrets are uncompromised.

Note that $\mathsf{clean}_{\texttt{EE}}$ is used twice: once to show that the randomness is uncompromised when generating ephemerals for the initial key exchange, and once to show that it is uncompromised when generating the first ratchet key pair.

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

For stages with only symmetric ratcheting, session keys should be secure only if the state used to derive the session key is unknown to the adversary. We capture this security property with Definition 5.9 below. 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 5.9 ($\mathsf{clean}_{\mathtt{sym}}$)**.** *Within the context as Definition 5.4, let* $s \coloneqq [\mathbf{sym\text{-}ir}{:}x{,}y]$*. Then we define* $\mathsf{clean}_{\mathtt{sym\text{-}ir}}(u,i,s)$ *as*

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

Note there is no stage of type `sym-ri` *with* $x=0$*. If we now let* $s \coloneqq [\mathbf{sym\text{-}ri}{:}x{,}y]$*, we similarly define* $\mathsf{clean}_{\mathtt{sym\text{-}ri}}(u,i,s)$ *as*

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

We may write $\mathsf{clean}_{\mathtt{sym}}$ *to denote* $\mathsf{clean}_{\mathtt{sym\text{-}ir}}$ *or* $\mathsf{clean}_{\mathtt{sym\text{-}ri}}$ *where it is clear from context which one we mean.*

Excluded attacks. Since no additional secrets are included in message keys derived from symmetric ratchet stages, these predicates simply capture that the adversary is not allowed to trivially compromise any previous state along the chain. This includes the asymmetric stage that created the chain, as well as any of the intermediate symmetric stages. In other words, we exclude the attack in which the adversary corrupts a chain key and computes subsequent messages keys from it.

5.5 Security analysis

We are now finally ready to prove that Signal is a secure multi-stage key exchange protocol in our model, under standard assumptions on the cryptographic building blocks. In this section, we provide the precise theorem and a proof sketch. The full proof appears in Appendix B. Before this, we summarise some key points about Signal.

We have made a few minor reorganisations in Figure 5.3 compared to the actual implementation of Signal. In our model, we partition the protocol execution of Signal into stages. We consider Signal to generate the first message keys for each chain at the same time that it initialises the chain. This means that all stages (including asymmetric stages) generate a session key. We also consider Bob to send his own one-time prekey $eprepk_{\hat{B}}$ instead of relaying it via the server. Another difference is that we do not have

AEAD messages in our model because this would trivially break key indistinguishability: the adversary could test a session and then attempt use the challenge key k_b to MAC a message. If the MAC is the same as an observed one sent from the actor of the Test session, then the adversary can be sure that k_b is the genuine session key. So in our model, we are only analysing the underlying key exchange aspects of Signal.

Theorem 5.10. *The Signal protocol is a secure multi-stage key exchange protocol under the Gap Diffie-Hellman assumption and assuming every* KDF *is a random oracle.*

Proof (sketch). Here we provide a proof sketch. The full proof appears in Appendix B. The proof is essentially a case distinction on every possible situation of the Test session being fresh. The proof considers if the Test session is from the initial handshake (`triple` or `triple+DHE`), an asymmetric stage or a symmetric stage. The proof considers each stage type and sub-clause of freshness predicate exhaustively and proceeds in a game hopping argument. Many cases are similar, which is why the techniques from Chapter 3 can be leveraged.

Initial stage. We start by proving the security of a Test session key that is output by the initial handshake. For a type `triple` key exchange, we show this via taking cases over the disjuncts in the $\mathsf{clean}_{\texttt{triple}}$ clause—over the different ways the session could be clean—noting that one of $\mathsf{clean}_{\texttt{LM}}(u, i)$, $\mathsf{clean}_{\texttt{EL}}(u, i, [0])$ or $\mathsf{clean}_{\texttt{EM}}(u, i, [0])$ must be upheld. We then consider each of these events in turn and bound the probability of the adversary winning in each by a simple reduction argument. The case of `triple+DHE` is the same except $\mathsf{clean}_{\texttt{EE}}(u, i, [0], [0])$ is an additional cleanness clause that must be considered.

Asymmetric stages. Next, we consider the security of a Test session key that is output from a stage that has type `asym-ir` or `asym-ri`. Again, we consider cases over the different ways to satisfy the cleanness predicate depending on the type of the stage. Most cases are of the form $\mathsf{clean}_{\texttt{EE}}$, and for these we obtain a probability bound by replacing the DH ratchet keys and shared secrets with values from a GDH challenger.

The only case not of this form involves $\mathsf{clean}_{\mathsf{state}}$, which describes a scenario where both recent ratchet keys were compromised, but the previous stage was still secure. Secrecy here is intuitive, and the bound follows from an inductive argument: if an adversary could win in this manner, then there is an adversary that could win against the previous stage.

Symmetric stages. Finally, we consider the security of a Test session key that is output from a stage of type `sym`. Here there is no disjunction in the cleanness predicate. Hence, there is only one case to consider. We replace the keys used to initialise the current sending chain with uniformly random values, since an adversary that could detect this could win against that previous stage.

Conclusion. The theorem follows by summing probabilities across all cases and noting that the advantage of any efficiency adversary is negligible. □

5.5.1 On hardness assumptions and the random oracle model

When performing a game hopping security proof in an extended Canetti-Krawczyk-style model, after each hop we must show that the resulting game is similar to the original. If certain values have been changed, the queries whose results differ must be simulated in an indistinguishable manner.

In particular, the eCK family of models all contain a query RevSessKey that reveals the session key derived by a targeted session. This models for example cryptanalysis of a large volume of encrypted traffic, or the ability to read certain locations in memory. When replacing certain DH keys with random values, we must ensure that the resulting game is similar to its original. For protocols using only ephemeral DH values g^x and g^y to compute session keys from g^{xy}, replacing g^x and g^y by random does not affect other sessions, and thus other RevSessKey queries are not affected. However, for more complex protocols (such as NAXOS and HMQV) in which the long-term keys are also included in the session key derivation, this game hop becomes more complex. Specifically, when the long-term keys are modified, *all* RevSessKey queries are affected, and their simulation is no longer trivial.

There is a proof obligation to show that the simulation of these queries does not allow an adversary to distinguish the two games. One way to do this is by using the Gap DH assumption in the random oracle model, assuming that the KDF is a random oracle. In the simulation, whenever the adversary makes a query to the random oracle, the challenger tests the relevant part of the argument using the DDH oracle to determine whether the adversary has successfully derived the DH secret. If so, the simulation can terminate and the challenger uses this value in the Gap DH game. This is the approach we take.

There are known issues with the ROM. An alternative to Gap DH is to take a PRF-ODH (pseudorandom function with oracle DH) assumption, which effectively provides an oracle for session key computations, and (roughly) asserts that it is hard to solve computational DH even with access to the oracle. The game hop then takes the computational DH values from the PRF-ODH game, and answers RevSessKey queries by querying the oracle. Thus, the probability jump over the game is bounded by the PRF-ODH advantage.

There is a further complication in the case of Signal. In most normal DH protocols, there is only one method to compute a session key given a collection of secret inputs. Such a method could be called a "combinator". For example, in NAXOS the combinator hashes one long-term key and uses that as a DH exponential. In Signal, on the other hand, there are many different combinators, and the oracle we use must be sufficiently flexible to simulate all of them. Thus, we have the following options:

(i) Define a PRF-ODH game parameterised by the combinator K used to assemble secrets into the arguments to the KDF. For each different type of key in Signal, assume hardness of this game and use this assumption to justify a game hop.

(ii) Assume that the KDF is a random oracle, and justify the game hop directly from the random oracle model and Gap DH.

We will prove the security of Signal (Theorem 5.10) in detail using the latter technique in Appendix B.1 and then we will give a more high-level proof using the former technique in Appendix B.2.

6
Related Work

Contents

In this chapter, we review related work to this thesis. In Section 6.1 we give an overview of alternative methods for analysing AKE protocols. In Section 6.2 we present additional work related to the concept of post-compromise security, which is relevant to the material of both Chapter 4 and Chapter 5.

6.1 Other provable security techniques

The primary proof technique in this thesis is computational game-based arguments. This proof technique is currently the most powerful in analysing complex protocols with state in intricate adversarial models. However, there are alternative security proof techniques that we did not deem appropriate for the work in this thesis.

6.1.1 Software tools

The provable security approach via game-based arguments is in principle able to deliver rigorous and detailed mathematical proofs. However, there is a trade-off between how comprehensive a security model can be and how easy it is to prove that a protocol is secure within it. New cryptographic designs (and the corresponding security analyses) are increasingly complex. Consequently, protocols that are proved secure may still have flaws if the model does not match reality. This was the case for TLS 1.2, for which the security proof of [91] predated attacks such as [3–5, 20–22]. The existence of these attacks does not contradict the security proof; the attacks simply fall outside the scope of the security model. For increasingly complex security models, there is a growing emphasis on shifting from algorithmic descriptions to implementation-level descriptions that account for implementation details, recommendations from standards when they exist, and possibly even side-channels. As we have seen in this thesis, the problem with these overly strong and comprehensive security models that capture many attacks is that as a consequence, cryptographic proofs are becoming increasingly error-prone and difficult to check. Sometimes, it can even be hard to formalise a proof at all.

One promising solution to address these concerns is to develop machine-checked frameworks that support the construction and automated verification of AKE protocols. These tools work in either the symbolic framework (such as with ProVerif [25] or Tamarin [102]), or the computational framework (such as with EasyCrypt [8] or CryptoVerif [24]). The computational framework is what we have been using throughout this thesis, whereby cryptographic primitives are functions on bitstrings and the adversary is a polynomial time Turing machine. The symbolic model on the other hand separates the capabilities of the adversary into a set of rules [9]. These tools have been used to rediscover and find new attacks automatically [9, 52]. While these tools are useful to brute force find attacks or proofs of some AKE protocols, they tend to struggle when a protocol contains a lot of state, as they can get stuck in backward inductive reasoning. Thus, at present, these tools are not adequate to analyse protocols with PCS such as Signal.

6.1.2 Universal composability

Another technique to achieve provable security is universal composability (UC) [39, 77]. In the UC framework, a general-purpose model for the analysis of cryptographic protocols is defined where security is proven via protocol emulation in an analogous way to zero-knowledge proofs. To prove that a protocol π_1 meets certain security definitions in the UC framework, first an ideal functionality π_2 is defined as a protocol that behaves in an specified “ideal" way. One way to think of the ideal functionality is as a trusted third party with access to a perfectly secure channel to all parties, that takes all the necessary input from all parties and computes the result and outputs it to those parties who should receive it. Next, an external entity known as the environment interacts with either the simulator (that acts as an ideal adversary) and a trusted party computing the ideal functionality π_2, or a real adversary running π_1 with honest parties. A security proof of π_1 in the UC model amounts to showing that the environment is unable to distinguish between these two situations; if the output distribution of the honest parties and the simulator in an ideal execution is computationally indistinguishable from the output of the honest parties and adversary in a real execution, then this means that anything that a real adversary can do, can also be achieved in the ideal model. However, trivially, the simulator can do no harm in the ideal model. Finally, a universal composability theorem [39] proves that if the protocol meets this definition of security, then it remains secure even if arbitrarily composed with other instances of the same or other protocols (subject to some conditions).

The UC framework and composability theorems in general have the useful advantage that they tend to produce powerful statements on strong protocols. However, they also suffer from weaknesses such as the model being too weak with regards to adversarial dynamic compromise; when it comes to proving useful real-world properties such as forward secrecy or PCS, defining the ideal functionality, and thus making proofs, can be very difficult. In addition, many existing protocols do not meet the conditions of the UC framework. As such, this methodology is difficult to apply to the work in this thesis.

6.2 Stateful protocols

A major part of this thesis has been the study of post-compromise security. As shown in Theorem 4.7, this can only possibly be achieved by stateful protocols. In this section, we provide an overview of some of the related work in this domain.

In [112], the term "future secrecy" is defined to mean "a leak of keys to a passive eavesdropper will be healed". In other words, an adversary who learns keys but does not interfere with network traffic only learns secret data for a bounded time period. This makes sense intuitively, but a closer look reveals that this is in fact a form of *forward* secrecy. The key insight is that a passive adversary can perform their computations at any point, and so no extra advantage is gained by learning long-term keys before the initial messages. Indeed, such security can be achieved by a basic, unsigned Diffie–Hellman key exchange, meaning that it is more relevant in the context of lightweight or otherwise restricted classes of protocols.

In more detail, suppose some protocol π has (weak) forward secrecy: no attacker can learn long-term keys to retroactively discover past session keys. Then π already satisfies the above form of future secrecy. Indeed, suppose for contradiction that some passive attacker $\mathcal{A}$ violates the future secrecy but not the (weak) forward secrecy of π: it learns the long-term keys, eavesdrops on a particular session s, and subsequently can distinguish the session key of s from random. Since the eavesdropping does not depend on knowing the long-term keys, it is possible to exchange the order of these two operations, defining an attacker $\mathcal{A}'$ that first eavesdrops and then reveals long-term keys. But $\mathcal{A}'$ is then a successful attacker against forward secrecy, contradicting our assumption.

The concept of "key continuity" or "self-healing" appears in various Internet Drafts [75, 136], but there is little formal work in the academic literature to mirror these discussions. We believe that this intuitive notion is challenging to capture formally, and that it is a form of post-compromise security. The difficulty arises when describing what is implied when a protocol is "healed": clearly it is meant that the protocol is secure against certain classes of adversaries, but the precise definition of what attacks fall into the class is not clear.

Various protocols appear to implement some sort of key continuity (or PCS) features, including OTR [1], ZRTP [136], and even many SSH [135] implementations via the known_hosts feature; however, existing analyses of these protocols do not cover this aspect.

In [31], the authors define a notion of "drifting keys", in which devices randomly modify their long-term key over time, such that a trusted verifier can detect the discrepancies caused by an attacker with a slightly incorrect key. This type of protocol detects the adversary but does not prevent the compromise. Nevertheless, we view their work as providing a type of PCS.

In [70] a "puncturable encryption" scheme is defined that achieves forward secrecy with zero round trip time. The idea is that as Alice receives each message, with a specific tag, encrypted under her public key, she can "puncture" her secret key, modifying it in a certain way so that her secret key irreservibly changes while her public key stays the same. The modified private key can be used to decrypt other messages with different tags but not another message with the same tag. Hence, this provides forward secrecy. We believe that this would be an elegant way to achieve asynchronous communication in Signal instead of relying on a server to constantly be online to distribute keys (e.g. for Signal email). However, despite some generalisations on the scheme [74], puncturable encryption currently requires key and ciphertext sizes that are somewhat cumbersome.

In [51], stateful protocols are defined that pool randomness across sessions. Pooling of randomness provides strong security guarantees against weak random number generation: roughly, an actor only needs one of its sessions to have secure random numbers for subsequent random number generation to be secure. However, the protocols in [51] do not consider synchronisation of state between agents so they do not achieve PCS. We view our work as orthogonal to theirs. Interestingly, an added advantage of our PCS transform is that it can also provide a measure of protection against weak random number generation, just like with the pooling of randomness construction. However, because the sharing of state is per-peer, so is the weak random number generation protection. We elaborate on this concept in Section 4.4.

There has not been much analysis of stateful AKE, but state is used in various other primitives: stateful symmetric encryption is well-known, and asymmetric encryption is also discussed in the literature [14]. State is used *within* sessions. For example, ACCE [87] provides security to the message channel in the sense of stateful length-hiding authenticated encryption. It is also used heavily at the application layer, particularly in the web context.

7 Conclusions

Contents

7.1 Summary

In this thesis, we developed a framework to prove novel security properties of protocols that were previously beyond the scope of current techniques. We then applied this knowledge to give the first ever formal analysis of Signal, a protocol used by billions of people worldwide.

We followed the three main themes of realistic, strong and provable key exchange security. We first ventured into the theme of provable security, where we developed techniques to reduce the length and complexity of security proofs. This involved providing generic theorems, applicable to a wide range of security models, to minimise the repetitive steps of a game hopping proof. We also provided generic conditions under which a useful game hop can occur. To demonstrate the usefulness of these techniques (as well as to make the proofs themselves clearer), we applied these techniques in later parts of this thesis.

Next, inspired by real-world protocols such as Off-the-Record Messaging and Signal, we investigated the strong security properties that can be achieved by protocol participants sharing and evolving state across sessions. These protocols intuitively provided a self-healing property, whereby participants could in some sense recover from compromise to re-establish security guarantees in later sessions. It was previously thought impossible to make any formal security guarantees in this case. We developed security models to capture this form of compromise. Specifically, we call such guarantees post-compromise security, and it comes in two flavours: weak and total. Weak PCS corresponds to temporary limited compromise of a long-term key, such as via an HSM. We provided a protocol and proof that security in this case can be achieved so long as the protocol fulfils an aliveness property. We supplemented this work with a note that QUIC and OPTLS do not achieve weak PCS because they allow the server to sign authentication tokens and use them for authentication in the distant future. On the other hand, total PCS refers to full compromise of state. Protocols can recover from full compromise if they consistently send and update keying material with their communication partners across sessions. We provided generic protocol transformations and proofs that, given a protocol secure in a model (such as eCK), we can transform the protocol to additionally achieve total PCS. We supplemented this work by noting that protocols such as OTR, which regularly add new keying material, may achieve total PCS, but the predecessor to Signal, TextSecure, does not because of its multiple device functionality: an adversary could compromise Bob and then register a new device as Bob's using Bob's long-term key only, and then establish a new session with Alice. The ever-evolving state across sessions that provides PCS is not used when a new device attempts to communicate; hence, the adversary can impersonate as Bob to Alice using Bob's long-term key only. Our generic reasoning led to a whole hierarchy of strong AKE models and accompanying protocols for them. Our work advances the state of the art while our case studies show that actually achieving PCS in practice can be non-trivial in subtle ways.

We then applied the previous work in this thesis to the formal security analysis of Signal. Despite the fact that Signal was already being used by billions of people on

a daily basis via its implementation in WhatsApp, Facebook Messenger and Google Allo amongst others, this constituted the first formal analysis of the protocol. Signal works in multiple stages via its so-called Double Ratchet mechanism. We therefore developed a novel tree-based multi-stage key exchange security model to analyse the key exchange security properties of Signal. We proved that Signal does achieve a form of PCS via its asymmetric ratchet, whereby Alice and Bob take turns to send each other new Diffie–Hellman (ratchet) keys. We went on to show, using precise freshness predicates for each stage, that different message keys have different security properties depending on which stage they are derived in (we identified six different security properties for message keys: `triple`, `triple+DHE`, `asym-ir`, `asym-ri`, `sym-ir` and `sym-ri`). We explained in detail how Signal works, how it could be improved (for instance, by including the long-term shared secret in the KDF to protect against bad random number generation) and what security guarantees it provably achieves. As with many real-world security protocols, there are no detailed security goals specified for the protocol, so it is ultimately impossible to say if Signal achieves its goals. However, our analysis proves that several standard security properties are satisfied by the protocol, and we have found no major flaws in its design.

7.2 Future work

In this section, we outline some potential future areas of research related to the work in this thesis.

Extensions to provable security results

Public key distribution. Throughout this thesis, we have assumed an honest public key distribution of long-term keys. For example, in Signal the server is not meant to be trusted. It is only there to deliver messages and public keys between participants. Of course, the server could deliver malicious public keys instead. To mitigate this, Signal supports a mechanism for out-of-band verification of public long-term keys via fingerprints, by either Alice and Bob comparing human-readable so-called safety numbers or scanning a QR code that encodes the public long-term keys. In our model,

we simply assumed that long-term and medium-term public key distribution is honest. Our work could be extended by more accurately modelling what really happens. On a similar note, the other models in this thesis could be extended by incorporating entities such as certificate authorities. Some work has been done on this. For example, in [32] the AKE Security Incorporating Certification Systems (ASICS) framework is developed, that allows for the explicit modelling of the certification process of public keys. The ASICS framework captures an adversary that can dynamically register arbitrary bitstrings as public keys with a certificate authority (CA) that does not do any checks. In [126], more attack scenarios related to the operations of CAs are identified, such as a corrupted CA issuing rogue certificates. Our models could be extended in non-trivial ways by incorporating these sorts of adversarial scenarios.

Tightness of the security reduction. As pointed out in [7], a limitation of conventional game hopping proofs for AKE protocols is that they do not provide tight security reductions. The underlying reason is that the reductions depend on guessing the specific party and session under attack. In the case of a widely deployed protocol with huge amounts of sessions, such as Signal, this leads to an extremely non-tight reduction. While [6] develops some new AKE protocols with tight reductions, their protocols are non-standard in their setup and assumptions. In particular, there is currently no known technique for constructing a tight reduction that is applicable to Signal or protocols like it.

Implementation-specific threats. In our security models, we make various assumptions on the components used by the protocols. In particular, we do not consider specific implementations of primitives (e.g. the particular choice of curve), instead assuming standard security properties in a black-box fashion. We also do not consider side-channel attacks.

Extensions to post-compromise security

Weak PCS. In our analysis of weak PCS, we only considered the scenario of temporary compromise of an HSM that has signing capabilities. Of course, in practice

most HSMs perform a wide range of operations. An extension of our work could consider the temporary compromise of a more realistic HSM. To do this, one could examine the PKCS#11 standard to determine what additional features of an HSM to model.

Identity-based encryption protocols. Identity-based encryption (IBE) protocols are highly efficient and many have been proven to give strong security guarantees [28]. However, they all have a single point of failure in that they fundamentally rely on trusting a third party private key generator (PKG) that literally gives every party their secret key. Therefore, at any time the PKG may decrypt the messages from any conversation or impersonate any party. While key escrow is sometimes described as a feature of all IBE protocols,[1] it is sometimes not a desirable property of the protocol at all. In an attempt to remove this trust, secure multi-party computation (MPC) protocols have been proposed. These protocols work by essentially partitioning a global shared PKG into individual participants and forcing each to use MPC to compute a secret that they themselves do not know unless a certain threshold cheat and collaborate. While these MPC protocols do work, in many identity-based protocols, such as the Sakai-Kasahara scheme [72, 119] as used in the highly efficient MIKEY-SAKKE telephony protocol [71], these methods do not scale well and can be very complex [66, 120]. Moreover, if a certain threshold of the distrusted PKG is malicious, then there are no security guarantees. They also do not offer additional security guarantees beyond removing (some) trust in the global third party. In particular, because the identity-based protocol is still stateless, all security of a user still depends on protecting their local private key, even if the master secret of the PKG is removed. We conjecture that by using the PCS technique of sharing state, it is possible to reduce the trust assumptions in the PKG. The intuition is that, by sharing and evolving state across sessions, it should only take one secure communication to forever lock out the PKG in the future.

[1] Sometimes it might be a desirable property for a trusted third party to be able to monitor participants, e.g. for investigative or regulatory purposes in enterprise situations.

Extensions to the analysis of Signal

Signal is a very complex protocol. As such, for a first formal analysis we have chosen (some) simplicity over a full analysis of all of the features of Signal. We hope that our presentation and model can serve as a starting point for future analyses of Signal and protocols like it. In addition to some of the remarks above, which apply to Signal, we could also extend our analysis of Signal in the following ways:

Header encryption variant. The open source libraries from Open Whisper Systems contain various sections of code that are not considered part of the "core" of Signal. A major section is the "header encryption" variant of the Double Ratchet mechanism, which is intended to hide metadata. It has been implemented in Adam Langley's Pond protocol [96].[2] This mechanism is included in the reference implementation but not used in the Signal implementation by Open Whisper Systems, WhatsApp, Facebook or Google. We did not consider this variant. It would be interesting to see a full analysis and what formal security guarantees can be achieved.

Same key for Ed25519 signing and Curve25519. Signal uses the same key ik for DH agreement and for signing the medium-term prekeys[3]. In [54, 110] security of a similar scheme is proven under the Gap-DH assumption, effectively showing that the signatures can be simulated using the hashing random oracle. We conjecture a similar argument could apply here, but we do not prove it. Instead, we omit the signatures from consideration and enforce authentication of the prekeys in the game via our freshness predicates. A more comprehensive security proof would consider this awkward key reuse situation.

Out-of-order decryption. To decrypt out-of-order messages, users must store message keys until the messages arrive, reducing their forward security. As discussed in Section 5.5, we do not consider this storage.

[2]The Pond project is currently in hiatus.

[3]This is done in practice by reinterpreting the Curve25519 point as an Ed25519 key and computing an EdDSA signature.

Simultaneous session initiation. Signal has a mechanism to deal silently with the case that Alice and Bob 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.

Signal implementation variants. There are a multitude of different variants of Signal that all implement the protocol slightly differently. In this thesis, we decided to only analyse the core of the protocol. However, popular applications using Signal sometimes change important details as they implement the protocol. Given the popularity of some of these variants, they merit separate security analyses.

For example, WhatsApp implements an automatic retransmission mechanism in the event a message is in transit and the intended peer changes their public key: if Bob appears to change his identity key while Alice's messages have not been received, Alice will automatically resend in-transit messages encrypted using the new public key of Bob. Hence, an adversary with control over identity registration can disconnect Bob and replace his key, and Alice will resend the message to the adversary. Despite being known for some time, this unexpectedly became international news many months later as a "backdoor" in WhatsApp [27, 64].[4] Whether it is a deliberate backdoor or a usability trade-off (users will want messages to be delivered in the event that their communication partner purchases a new phone) is unknown. On the other hand, the Signal implementation from Open Whisper Systems gives users the option to block resending messages encrypted using the new public key. Interestingly, WhatsApp offers an additional feature to show security notifications for security concious users. Even with this setting enabled, WhatsApp will still resend messages and only tell Alice that Bob's public key has changed after messages have been automatically resent. In [101], Moxie Marlinspike of Open Whisper Systems argues that this is a user

[4]The original article from the Guardian [64] has been extensively amended and the headline changed. The Guardian did eventually admit flawed reporting [42].

experience trade-off and actually blocking messages for security conscious users (as Signal from Open Whisper Systems itself does for all users) would reveal to WhatsApp which users have these settings enabled. This claim is wrong: users could simply send dummy messages in the place of the real messages if they wanted to block and fool the server into thinking they had not (the fake messages should be received silently in the user interface to preserve a user experience). A standard encryption scheme should make this indistinguishable to the server. Moreover, it is not clear from the implementation of WhatsApp whether the retransmission vulnerability only applies for one sequence of consecutive messages from Alice to Bob without reply, or whole conversations. The WhatsApp server could block all "message has been delivered" notifications (represented by two ticks in the user interface) as these are presumably always sent after a message is delivered, and so easy for the server to recognise. Users would then only see notifications for "message sent" (one tick), but their actual conversation will proceed as normal. The adversary could then launch the above attack on both parties to reveal the entire conversation transcript.

Other interesting variants of Signal to investigate include implementations of large group chats or a possible multiple device functionality. For instance, in the most popular implementations such as the ones from WhatsApp and Open Whisper Systems, large group chats are initially implemented as pairwise Signal sessions, but after a certain threshold of group participants are reached, the group switches to using a single "sender key" for efficiency reasons [129]. Clearly switching to only using sender keys, and thus abandoning asymmetric ratcheting, means there are no total PCS guarantees.

One proposal, called Asymmetric Ratcheting Tree (ART) [45] proposes a scheme that uses a tree of DH values that can ratchet forward to provide PCS while at the same time being more efficient than pairwise communication. Even without the degradation to sender keys, more investigation needs to be done on security properties such as transcript consistency across the group chat. Finally, as we saw in Chapter 4, maintaining PCS while implementing a multiple device functionality is non-trivial. TextSecure did not achieve it. Signal and WhatsApp only currently have a multiple

device functionality by routing all communication through the phone. More analysis of how best to implement the multiple device functionality needs to be done.

Facebook Messenger does not currently implement group chats in Signal, but they have implemented so-called message franking. Message franking makes it possible for users to verifiability report abuse even in their end-to-end encrypted conversations [104]. The formal analysis of these kinds of schemes has only just begun [73].

Other end-to-end encrypted messengers. Finally, we focused on Signal because it is the most widely used and (in our view) the most cryptographically interesting end-to-end encrypted messaging protocol. There are, however, an enormous variety of different messaging apps with little or no formal academic analysis [47, 59, 65, 99, 123, 125, 128, 130, 133]. Some claim to use their own designs based on Signal [128] [133], while others have radically different designs [123]. Given the popularity of some of these messaging apps,[5] more security analyses are needed.

7.3 Final remarks

We conclude this thesis by placing the impact of this work in context and outlining our vision for the future of the field.

Historically, there has been a worrying trend of real-world protocols being implemented before academic analysis has been able to catch up. Similarly, strong theoretical protocols have not been used widely. As a result, real-world security has not been as good as it could be. Indeed, in the last few years we have seen an enormous proliferation of competing encrypted messaging apps with little or no accompanying security analyses. Our analysis of Signal was not easy due to lack of documentation and the complex design.[6] We would like to abate this worrying trend; cryptography does not have to be this way.

We believe there are more generic useful contributions in this thesis besides the analysis of some protocols. For instance, we explain how to introduce state to provide

[5] Confide [47] has ostensibly been used by White House staff at the Trump Administration [109] while Telegram [123] claims over 100 million active users [124].

[6] Although Trevor Perrin of Open Whisper Systems did provide helpful discussions.

stronger security guarantees, common pitfalls of real-world protocols and generic proof techniques. Cryptographers can use these principles to design secure cryptosystems from the ground up. By achieving a good balance between simplicity and security, such protocols are far more likely to be analysed and implemented widely in the future. We believe that this thesis provides useful tools for the design and analysis of protocols now and in the future. We hope that others in industry and academia will build upon our work in the future for the design of realistic, strong and provable cryptography.

Appendices

A

Post-Compromise Security Proofs

Contents

A.1 Proof for the weak compromise model

Theorem 4.4. *Let* $\mathcal{I}(A, \mathsf{sk}_{\hat{A}}, m) \coloneqq \mathsf{Sig}_{\mathsf{sk}_{\hat{A}}}(m)$ *be the signature algorithm of an EUF-CMA signature scheme* $(\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$*. Let* π *be the protocol in Figure 4.4 and recall* $X_{basic} = (Q, F)$ *from Definition 4.1. The protocol* π *is secure in the model* $(Q \cup \{\mathsf{HSM}\}, F \cap \mathsf{PCS\text{-}HSM})$ *under the random oracle and DDH assumptions.*

Proof. Assume the adversary always selects a Test session s where $s_{\mathsf{actor}} = \hat{A}$ is the initiator. The other case is similar. For the adversary to win, the Test session must have accepted, so the received signature must be correct. Let the transcript of messages sent and received be as in Figure 4.4. We call the actor of the Test session Alice ($\hat{A}$) and the intended peer Bob ($\hat{B}$).

Game 0. This is the original security game.

Game 1. This is the original security game except it aborts if two nonces ever collide or if two ephemeral DH keys ever collide. Similarly, abort if the random oracle KDF ever produces a collision. This is a standard game hop from Section 3.4, so we see that $|\Pr(S_0) - \Pr(S_1)|$ is negligible.

Game 2. This game is identical to Game 1 except it aborts if the challenger fails to correctly guess the Test session s in advance. Similarly, we can also guess the peer to the Test session in advance. This is another standard game hop from Section 3.4.

Game 3. This game is identical to Game 2 except it aborts if the adversary ever queries $\mathsf{HSM}(\hat{B}, \hat{A}, g^y, n_{\hat{A}})$ (for the unique g^y received by $\hat{A}$ in the second message of s and the unique nonce $n_{\hat{A}}$ selected by $\hat{A}$ in the first message of s). As access to the HSM query is revoked before $n_{\hat{A}}$ is even generated by $\hat{A}$, the probability that the adversary will query HSM on the random string $n_{\hat{A}}$ is negligible. Hence, $|\Pr(S_2) - \Pr(S_3)|$ is negligible.

Game 4. This game is identical to Game 3 except it aborts if in the Test session s, the message received $(\hat{B}, \hat{A}, g^y, n_{\hat{B}}, \mathsf{Sig}_{\mathsf{sk}_{\hat{B}}}(\hat{B}, \hat{A}, g^y, n_{\hat{A}}))$ is such that the intended peer $\hat{B}$ never honestly produced a signature on $(\hat{B}, \hat{A}, g^y, n_{\hat{A}})$ in the run of any its sessions.

By a simple reduction argument, we can see that $|\Pr(S_3) - \Pr(S_4)|$ is negligible under the EUF-CMA assumption. We can construct a forger against an EUF-CMA signature scheme as follows: the forger simply runs an instance of this AKE game. The forger already knows which session will be the Test session and which entity will be the peer to the Test session in advance due to Game 2. The forger inserts the public key of Bob as the challenge EUF-CMA public key form the EUF-CMA game. If the forger ever needs to simulate a signature with Bob's key, it simply queries the EUF-CMA challenger. If it is the case that the Test session ever receives a signature over $(\hat{B}, \hat{A}, g^y, n_{\hat{A}})$ despite the fact that Bob never honestly produces this signature,

then this implies a forgery in the EUF-CMA game. Hence, $|\Pr(S_3) - \Pr(S_4)|$ is negligible under the EUF-CMA assumption.

Game 5. This game is the same as Game 4 except that in Test session s that sends g^x and receives g^y, it swaps all occurrences of g^{xy} with g^z for z chosen uniformly at random.

Note that because of Game 4, y is honestly generated and unknown to the adversary. Similarly, the x is chosen in the Test session itself so is also unknown to the adversary. Thus, intuitively the adversary should struggle to differentiate between these two games because it should struggle to differentiate between g^{xy} and g^z.

Formally, we argue as follows: consider a DDH adversary that receives challenge values (g^x, g^y, g^z) and must determine if $z = xy$ or $z \neq xy$. The adversary inserts these values in this game (which it can do because it already knows which session will be the Test session and which entity will be the peer to the Test session). Clearly, if $z = xy$, then this is Game 4, while if z is chosen uniformly at random, then this is Game 5. If the difference in success probability of the adversary between these games were non-negligible, then this would yield a way to correctly answer the DDH challenge with non-negligible advantage. Hence $|\Pr(S_4) - \Pr(S_5)|$ is negligible under the DDH assumption.

Conclusion. At this point it is hopefully clear that the adversary has little hope to distinguish the Test session key from random: the KDF is a random oracle and the KDF input has been replaced with a secret chosen uniformly at random. However, there is one last argument we need to make. Note that because the KDF includes the transcript T (and the random oracle produces no collisions due to Game 1), we can say that sessions s and s' match if and only if they compute the same session key. As such, the no-match attacks of [98] do not apply. □

A.2 Network Robustness

One-round protocols with post-compromise secrecy are necessarily not robust to an unreliable network. The intuitive reason for this fact is that if the responder Bob completes an honest session, he cannot be sure that his message to Alice was received. This will cause problems with future sessions.

On one hand, for protocol π to have PCS, Bob (and Alice) must inexorably ratchet state forward: Bob cannot revert back to his previous state, because then his completed (refreshing) session counts for nothing. On the other hand, to have correctness, Bob must revert back to his previous state to be able to correctly talk to Alice in the event that his message did not go through. Because the protocol only has one message each, there is no way for Alice to confirm to Bob that his message was successfully delivered or not. Intuitively, protocols of more than two messages do not have this problem because, although there is never a guarantee that the last message sent is delivered, such protocols have enough messages for one side to provide confirmation of what they think the synchronised state is. We now formalise this argument.

Theorem 4.11. *No correct one-round protocol π that is secure in* KE-PCS *is post-network robust.*

Proof. We give an explicit adversary $\mathcal{A}$ in KE-PCS against π. The Test actor will be Bob. The adversary corrupts some agent Alice and then completes an honest (refreshing) session between Alice and Bob. The adversary then uses the *previous* state value, which it learnt from the corruption, to try to impersonate as Alice to Bob in a subsequent session. We must argue (i) that this adversary does not contradict freshness, which holds because of the intermediate refreshing session, and also (ii) that it wins the security game, which we show with an explicit reduction to robustness.

We define $\mathcal{A}$ as follows: First, it picks two targets $\hat{A}$ and $\hat{B}$ and issues a $\mathsf{corrupt}(\hat{A})$ query. Since π is post-network robust, the correct execution adversary $\mathcal{C} = \mathcal{C}(\hat{A}, \hat{B})$ induces matching sessions at $\hat{A}$ and $\hat{B}$, with $\hat{A}$ the initiator and $\hat{B}$ the responder. Our adversary $\mathcal{A}$ next executes $\mathcal{C}$. That is, it simulates its execution; whenever $\mathcal{C}$ issues a query, $\mathcal{A}$ issues the same query and returns the response to $\mathcal{C}$. Denote the message

from $\hat{A}$ to $\hat{B}$ by m_1 and the message $\hat{B}$ to $\hat{A}$ by m_2. This will create a refreshing session for $\hat{B}$.

Next, $\mathcal{A}$ uses the response to the $\mathsf{corrupt}(\hat{A})$ query to set up a simulated $\hat{A}$-oracle Σ, initialising it with the setup algorithm used by the game. At the moment, this $\hat{A}$-oracle has the same state as $\hat{A}$ at the start of the game. But note that $\hat{A}$ just had an honest session with $\hat{B}$, so the state of the real $\hat{A}$ is most likely different.

Now $\mathcal{A}$ tries to use Σ to impersonate as $\hat{A}$ to $\hat{B}$. The idea is that the Σ oracle simulates Alice's state in the world where Bob's reply message did not get through. Therefore, Σ simulates this situation by sending a message in a first session without receiving a reply. This simulates the situation of Alice sending m_1 but not receiving m_2. Of course, Σ may not be able to produce the state required for the real message m_1 (e.g. it may require knowing an x for Alice's sent message m_1 containing g^x), but given that the state of Σ is the same as Alice's state at the start of the game, the simulated message will be equidistributed the same as m_1. The adversary next executes $\mathcal{C}$ for a second time, except redirecting the $\mathsf{create}(\hat{A}, \mathcal{I}, \hat{B})$ and $\mathsf{send}(\hat{A}, 2, m_3)$ queries made by $\mathcal{C}$ to Σ (where m_3 denotes the message sent by $\hat{B}$, if $\hat{B}$ indeed accepts and sends a reply). That is, $\mathcal{A}$ stores the current state st_Σ in its memory, and whenever $\mathcal{C}$ issues a query addressed to Alice, $\mathcal{A}$ computes the response using Σ and returns its output as the result of the query. All other queries (i.e. the ones addressed to $\hat{B}$) are faithfully invoked by $\mathcal{A}$.

Let s denote the session created at $\hat{B}$ by this execution. Note that s must complete, and therefore has $s.\mathrm{status} \in \{\mathtt{accept}, \mathtt{reject}\}$. We consider these cases separately. The crux of this proof is that, for π to be secure in $\mathsf{KE\text{-}PCS}$, s must surely not accept. On the other hand, if s does not accept, then π is not post-network robust, because the state of Σ could have happened naturally for the real Alice with a simple dropped message in the initial honest sessions.

Case 1: s accepts

In this case the impersonate attack was successful. Intuitively, this must mean that Bob did not inexorably ratchet state forward. Here we show that, because the protocol

π is defined to be correct in the theorem statement, the adversary can win the security game. We argue that s is fresh and that the adversary derives the same key as the Test session.

It is simple to check the two freshness conditions of KE-PCS are satisfied: (i) the only corrupt query that was issued was indeed only issued against $\hat{A}$, the peer to the actor of the Test s ($\hat{B}$), and this was before the creation of s, and (ii) since s does indeed have no origin-session, s is refreshed because of the honest matching sessions between $\hat{A}$ and $\hat{B}$ that completed before the creation of s. Hence, s is fresh.

Therefore, the adversary $\mathcal{A}$ may issue test(s) and compare the result to the session key in the memory of the unique session s' at the simulated $\hat{A}$-oracle. Since the protocol is correct and s matches s', we must have that s.key $= s'$.key. Hence, the adversary can easily win. In this case, the theorem is proved.

Case 2: s rejects

In this case the impersonation attack was unsuccessful. Intuitively, this must mean that Bob did inexorably ratchet state forward. Here we show that this leads to a contradiction on the post-network robustness of π. We argue that the marginal distribution at Σ could also be induced by a network adversary. Thus, s should have accepted because π is post-network robust.

Consider the network adversary $\mathcal{C}'$ that is equal to $\mathcal{C}$ except that the final message from $\hat{B}$ to $\hat{A}$ is dropped. That is, Alice sends a message to Bob and gets no reply. All of the state alterations of Σ were local and unaffected by the messages of Bob. Therefore, the real Alice could also have the same state as Σ with $\mathcal{C}'$. In fact, because the adversary corrupted Alice and did the same setup of Σ as the real Alice, both distributions of possible state are the same. But $\hat{B}$ rejected, which means that $\mathcal{C}'$ could also cause $\hat{B}$ to reject the real Alice. Since $\mathcal{C}'$ is a network adversary, π is not post-network robust. In this case the theorem is proved. □

A.3 Generic PCS Transforms

A.3.1 One-round transform

Consider the generic protocol transformation of Figure 4.5 for one-round protocols. The transformation takes a protocol π in the class ISM and transforms it into a protocol $\pi^\star$ not in ISM. The intuition is that, under the right conditions, the transformed protocol $\pi^\star$ should achieve PCS. In this section, we make this precise. Specifically, we prove Theorem 4.10.

Theorem 4.10. *Let π be any AKE protocol in the class $\mathsf{AKE} \cap \mathsf{ISM}$.*

(i) If π is secure in eCK$^\omega$, then $\pi^\star$ is secure in eCK$^\omega$-PCS.

(ii) If π is secure in $\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$, then $\pi^\star$ is secure in Ω_{AKE}.

Note that we are only aiming to prove key indistinguishability. As we know from Theorem 4.11, no one-round protocol is post-network robust; therefore, a trade-off between security and efficiently must always occur.

One might conclude that this transformation is useless then. However, the proof technique for our one-round protocol transformation allows us to then provide a security proof for a more practical two-round protocol (Theorem 4.13). Our two-round protocol transformation is more complicated in that it involves MACs and potential as well as actual intermediate secrets in memory, but the basic intuition of sharing and updating an intermediate secret is the same. We prove in Appendix A.3.2 that this protocol can be robust and achieve PCS.

To prove Theorem 4.10, we first show that $\pi^\star$ is at least as secure as π, and then we show that $\pi^\star$ additionally meets security in the PCS models, eCK$^\omega$-PCS and Ω_{AKE}.

Lemma A.1. *Let $X^\star \coloneqq$ eCK$^\omega$ or $X^\star \coloneqq \Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$. If π is secure in model $X^\star$, then so is $\pi^\star$.*

Proof. We prove the contrapositive. That is, we prove that if $\pi^\star$ is insecure in model $X^\star$, then so is π. For an adversary $\mathcal{A}_{\pi^\star}$ in model $X^\star$ that succeeds against protocol $\pi^\star$ with non-negligible probability, consider adversary $\mathcal{A}_\pi$, also in model $X^\star$ but

against protocol π. $\mathcal{A}_\pi$ proceeds in an almost identical way to $\mathcal{A}_{\pi^\star}$ except for minor differences. $\mathcal{A}_\pi$ ignores intermediate secrets, e.g. where $\mathcal{A}_{\pi^\star}$ computes $\mathrm{KDF}(\sigma, IS, 0)$, $\mathcal{A}_\pi$ instead computes $\mathrm{KDF}(\sigma)$. Where $\mathcal{A}_{\pi^\star}$ uses a $\mathsf{session\text{-}key}(s)$ query to reveal key $k^\star = \mathrm{KDF}(\sigma, IS, 0)$, which $\mathcal{A}_{\pi^\star}$ then uses in later computations, $\mathcal{A}_\pi$ uses the same $\mathsf{session\text{-}key}(s)$ query to reveal a different session key $k = \mathrm{KDF}(\sigma)$, which $\mathcal{A}_\pi$ uses instead (subject to the condition of $\mathcal{A}_\pi$ ignoring intermediate secrets as already specified). Subject to these conditions, $\mathcal{A}_\pi$ executes the same trace as $\mathcal{A}_{\pi^\star}$. Denote the Test session for $\mathcal{A}_{\pi^\star}$ (resp. $\mathcal{A}_\pi$) against protocol $\pi^\star$ (resp. π) by $s_{\pi^\star}$ (resp. s_π). Similarly let the Test session key $k_{\pi^\star}$ (resp. k_π) be computed as $\mathrm{KDF}(\sigma_{\pi^\star}, IS, 0)$ (resp. $\mathrm{KDF}(\sigma_\pi)$). As $\mathcal{A}_{\pi^\star}$ is a successful adversary, $s_{\pi^\star}$ is fresh. By the construction of $\mathcal{A}_\pi$ and the freshness conditions in $X^\star$, s_π is also fresh.

As session keys are not used to compute any other terms in the protocol, and as intermediate secrets are only used to compute other intermediate secrets or session keys, for all intents and purposes $\mathcal{A}_\pi$ implements the same attack as $\mathcal{A}_{\pi^\star}$. If $\mathcal{A}_{\pi^\star}$ wins against $\pi^\star$ by querying $\mathrm{KDF}(\sigma_{\pi^\star}, IS, 0)$, then $\mathcal{A}_{\pi^\star}$ must have deduced the input tuple $(\sigma_{\pi^\star}, IS, 0)$, and in particular $\sigma_{\pi^\star}$, which is a function of only the key exchange. But since $\mathcal{A}_\pi$ has the same trace as $\mathcal{A}_{\pi^\star}$, $\mathcal{A}_\pi$ is able to deduce σ_π and k_π. Therefore, if the adversary $\mathcal{A}_{\pi^\star}$ succeeds against $\pi^\star$ with this strategy, then $\mathcal{A}_\pi$ can succeed against π. If on the other hand $\mathcal{A}_{\pi^\star}$ wins against $\pi^\star$ while never querying $\mathrm{KDF}(\sigma_{\pi^\star}, IS, 0)$, then as the KDF is a random oracle, the only possibility is that this is a key replication attack. Clearly if $\mathcal{A}_{\pi^\star}$ can do this, then so can $\mathcal{A}_\pi$. □

Before we prove the main theorem of this section (Theorem 4.10), we make the following observation. Suppose an adversary has to derive an intermediate secret $IS_i = \mathrm{KDF}(\sigma, IS_{i-1}, 1)$. As the KDF is a random oracle, the only way of doing so is by evaluating the KDF on the input $(\sigma, IS_{i-1}, 1)$. If the adversary can do this, then she can also compute the session key that comes with the same session as IS_i, as $k = \mathrm{KDF}(\sigma, IS_{i-1}, 0)$. If the adversary can compute a session key, then she does not need to use a $\mathsf{session\text{-}key}$ query on the session (or any matching session). Note we do not have circular reasoning here as session keys are never used to compute anything

in the protocol, including intermediate secrets. This leads us to the following two lemmata, which we will use in addition to Lemma A.1 to prove Theorem 4.10.

Lemma A.2 is cumbersome to write, but intuitively it states that computing an intermediate secret from a session with uncompromised state is as difficult as computing the session key of that session.

Lemma A.2. *Take either (i)* $\Omega \coloneqq eCK^{\omega}$ *and* $\mathsf{KR} \coloneqq eCK^{\omega}$*-PCS or (ii)* $\Omega \coloneqq \Omega_{\mathsf{AKE} \cap \mathsf{ISM}}$ *and* $\mathsf{KR} \coloneqq \Omega_{\mathsf{AKE}}$*. Consider all adversaries* $\mathcal{A}_{\pi^\star}$ *in* KR *against protocol* $\pi^\star$ *such that the Test session* s *of* $\mathcal{A}_{\pi^\star}$ *is refreshed and consider sessions* s' *and* s'' *as in the definition of* s *being refreshed. Denote the intermediate secret computed by* s_{actor} *(and possibly* s_{peer} *if synchronised) in* s' *(and possibly* s''*) by* IS_i*. So in other words,* $IS_i = \mathrm{KDF}(\sigma_{i-1}, IS_{i-1}, 1)$ *for some key exchange information* σ_{i-1} *and previous intermediate secret* IS_{i-1}*. If protocol* π *is secure in* Ω*, then the probability of adversary* $\mathcal{A}_{\pi^\star}$ *computing the intermediate secret* IS_i *is negligible.*

Proof. Assume for contradiction that an adversary $\mathcal{A}_{\pi^\star}$ exists that can compute IS_i with non-negligible probability. Recall the observation that if $\mathcal{A}_{\pi^\star}$ can compute IS_i, then $\mathcal{A}_{\pi^\star}$ can also compute the session key of the same session s'. Therefore, we can assume without loss of generality that $\mathcal{A}_{\pi^\star}$ does not issue a session-key(s') (or, if synchronised, a session-key(s'') query).

Consider adversary $\mathcal{A}_{\pi}$ against (secure) protocol π in Ω that copies $\mathcal{A}_{\pi^\star}$ apart from in two ways: (i) $\mathcal{A}_{\pi}$ ignores intermediate secrets in the same way as in the proof of Lemma A.1 and (ii) $\mathcal{A}_{\pi}$ makes the same queries as adversary $\mathcal{A}_{\pi^\star}$ except where $\mathcal{A}_{\pi^\star}$ queries $\mathsf{test}(s)$, $\mathcal{A}_{\pi^\star}$ instead queries $\mathsf{test}(s')$, and $\mathcal{A}_{\pi}$ does not necessarily make the same guess query as $\mathcal{A}_{\pi^\star}$. We see that since $\mathcal{A}_{\pi^\star}$ always makes a Test session refreshed, it follows that s' in fresh. Therefore, if $\mathcal{A}_{\pi^\star}$ can compute IS_i with non-negligible probability, then $\mathcal{A}_{\pi}$ can compute the Test session key and succeed against π in Ω with non-negligible probability. But π is secure in Ω, which is a contradiction. □

Lemma A.3 generalises this result to a *sequence* of intermediate secrets, since they are chained together inside random oracle invocations. It states intuitively that if an

intermediate secret comes from a refreshed session, then the adversary is unable to compute any subsequent intermediate secret derived from it.

Lemma A.3. *Take either (i) $\Omega \coloneqq eCK^{\omega}$ and $\mathsf{KR} \coloneqq eCK^{\omega}$-PCS or (ii) $\Omega \coloneqq \Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$ and $\mathsf{KR} \coloneqq \Omega_{\mathsf{AKE}}$. Consider all adversaries $\mathcal{A}_{\pi^\star}$ in KR against protocol $\pi^\star$ such that the Test session s of $\mathcal{A}_{\pi^\star}$ is refreshed and consider sessions s' and s'' as in the definition of s being refreshed. Denote the intermediate secrets s_{actor} stores for s_{peer} as $IS_0, IS_1, IS_2 \ldots$ where the intermediate secret computed in session s' is IS_i and the intermediate secret used in the computation of the Test session key of session s is IS_j (so $j \geq i$). If π is secure in Ω, then the probability of adversary $\mathcal{A}_{\pi^\star}$ computing the intermediate secret IS_j is negligible.*

Proof. We see that $IS_j = \mathrm{KDF}(\sigma_{j-1}, IS_{j-1}, 1)$ for some key exchange information σ_{j-1}. Intermediate secrets are never sent or received, so for $\mathcal{A}_{\pi^\star}$ to compute IS_j, $\mathcal{A}_{\pi^\star}$ must compute the input IS_{j-1} (as well as σ_{j-1}). But $IS_{j-1} = \mathrm{KDF}(\sigma_{j-2}, IS_{j-2}, 1)$ for some key exchange information σ_{j-2}, and so on. By induction we see that for $\mathcal{A}_{\pi^\star}$ to compute IS_j, $\mathcal{A}_{\pi^\star}$ must compute IS_i. But by Lemma A.2 this can only be done with negligible probability. □

We are now ready to prove Theorem 4.10.

Theorem 4.10. *Let π be any AKE protocol in the class $\mathsf{AKE}\cap\mathsf{ISM}$.*

(i) If π is secure in eCK^{ω}, then $\pi^\star$ is secure in eCK^{ω}-PCS.

(ii) If π is secure in $\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$, then $\pi^\star$ is secure in Ω_{AKE}.

Proof. The high-level proof structure is as follows: For both (i) and (ii) we have already shown that $\pi^\star$ is at least as secure as π by Lemma A.1, so all that remains is to cover the cases of additional adversarial behaviour allowed against $\pi^\star$ but disallowed against π. In these additional cases, we argue as in Lemma A.2 and Lemma A.3 that a refreshed session protects the necessary intermediate secret; thus, the Test session key is secure. If $\mathcal{A}$ creates a fresh Test session we say it is *valid*. Below, we prove case (ii), but case (i) has the same reasoning.

(ii): Since the Ω_{AKE} model is identical to the $\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$ model except for the difference in the 5$^{\text{th}}$ predicate, it follows by Lemma A.1 that $\pi^\star$ will be secure in Ω_{AKE} if we can show that for all adversaries $\mathcal{A}$ that are valid in Ω_{AKE} but not in $\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$, $\mathcal{A}$ can only defeat $\pi^\star$ with negligible probability. In other words, such an adversary $\mathcal{A}$ must create a Test session s for which no origin session exists, and also issue a $\mathsf{corrupt}(s_{\text{peer}})$ query. We can rule out the case that $\mathcal{A}$ wins by doing this as well as never computing the Test session key $k = \text{KDF}(\sigma_j, IS_j, 0)$ by the same arguments as before. Now, by the 5$^{\text{th}}$ predicate in Ω_{AKE}, the Test session s must be refreshed, so there exists two other sessions s' and s'' as in the definition of s being refreshed. To compute the KDF on the inputs $(\sigma_j, IS_j, 0)$ required for the Test session key, the adversary needs the intermediate secret IS_j, but this cannot be computed with non-negligible probability by Lemma A.3. Case case (i) is similar. □

We can now prove Theorem 4.18, which formally captures the bonus per-peer randomness guarantees our PCS transform can provide. We explain this in detail in Section 4.4.

Theorem 4.18. *Let π be any AKE protocol in the class $\mathsf{AKE}\cap\mathsf{ISM}$. If π is secure in $\Omega_{\mathsf{AKE}\cap\mathsf{SL}}$, then $\pi^\star$ is secure in $\mathcal{S}_{\mathsf{SL}}$.*

Proof. This proof proceeds in a similar way to the proof of theorem 4.10, except there are more cases to consider. In particular, there are more scenarios where an adversary can be valid in $\mathcal{S}_{\mathsf{SL}}$ but not valid in $\Omega_{\mathsf{AKE}\cap\mathsf{SL}}$.

First note that Lemma A.2 and Lemma A.3 also apply for the case (i) $\Omega \coloneqq \Omega_{\mathsf{AKE}\cap\mathsf{SL}}$ and $\mathsf{KR} \coloneqq \mathcal{S}_{\mathsf{SL}}$. Similarly, Lemma A.1 also applies in the case $X^\star \coloneqq \Omega_{\mathsf{AKE}\cap\mathsf{SL}}$. Consider adversary $\mathcal{A}$ in $\mathcal{S}_{\mathsf{SL}}$ against $\pi^\star$. Denote the Test session by s with Test session key $k = \text{KDF}(\sigma_j, IS_j, 0)$. We partition into the events (i) V, (ii) $V^c \cap T_0^c$, (iii) $V^c \cap T_0 \cap S$ and (iv) $V^c \cap T_0 \cap S^c$; where V, T_0 and S are defined as follows:

V $\mathcal{A}$ is a valid adversary in $\Omega \coloneqq \Omega_{\mathsf{AKE}\cap\mathsf{SL}}$.

T_0 There exists an origin session for the Test session.

S $\mathcal{A}$ issues a $\mathsf{corrupt}(s_{\text{actor}})$ and a ($\mathsf{randomness}(s)$ or $\mathsf{cr\text{-}create}(s)$) query.

Event V

This case is covered by Lemma A.1.

Event $V^c \cap T_0 \cap S^c$

If $\mathcal{A}$ is a valid adversary in $\mathcal{S}_{\mathsf{SL}}$ but not in $\Omega \coloneqq \Omega_{\mathsf{AKE} \cap \mathsf{SL}}$, then it must be because of the difference in the 3rd, 4th or 5th freshness predicates as the 1st and 2nd are identical. If an origin session also exists, it must be because of the 3rd or 4th predicates. If $\mathcal{A}$ also does not issue a $\mathsf{corrupt}(s_{\text{actor}})$ and a $\mathsf{randomness}(s)$ (or $\mathsf{cr\text{-}create}(s)$) query, then the difference must be in the 4th predicate. In this case, since $\mathcal{A}$ is still a valid adversary in $\mathcal{S}_{\mathsf{SL}}$, it must be the case that there exists a session s' such that s is partially matching s', where the queries $\mathsf{corrupt}(s_{\text{peer}})$ as well as ($\mathsf{randomness}(s')$ or $\mathsf{cr\text{-}create}(s')$) have been issued, where s' is refreshed. By analogous reasoning to Lemma A.2 and Lemma A.3, the intermediate secret IS_j used in the computation of the Test session key k is safe, so we are done.

Event $V^c \cap T_0 \cap S$

If $\mathcal{A}$ is a valid adversary in $\mathcal{S}_{\mathsf{SL}}$ but not in $\Omega \coloneqq \Omega_{\mathsf{AKE} \cap \mathsf{SL}}$, then it must be because of the difference in the 3rd, 4th or 5th freshness predicates as the 1st and 2nd are identical. If an origin session also exists, it must be because of the 3rd or 4th predicates. If it is because of the 4th, see the analysis of event $V^c \cap T_0 \cap S^c$. If it is (possibly also) because of the difference in the 3rd predicate, then by the 3rd predicate of $\mathcal{S}_{\mathsf{SL}}$, s must be refreshed. By analogous reasoning to Lemma A.2 and Lemma A.3, the intermediate secret IS_j used in the computation of the Test session key k is safe, so we are done.

Event $V^c \cap T_0^c$

If $\mathcal{A}$ is a valid adversary in $\mathcal{S}_{\mathsf{SL}}$ but not in $\Omega \coloneqq \Omega_{\mathsf{AKE} \cap \mathsf{SL}}$, then it must be because of the difference in the 3rd, 4th or 5th freshness predicates as the 1st and 2nd are identical. If it is because of the difference between the 3rd and 4th predicates, see the analysis of Event $V^c \cap T_0 \cap S$. If it is (possibly also) because of the difference in the 5th predicate, then this case is covered by analogous reasoning to the proof of Theorem 4.10. □

A useful corollary of Theorem 4.10 and Theorem 4.18 is that one can use it to reverse engineer proofs of security; real-world protocols that use synchronised state may be complex and lack proofs of security, but if they are of the same form as $\pi^\star$ then it will suffice to prove the security of the simpler corresponding protocol of the same form as π.

A.3.2 Two-round transform

The previous PCS transformation did not provide post-network robust protocols. In this section, we prove that our two-round transformation of Figure 4.7 can produce protocols that achieve PCS as well as post-network robustness. As before, the transformation converts a protocol π subject to some restrictions into a protocol $\pi^\dagger$ not in ISM.

Intuitively, the major obstacle to post-network robustness is related to the two generals' problem: if a party updates its state, the outgoing message might be dropped. However, a protocol of more than one round can solve this problem by ensuring that both parties continually confirm to the other what they think the state should be. It can do this in the following ways: (i) the initial messages are authenticated with the current state; (ii) secrets are tied to the particular protocol role of each party; and (iii) there is a mechanism to handle the case when a final message is dropped. Specifically, each party maintains separate initiator and responder states. The latter contains not only the current IS value but also a collection of potential ones, corresponding to sessions that might have been accepted by initiator $\hat{A}$. Upon receipt of the final protocol message, the responder $\hat{B}$ updates its state and deletes all other potential values. However, in the case this final message is dropped, $\hat{B}$ also checks incoming initial messages against the current potential states and re-synchronises if necessary.

The formal description of $\pi^\dagger$ is depicted in Figure 4.7, and its message flows in Figure 4.8. It assumes that public DH keys are distributed for all parties and that π as before computes a pre-master secret from which session keys are derived.

Theorem 4.13. *Let π be a stateless protocol.*

(i) If π is secure in eCK$^\omega$, then $\pi^\dagger$ is secure in eCK$^\omega$-PCS.

(ii) If π is secure in $\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$, then $\pi^\dagger$ is secure in Ω_{AKE}.

(iii) If π is robust, then $\pi^\dagger$ is post-network robust.

The high-level argument for Theorem 4.13 is as follows: We alter the protocol $\pi^\star$ in minor steps until it becomes protocol $\pi^\dagger$, arguing about the security of each protocol along the way. First, we transform π into $\pi^\star$ to gain the claimed extra security by Theorem 4.10. We then consider an intermediate protocol $\pi^\star_{\mathsf{MAC}}$ and show it is at least as secure as $\pi^\star$. Finally, we show that $\pi^\dagger$ is at least as secure as $\pi^\star_{\mathsf{MAC}}$. At the end, we will have shown that $\pi^\dagger$ is at least as secure as $\pi^\star_{\mathsf{MAC}}$, which is at least as secure as $\pi^\star$, which is secure in the required model by Theorem 4.10. We will then additionally prove the post-network robustness of $\pi^\dagger$.

Proof sketch. We only prove that security in $\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$ implies security in Ω_{AKE}; the other implication is very similar.

First transform π into $\pi^\star$ using the PCS transformation of Figure 4.5. By Theorem 4.10, $\pi^\star$ is secure in Ω_{AKE}. Next, transform $\pi^\star$ into $\pi^\star_{\mathsf{MAC}}$, which is identical to $\pi^\star$ except each message m_j is replaced with $\langle m_j, \mu_j \rangle$ as in the transformation of Figure 4.7. Note that $\pi^\star_{\mathsf{MAC}}$ is not robust as it does not use a global $st^{\mathcal{R}}_{\hat{B}}.\text{potential}_{\hat{A}}$ variable as in the transformation of Figure 4.7. However, $\pi^\star_{\mathsf{MAC}}$ is also secure in Ω_{AKE}; if an adversary can succeed against $\pi^\star_{\mathsf{MAC}}$ in Ω_{AKE} with non-negligible probability, then essentially the same attack works on $\pi^\star$. If the attack requires the Test session being fresh and refreshed, then we can apply the random oracle assumption to show that no extra information is gained from $\pi^\star_{\mathsf{MAC}}$ over $\pi^\star$. Hence, by Lemma A.3 the attack must not involve the adversary gaining any knowledge of the *IS*. As such, the same attack (but ignoring the MACs) can work against $\pi^\star$. If, on the other hand, the attack works with the Test session not being refreshed, then no security of the IS is guaranteed and so the MAC is superfluous information. As such, an attack of this type on $\pi^\star_{\mathsf{MAC}}$ implies an attack on $\pi^\star$. So any attack on $\pi^\star_{\mathsf{MAC}}$ can be translated onto an attack on $\pi^\star$. Thus, $\pi^\star_{\mathsf{MAC}}$ is secure in Ω_{AKE}.

To prove $\pi^\dagger$ is at least as secure as $\pi^\star_{\mathsf{MAC}}$, we assume the existence of a successful adversary $\mathcal{A}_{\pi^\dagger}$ against $\pi^\dagger$ and use it to construct a successful adversary $\mathcal{A}_{\pi^\star_{\mathsf{MAC}}}$ against $\pi^\star_{\mathsf{MAC}}$. For the queries send, create and $\mathsf{cr\text{-}create}$, naive forwarding of queries and replies

in the simulation will fail, since the list of potential *IS* values is used in $\pi^\dagger$ to compute the MAC values. For example, consider the case where $\hat{A}$ and $\hat{B}$ engage in two pairs of sessions, where in both $\hat{A}$ is the initiator and $\hat{B}$ the responder. If $\mathcal{A}_{\pi^\dagger}$ does not send the final message of the first session $\hat{A} \to \hat{B}$, then the messages in the second session will differ. Indeed, $\pi^\dagger$ is precisely designed to detect this message loss and update anyway, so it will send a MAC with the new state; while $\pi^\star_{\mathsf{MAC}}$ will not, and therefore it will use the old state. But this is not a problem, because while simulating $\mathcal{A}_{\pi^\dagger}$, $\mathcal{A}_{\pi^\star_{\mathsf{MAC}}}$ can make an additional $\mathsf{send}(\hat{B}, \cdot)$ to $\hat{B}$ to complete the original session. The update procedures in $\pi^\star_{\mathsf{MAC}}$ and $\pi^\dagger$ are identical, so this causes the states to agree before the construction of the final message.

This will perfectly simulate $\hat{B}$'s reply, as if $\hat{B}$'s state updated because of the $st^{\mathcal{R}}_{\hat{B}}.\text{potential}_{\hat{A}}$ variable. It is not difficult to check the other cases, or that the simulation can still work if $\mathcal{A}_{pi^\dagger}$ ever decides to go back and finish a session that $\mathcal{A}_{\pi^\star_{\mathsf{MAC}}}$ automatically finished. The other queries can be simulated in the normal way. This shows that $\pi^\dagger$ is secure in Ω_{AKE}. □

We now give a symbolic argument for robustness. Let π be a stateless robust protocol, and $\pi^\dagger$ its transformation. Let $\#(t)$ denote the number of invocations of the KDF in the construction of a term t.

Lemma A.4. *For any network adversary $\mathcal{A}$, if at some time during the execution of $\mathcal{A}$ we have two values $st_1, st_2 \in st^{\mathcal{R}}_{\hat{B}}.potential_{\hat{A}}$, then $\#(st_1) = \#(st_2)$*

Proof. A value st' is added to a non-empty $st^{\mathcal{R}}_{\hat{B}}.\text{potential}_{\hat{A}}$ just when a message $g^x, \text{MAC}(st^{\mathcal{R}}_{\hat{B},\hat{A}}; \cdot)$ is received. Since $st' = KDF(\ldots, st)$ we have that $\#(st') = 1+\#(st)$. Thus, we have the stronger invariant that $\#(st') = 1 + \#(st^{\mathcal{R}}_{\hat{B},\hat{A}})$ at all times, which proves the lemma. □

Lemma A.5. *If $st^{\mathcal{I}}_{\hat{A},\hat{B}} = st_1$ and $st^{\mathcal{R}}_{\hat{B},\hat{A}} = st_2$, then $\#(st_1) \geq \#(st_2)$.*

Proof. Every state update $st \mapsto st'$ increases $\#(st)$ by one; thus, we need to show only that $\hat{A}$ performs this update before $\hat{B}$, but this indeed holds by induction on the trace length. If the final event in the trace updates $st^{\mathcal{R}}_{\hat{B},\hat{A}}$ then we have equality:

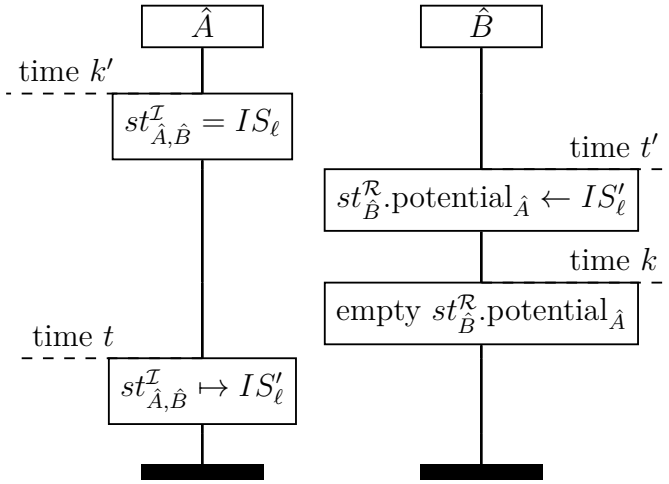


Figure A.1: Time ordering of events in the proof of Lemma A.7.

(i) If it was receipt of the third protocol message then $\hat{B}$ receives a message from $\hat{A}$ with state st and updates its state to st.

(ii) If it was receipt of the first protocol message then the same still holds.

Thus, for all traces we have the following inequality: □

Corollary A.6. $\#(st^{\mathcal{I}}_{\hat{A},\hat{B}}) - \#(st^{\mathcal{R}}_{\hat{B},\hat{A}}) \in \{0, 1\}$.

Proof. We need only show that $\hat{A}$ cannot update twice without an intermediate update by $\hat{B}$. The result then follows since each responder update restores equality. But this holds easily: $\hat{A}$ updates state only when receiving a message m from $\hat{B}$ with the new state, and thus $\#(st^{\mathcal{I}})$ can be at most one ahead of $\#(st^{\mathcal{R}})$ at the time m was sent, and hence certainly no further ahead. □

Lemma A.7. *If $st^{\mathcal{I}}_{\hat{A},\hat{B}} = IS$ at some time t, then either (i) $st^{\mathcal{R}}_{\hat{B},\hat{A}} = IS$ at time t, or $IS \in st^{\mathcal{R}}_{\hat{B}}.potential_{\hat{A}}$ at time t.*

Proof by symbolic argument. Note that state is linear; that is, that there is only one possible value of $st^{\mathcal{R}}_{\hat{A},\hat{B}}$ at a given time. We first claim that the state never repeats; that is, if $st^{\mathcal{I}}_{\hat{A},\hat{B}} = IS$ at times $t_1 < t_2$ then $st^{\mathcal{I}}_{\hat{A},\hat{B}} = IS$ at all times $t_1 \leq t \leq t_2$. This holds by case analysis on the rules that change the state, all of which make the change

$st \mapsto KDF(\ldots, st)$. Therefore, the number of function symbols in the state term only increases.

Now, let S_n be the statement that the lemma holds for all traces of length $\leq n$. We prove S_n for all n by induction. The base case is trivial; for the inductive case, we need only consider the rules that may change either the $\mathcal{I}$ or the $\mathcal{R}$ state, since if the final rule does not change either state then the induction hypothesis applies to the truncated trace.

Update initiator state

Suppose that we update $st^{\mathcal{I}}_{\hat{A},\hat{B}} : IS_\ell \mapsto IS'_\ell \neq IS_r = st^{\mathcal{R}}_{\hat{B},\hat{A}}$ at time t. By explicit key confirmation we have that $IS'_\ell \in st^{\mathcal{R}}_{\hat{B}}.\text{potential}_{\hat{A}}$, added at some time $t' < t$. If it was never removed then we are done, so suppose that it was removed by sending a message m to $\hat{B}$ at some time $t' < k < t$; choose the earliest possible $k > t'$. Finally, since the message causing it to be added must have been sent with state IS_ℓ at some time k', we have $k' < t' < k < t$. The removal can be upon $\hat{B}$'s receipt of either the first or the third protocol message, so there are two cases to consider.

Third. Suppose that m is the third and final protocol message. This will cause $st^{\mathcal{R}}_{\hat{B}}.\text{potential}_{\hat{A}}$to be emptied as part of the pre-accept computation. We derive a contradiction from Lemma A.4: suppose $\#(IS_\ell) = n$ so that $\#(IS'_\ell) = n + 1$. The construction of the third protocol message m occurs only when $\hat{A}$ changes state; by linearity, since $st^{\mathcal{I}}_{\hat{A},\hat{B}} = IS_\ell$ at k' and k it must be so for all times between k' and k. Therefore, m was constructed at $\hat{A}$ before k', say with state st such that $x := \#(st) < n$.

Now, since k was chosen as early as possible, we have that $IS_\ell \in st^{\mathcal{R}}_{\hat{B}}.\text{potential}_{\hat{A}}$ at time k since it was not removed beforehand. Since m is only accepted if the new state is currently in $st^{\mathcal{R}}_{\hat{B}}.\text{potential}_{\hat{A}}$, we have that $st \in st^{\mathcal{R}}_{\hat{B}}.\text{potential}_{\hat{A}}$ by Lemma A.4; therefore, $n = \#(IS_\ell) = \#(st) < n$, which is a contradiction.

First. Suppose that m is the first protocol message, causing $st^{\mathcal{R}}_{\hat{B}}.\text{potential}_{\hat{A}}$to be emptied because it is sent with a state $st \in st^{\mathcal{R}}_{\hat{B}}.\text{potential}_{\hat{A}}$. Again by Lemma A.4, since $IS'_\ell \in st^{\mathcal{R}}_{\hat{B}}.\text{potential}_{\hat{A}}$ we must have that $\#(st) = n + 1$. This is impossible, since

the final event in the trace has $\hat{A}$ update to IS'_ℓ, and therefore it sent no messages with such a state.

Update responder state

Suppose that the final rule, invoked at time t, changes the responder state. Let t' be the time of the last change to $st^{\mathcal{I}}_{\hat{A},\hat{B}}$, setting it say to IS_ℓ so that the induction hypothesis holds at time $t' < t$. Thus, we have either $st^{\mathcal{R}}_{\hat{B},\hat{A}} = IS_\ell$ or $IS_\ell \in st^{\mathcal{R}}_{\hat{B}}.\text{potential}_{\hat{A}}$. By Corollary A.6 we must have that $\#(st^{\mathcal{R}}_{\hat{B},\hat{A}})$ changes from $\#(IS_\ell) - 1$ to $\#(IS_\ell)$, since the corollary holds both before and after the change. The only messages that trigger such a change are sent by $\hat{A}$ with a state st having $\#(st) = \#(IS_\ell)$, and hence since states are linear $st = IS_\ell$. It follows that $st^{\mathcal{R}}_{\hat{B},\hat{A}} = st^{\mathcal{I}}_{\hat{A},\hat{B}}$ after the update, satisfying the conclusion of the lemma. □

Theorem A.8. *If $\pi \in \mathsf{SL}$ is robust then the transformed protocol $\pi^\dagger$ is post-network robust.*

The high-level argument here is as follows: For robustness it suffices to show that if the initiator has some state *IS* then the corresponding responder either has the same state or derived it in a previous session and did not erase it. Since updates occur upon receipt of a message depending on the previous state, parties can never move more than one update out of sync. Therefore, we just need to show that required states are not erased by the responder before they are needed. This comes down to a careful case analysis on the triggers for such an erasure.

Proof. If $\pi^\dagger$ is not post-network robust, then by definition this means that there exists some network adversary $\mathcal{A}$ such that, after $\mathcal{A}$ terminates, either (i) the running of $\mathcal{C}$ (the benign adversary that causes a pair of matching sessions between $\hat{A}$ and $\hat{B}$ and then terminates) does not result in $\hat{A}$ and $\hat{B}$ deriving the same session key, or (ii) $\mathcal{C}$ is unable to even create a matching session between $\hat{A}$ and $\hat{B}$. The latter case is ruled out by Corollary A.6. The former case is dealt with by Lemma A.7: the initial message sent by $\mathcal{C}$ has a state, which is already known to $\hat{B}$, and hence by construction the protocol accepts and derives equal session keys. □

B

Security Proof of Signal

B.1 A random oracle model proof of Signal

The proof considers different cases corresponding to the possible behaviour of an adversary. We first describe the high-level proof structure in Appendix B.1.1. We then recall the main theorem and provide the actual proof in Appendix B.1.2.

B.1.1 Signal proof structure overview

Security in this sense means that no efficient adversary can break the multi-stage key indistinguishability game for the two-party protocol Signal, parametrised by freshness condition fresh, with non-negligible probability. Suppose for contradiction that such an adversary $\mathcal{A}$ exists. Whatever the behaviour of the adversary, trivially (by the definition of the security experiment in Figure 5.6) it can only succeed when the Tested session $[s]$ is fresh. By Definition 5.4, this means that the $\mathsf{Test}(u, i, s)$ query satisfies:

(i) $\pi_u^i.status[s] = \texttt{accepted}$,

(ii) $\neg\pi_u^i.\mathsf{rev\_session}[s]$,

(iii) for all j such that $\pi_u^i.sid[s] = \pi_v^j.sid[s]$, $\neg\pi_v^j.\mathsf{rev\_session}[s]$, and

(iv) $\mathsf{clean}_{\pi_u^i.type[s]}(u, i, s)$

where v denotes $\pi_u^i.\mathsf{peerid}$, the identity of the intended peer to the Tested session, and $\mathsf{clean}_{\pi_u^i.type[s]}(u, i, s)$ is a cleanness clause as referenced in Definition 5.4 and subsequent definitions, further restricting the behaviour of the adversary. In the following overview, we consider the case that the Tested session is the initiator; the responder is analogous.

Overview of the case distinction

A high-level overview of the proof with its main game sequences and case distinctions is given in Figure B.1. Signal has many different types of stage, and we analyse each of them separately. Formally, we start with a sequence of generic game hops that apply to all cases. For instance, we rule out the low probability event that two randomly generated DH keys collide. We then guess which session the adversary will choose as the Test session; since there can be only polynomially many sessions this guess succeeds with non-negligible probability. (This is the source of the looseness in the proof.)

We then make a case distinction based on the stage type of the Tested session. Each stage type has its own cleanness predicate, and we deal with them in subcases. For example, if the adversary issues a query $\mathsf{Test}(u, i, [0])$, then we are in the analysis of stage [0], and if the stage type is `triple`, then we consider in turn the subcases where $\mathsf{clean}_{\mathtt{LM}}(u, i)$, $\mathsf{clean}_{\mathtt{EL}}(u, i, [0])$, or $\mathsf{clean}_{\mathtt{EM}}(u, i, [0])$ are true.

For each subcase, we perform an additional game hop, relying on one of the security assumptions in the statement of the theorem. The initial game will be ms-ind, and in the final games the session key will be replaced by a uniformly random value. By summing up the advantages along the way, we can obtain an overall bound on the success probability of the adversary.

The game hops in each of the subcases all build on a core type of reduction to GDH: the simulator queries for challenge values from the GDH oracle, inserting them into the game in place of certain DH keys and simulating the responses. We then intercept all adversarial calls to the random oracle, extracting the value taking the place of the GDH solution, and apply the DDH oracle to decide whether this value is indeed a solution. If it is then the simulator has broken GDH; if not, we continue the

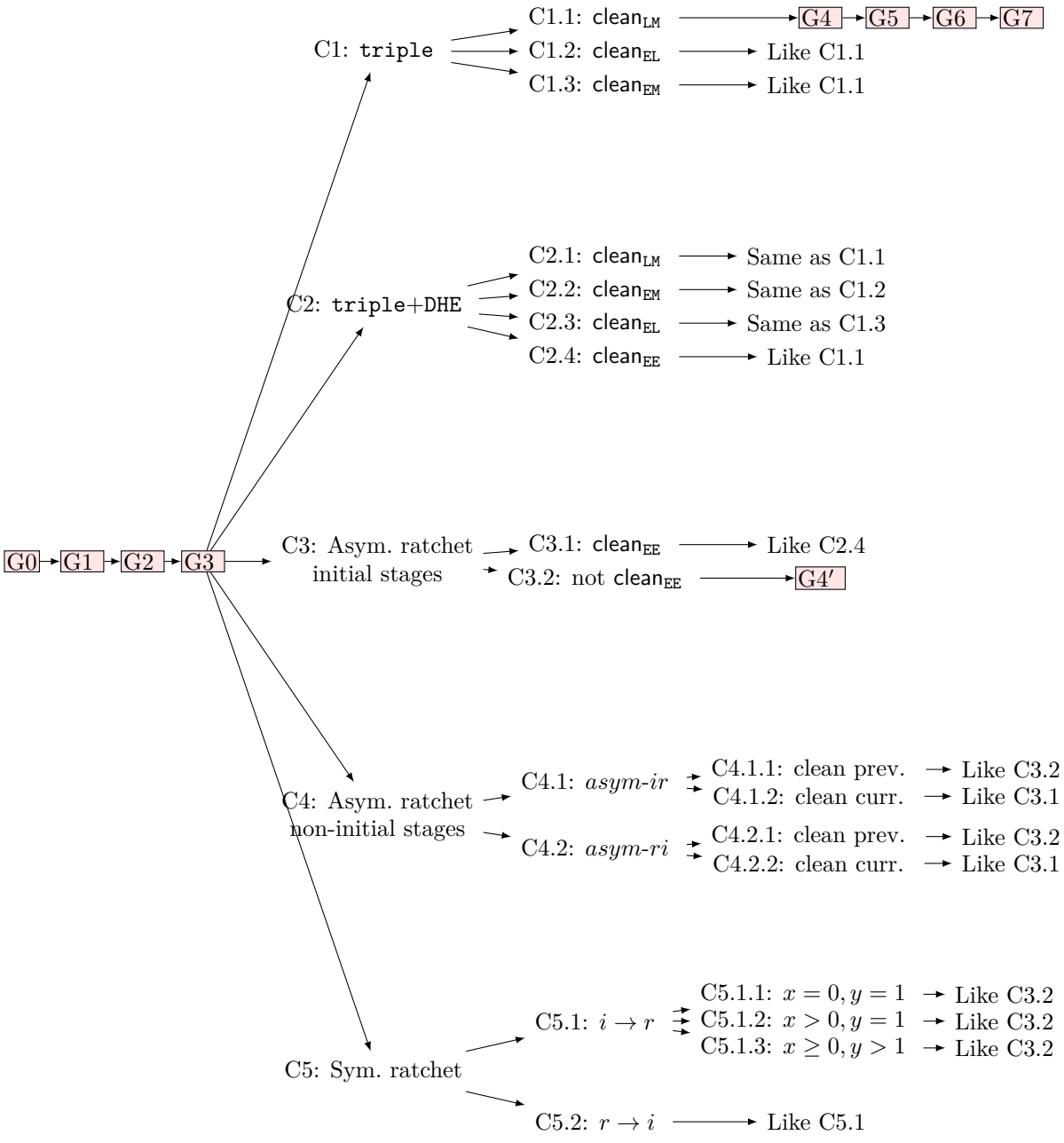


Figure B.1: High-level overview of the proof structure. Games are identified by G0, G1, etc. The main case distinctions are identified vy C1, C2, etc. G0 denotes the multi-stage security experiment from Section 5.4.2.

simulation. We will show that replacing the keys is not detectable by the adversary, and that violations of key indistinguishability imply solutions of the GDH challenge.

The challenger does not know in advance what adversary behaviour it is up against; only at the end of the game will the challenger know which of the clean predicates were satisfied. That is, the adversary might decide on the fly which session to Test and

which clean predicate to satisfy. As such, no global reduction can be given. This is not a problem: in our proof, we are ultimately just ruling out classes of attacks. If an attack exists, it corresponds to some specific adversary, which is covered by one of our cases.

Cases. We begin by considering the case that the $\mathsf{Test}(u, i, s)$ query was issued on the first stage ($s = [0]$). We break this up into two separate cases:

(i) the initial key exchange had stage type `triple` (so 3 separate pairs of DH shared secrets were used to compute the master secret ms), or

(ii) the initial key exchange had stage type `triple+DHE` (so 4 separate pairs of DH shared secrets were used to compute the master secret ms).

Case 1: `triple`. In the first case, where $\mathsf{Test}(u, i, [0])$ and $\pi_u^i.type[0] = \texttt{triple}$, we see by Definition 5.5 that the following condition must be satisfied:

$$\mathsf{clean}_{\texttt{LM}}(u, i) \vee \mathsf{clean}_{\texttt{EL}}(u, i, [0]) \vee \mathsf{clean}_{\texttt{EM}}(u, i, [0])$$

Case 2: `triple+DHE`. In the second case, where $\mathsf{Test}(u, i, [0])$ and $\pi_u^i.type[0] = \texttt{triple+DHE}$. By Definition 5.6, there is an additional disjunct $\mathsf{clean}_{\texttt{EE}}(u, i, [0])$, and we must have

$$\mathsf{clean}_{\texttt{LM}}(u, i) \vee \mathsf{clean}_{\texttt{EL}}(u, i, [0]) \vee \mathsf{clean}_{\texttt{EM}}(u, i, [0]) \vee \mathsf{clean}_{\texttt{EE}}(u, i, [0])$$

We consider each of these cases in turn, and by a game hopping argument replace the relevant keys by random values, allowing us subsequently to replace the session keys with random values.

Case 3: Asymmetric ratchet, initial stage. Next, we consider the security of the case that the $\mathsf{Test}(u, i, s)$ query was issued on the initial responder-to-initiator asymmetric stage $s = [\textbf{asym-ri}{:}1]$. We partition this into two cases corresponding to Definition 5.8: either the adversary has not issued queries that would break the cleanness of the root key from the first stage $s = [0]$; or the adversary did not inject malicious DH shares in either of the ephemeral shares used in the stage, which were

generated in stage $s = [0]$. That is, we consider the case that $\mathsf{Test}(u, i, s = [\textbf{asym-ri}{:}1])$ where $\pi_u^i.type[s] = \texttt{asym-ri}$, and apply Definition 5.8 to conclude that

$$\Big(\mathsf{clean}_{\pi_u^i.type[0]}(u, i, [0]) \wedge \mathsf{clean}_{\mathsf{state}}(u, i, [\textbf{asym-ir}{:}1])\Big) \vee \mathsf{clean}_{\texttt{EE}}(u, i, 0, 0)$$

In a similar fashion to the argument for the initial handshake, we replace certain DH values by values from a GDH challenger, reducing indistinguishability of the session key of this stage to hardness of GDH.

Case 4: Asymmetric ratchet, non-initial stages. We continue, considering the security of the case that the $\mathsf{Test}(u, i, s)$ query was issued in the x^{th} asymmetric responder-to-initiator stage $s = [\textbf{asym-ri}{:}x]$. That is, we consider the case that $\mathsf{Test}(u, i, s = [\textbf{asym-ri}{:}x])$, where $x \geq 2$ and $\pi_u^i.type[s] = \texttt{asym-ri}$. By Definition 5.8, we conclude that we must either have either $\mathsf{clean}_{\texttt{EE}}(u, i, x-1, x-1)$ or $\Big(\mathsf{clean}_{\texttt{asym-ri}}(u, i, [\textbf{asym-ri}{:}x-1]) \wedge \mathsf{clean}_{\mathsf{state}}(u, i, [\textbf{asym-ir}{:}x])\Big)$. Therefore, a similar argument holds.

We must also consider the case that the Test query was issued against an asymmetric stage of type `asym-ir` (i.e. a stage used to derive keys for the initiator to encrypt for the responder). The argument in this case is analogous but the cleanness predicates are subtly different and again vary depending on whether the stage is the first of its type. However, the core argument remains the same: we replace certain keys in the Tested session with values from the GDH challenger, in such a way that distinguishing session keys from random would give a GDH advantage.

Case 5: Symmetric ratchet. Finally, we consider symmetric stages. We partition into two cases:

(i) the first symmetric stage $y = 1$, where security follows from the asymmetric stage before it, which could be either the initial handshake or an asymmetric stage

(ii) a later symmetric stage $y > 1$, where security follows from cleanness of the previous symmetric update

B.1.2 Full proof of Signal

Conventions

We remark on a few conventions, which we adopt during the proof.

Many cases technically differ based on whether the actor of the Test session has the initiator or responder role. For example, the first session key derived by the initiator is from a sending chain, while the first one derived by the responder is from a receiving chain. Where the security arguments are identical except for obvious symmetries, we just consider the case of the initiator and leave the responder as analogous.

Signal uses HMAC and HKDF within the KDF invocations. We assume that the KDF invocations themselves (as defined in Figure 5.2) are random oracles, and thus need not make any assumptions on HMAC and HKDF specifically.

By $\Pr(break_i)$ we mean the probability that the adversary wins game G_i. We aim to show that $\Pr(break_0)$ is close to $1/2$. To avoid overfilling our subscripts, we overload where it is obvious which game is meant.

Theorem 5.10. *The Signal protocol is a secure multi-stage key exchange protocol under the Gap Diffie-Hellman assumption and assuming every* KDF *is a random oracle.*

Proof. We begin by performing a series of game hops that affect all potential cases. After these, the game hops diverge depending on which case we are considering.

Game hops for all cases

Game 0

This game equals the multi-stage security experiment described in Section 5.4.2. As such, the probability of the adversary winning this game is bounded above by $\Pr(break_0)$.

Game 1

In this game we ensure no collision of honestly generated DH public keys. Specifically, the challenger $\mathcal{C}$ maintains a list L of all DH private values (for *ik*, *prek*, *ek*, *eprek*, *rchk*) honestly generated during the game. If a DH private value appears twice, $\mathcal{C}$ aborts the simulation and the adversary automatically loses. For the execution of the adversary during the game, let $\mathsf{n_P}$ denote the total number of parties, $\mathsf{n_S}$ the maximum number of sessions, $\mathsf{n_M}$ the maximum number of medium-term keys per party, and $\mathsf{n_s}$ the maximum number of stages. We note that there are $\mathsf{n_P}$ long-term keys in the game, a maximum of $\mathsf{n_M}$ medium-term keys generated for each of the $\mathsf{n_P}$ parties for a maximum of $\mathsf{n_M n_P}$ medium-term keys, and a maximum of $\mathsf{n_s}$ ephemeral/ratchet keys per session for a total maximum of $\mathsf{n_S n_s}$ ephemeral/ratchet keys. This means a total maximum of $\mathsf{n_P} + \mathsf{n_P n_M} + \mathsf{n_S n_s}$ DH keys in the list L, every pair of which must not collide. There are $\binom{|L|}{2}$ such pairs of DH keys to consider in the game. Each DH key in L is in the same group of order q so collides with another key in L with probability $1/q$; therefore, we have the following bound:

$$\Pr(break_0) \leq \frac{\binom{\mathsf{n_P}+\mathsf{n_P n_M}+\mathsf{n_S n_s}}{2}}{q} + \Pr(break_1)$$

We now know that from this game onwards each honestly generated DH public key is unique. In future game hops, we will replace certain DH values with ones sampled by a GDH challenger. This means that if these replacement values collide, we must abort the game; therefore, we will be unable to answer the GDH challenge. This will appear in game $G4$. Fortunately, as we will see, the probability that the GDH challenger produces colliding GDH challenge values is negligible (with probability $1/q$).

Game 2

In this game, the challenger guesses in advance the session π_u^i against which the $\mathsf{Test}(u, i, s)$ query is issued: the challenger guesses a pair of indices $(u^*, i^*) \in [1..\mathsf{n_P}] \times [1..\mathsf{n_S}]$, and aborts (and the adversary automatically loses) if the adversary issues a

Test query $\mathsf{Test}(u, i, s)$ where $(u, i) \neq (u^*, i^*)$. This will occur with probability $1/\mathsf{n_S n_P}$, and hence we have the following inequality:

$$\Pr(break_1) \leq \mathsf{n_S n_P} \cdot \Pr(break_2)$$

Game 3

In this game, the challenger guesses an index $v^* \in [\mathsf{n_P}]$ and aborts if there exists a session π_v^j that matches the Test session π_u^i but $v \neq v^*$. Note that it might be the case that no such matching π_v^j exists, but this game ensures that if such a π_v^j does exist, v is unique and known in advance by the challenger.

We must first show that there can exist at most one identity v with the same session identifier as π_u^i (note v may have multiple sessions that match π_u^i as the responder does not contribute freshness in the Triple-DH case). Alice's session identifier for stage [0] contains ipk_v (the identity public key of the peer). In $G1$ we ensured that all DH values were unique, and hence the claim holds.

It follows that the challenger's guess is correct with probability $1/\mathsf{n_P}$, and so:

$$\Pr(break_2) \leq \mathsf{n_P} \cdot \Pr(break_3)$$

In this game, we do not guess the partner session because the responder does not always contribute an ephemeral key. As such, it is perfectly possible for v to have multiple sessions that match the Test π_u^i because the adversary may replay π_u^i's ephemeral key to multiple sessions of v, which only uses the same public key and medium term key. Only in `triple+DHE` does v contribute a ephemeral key (that is unique due to Game 1) and indeed in this case we will do another game hop to guess the unique partner session in advance.

Currently, we have derived the following probability bound:

$$\Pr(break_0) \leq \frac{\binom{\mathsf{n_P}+\mathsf{n_P n_M}+\mathsf{n_S n_s}}{2}}{q} + \mathsf{n_S {n_P}^2} \cdot \Pr(break_3)$$

At this point, we need to partition our analysis for individual cases, with the ultimate aim of bounding $\Pr(break_3)$ above, which is upper bounded by the maximum

success probability of the adversary in each case. Once we have bounded $\Pr(break_3)$, then we have bounded $\Pr(break_0)$ and we are done. Since this is $G3$, each different case begins with a hop to some $G4$.

C1: Initial key exchange: $\boldsymbol{type}[0] = \texttt{triple}$

First, we consider the security of Signal in the multi-stage key indistinguishability game against an adversary $\mathcal{A}$ that issues a $\mathsf{Test}(u, i, [0])$ query with $\pi_u^i.type[0] = \texttt{triple}$. By construction, the only way for the adversary to win (with non-negligible probability) is if $\mathsf{clean}_{\texttt{triple}}(u, i, [0])$ is true. We partition these scenarios into subcases. Note also that a $\mathsf{RevState}(u, i, [0])$ or $\mathsf{RevState}(v, j, [0])$ (where π_v^j is a session matching π_u^i if one exists) query will reveal nothing to the adversary, as there exists no previous state. Moreover, after our game hops, we will have replaced the Tested message key with a uniformly random value that is independent to all other keys, so other issued $\mathsf{RevState}$ queries will only reveal independent root keys and chain keys. As the state will be independent from the Tested session key, it will not help the adversary distinguish the Test session key from random. How to simulate reveal queries will be dealt with formally in the game hops.

We now begin to separate our analysis based on sub-clauses of the cleanness predicate. Let $E^{\texttt{triple}}$ be the event that an adversary $\mathcal{A}$ wins the $\mathsf{ms\text{-}ind}$ game by issuing a Test query $\mathsf{Test}(u, i, [0])$, such that $\pi_u^i.type = \texttt{triple}$, and let $E^{\texttt{triple}}_{\mathsf{clean}_{\texttt{LM}}}$ (resp. $E^{\texttt{triple}}_{\mathsf{clean}_{\texttt{EL}}}$, $E^{\texttt{triple}}_{\mathsf{clean}_{\texttt{EM}}}$) be the sub-case in which $\mathsf{clean}_{\texttt{LM}}(u, i)$ (resp. $\mathsf{clean}_{\texttt{EL}}(u, i, [0])$, $\mathsf{clean}_{\texttt{EM}}(u, i, [0])$) is also true. By definition of $\mathsf{clean}_{\texttt{triple}}$,

$$\Pr(E^{\texttt{triple}}) = \max\left\{\Pr\left(E^{\texttt{triple}}_{\mathsf{clean}_{\texttt{LM}}(u,i)}\right), \Pr\left(E^{\texttt{triple}}_{\mathsf{clean}_{\texttt{EL}}(u,i,0)}\right), \Pr\left(E^{\texttt{triple}}_{\mathsf{clean}_{\texttt{EM}}(u,i,0)}\right)\right\}$$

C1.1: Case $\boldsymbol{type}[0] = \texttt{triple}$ and $\mathsf{clean}_{\texttt{LM}}(u, i)$:

In this case, $\mathcal{A}_{\texttt{triple}}$ issued a $\mathsf{Test}(u, i, [0])$ query such that $\mathsf{clean}_{\texttt{LM}}(u, i)$ is upheld. For Test sessions where $\pi_u^i.role = \mathcal{I}$, this requires that $\mathcal{A}_{\texttt{triple}}$ has not issued the queries $\mathsf{RevLongTermKey}(u)$ or $\mathsf{RevMedTermKey}(v, n)$, where $\pi_u^i.\mathsf{peerpreid} = n$. Since we do

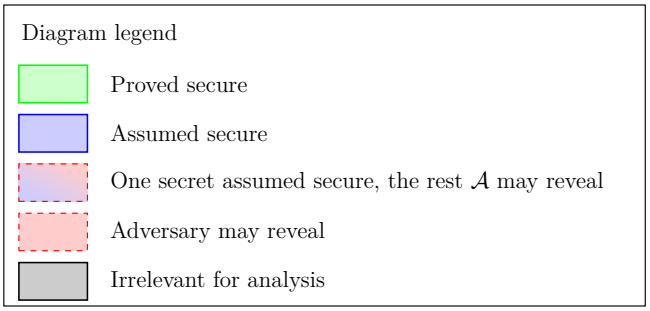


Figure B.2: Legend for the boxes in the following diagrams. boxes indicate secrets that the adversary may gain access to via with reveal queries (or by computing the secrets as a result of the reveal query). boxes indicate secrets that are replaced based on the challenge. boxes indicate secrets that the challenger is able to replace with random; this is crux of the argument for why the Test session key is secure in each particular case. boxes indicate secrets where one of which is assumed uncompromised, but the rest may be revealed by the adversary. boxes indicate secrets that may or may not be compromised; they are irrelevant for our analysis in the specific case we consider.

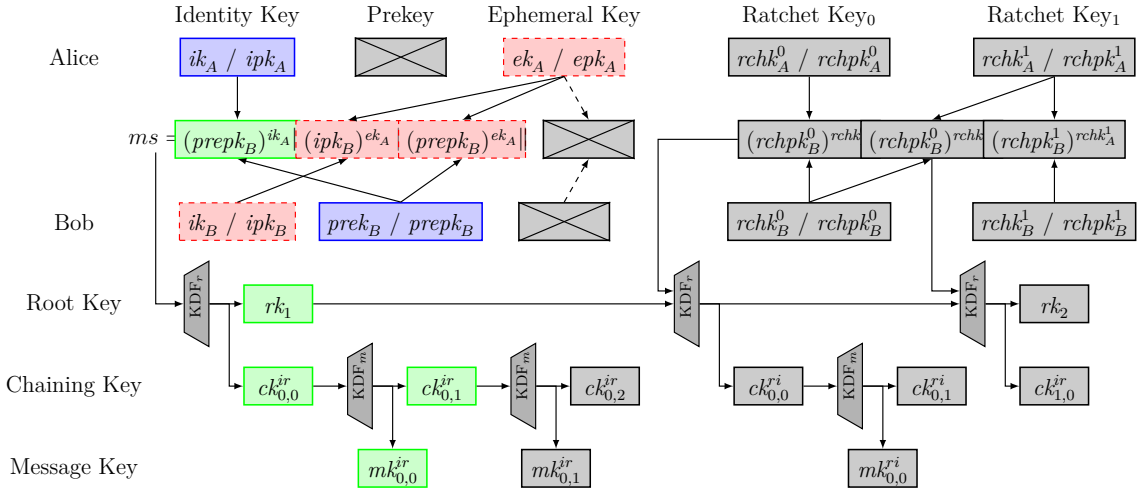


Figure B.3: A diagram showing the replacement of secrets in Game 6 of Case 1.1. Refer to Figure B.2 for the meanings of the coloured boxes.

not consider the signatures over the medium-term prekeys in our model, we may assume that π^i_u has received $prepk^n_v$ without modification.

Recall that a session derives a master secret $ms := g^{ik_u \cdot prek^n_v} \| g^{ek_u \cdot ik_v} \| g^{ek_u \cdot prek^n_v}$, and then assigns $rk^1 \parallel ck^{\textbf{sym-ir}:0,0} \leftarrow \text{HKDF}(ms)$.

Our goal will be to replace the session key with a random value so that the adversary cannot guess the hidden bit (game 7). Since we are working in the random oracle model and the session key is the output of a call to the random oracle, this means the adversary must query the random oracle on the exact input (game 6). We embed a Gap

Diffie–Hellman challenge into one of the components of the input to the random oracle. For this particular case (C1.1), which depends on the $\mathsf{clean}_{\mathtt{LM}}(u, i)$ condition, we embed the GDH challenge into the long-term key of party u and one of the medium-term keys of the peer v (hop from game 5 to game 6). To do this embedding, we must guess which of the medium-term keys of the peer is actually used (game 4). (There is also a minor technicality covered in game 5, described below.)

Game 4

In this game, the challenger guesses the index $n \in [1..\mathsf{n}_{\mathsf{M}}]$ of the signed prekey of the peer ($prek_v^n$) that the Test session will use in the execution of the protocol, and aborts if the guess is wrong. This yields that

$$\Pr(break_3) \leq \mathsf{n}_{\mathsf{M}} \cdot \Pr(break_4) \ .$$

Game 5

In this game, the experiment *does not* abort if ipk_u and $prepk_v^n$ are the same. (Recall that in Game 1, we added an abort event if any DH values were the same. We will soon want to employ a GDH challenger, but the two challenge public keys in a GDH challenger may (with small probability) be the same, so we need to re-allow that in our game hops.) Since the keys are elements of a group of order q, the probability that one of them equals the other is $1/q$. Thus, we have the following inequality:

$$\Pr(break_4) \leq 1/q + \Pr(break_5) \ .$$

Game 6

This game is the same as the last except it aborts if the adversary ever queries $g^{ik_u \cdot prek_v^n} := \mathrm{CDH}(ipk_u, prepk_v^n)$ as the first component of a call to the HKDF random oracle. Denote this event $abort_6$. Thus, we have the following inequality:

$$\Pr(break_5) \leq \Pr(break_6) + \Pr(abort_6) \ .$$

We now need to bound $\Pr(abort_6)$. To do so, we show that, whenever event $abort_6$ occurs, we can construct an algorithm $\mathcal{B}_0$ that can win the Gap Diffie–Hellman problem.

In the GDH experiment, $\mathcal{B}_0$ receives as input a DH pair (g^α, g^β) (for α and β unknown to $\mathcal{B}_0$), and has access to an oracle $\mathcal{O}_{\text{DDH}}$ that on input (g^x, g^y, g^z) returns 1 if and only if $g^{xy} = g^z$.

$\mathcal{B}_0$ will simulate game 5, except that it replaces ipk_u with g^α and $prepk_v^n$ with g^β. Because certain keys have been replaced with public keys whose corresponding private values are unknown to $\mathcal{B}_0$, we must define the actions that should be taken when these private values would normally be used in a computation. Cleanness implies that $\neg\mathsf{rev\_ltk}_u \wedge \neg\mathsf{rev\_mtk}_v^n$, so $\mathcal{B}_0$ will not need to answer any Reveal queries from $\mathcal{A}$ on these values. However, since $\mathcal{B}_0$ has replaced the long-term identity key and a medium-term public key of two parties, if $\mathcal{A}$ decides to direct parties u or v to execute the protocol in a non-Tested session, then $\mathcal{B}_0$ may need to perform simulations of concrete computations with the private keys α and β, despite not knowing them. There are three distinct types of sessions in which $\mathcal{B}_0$ may lack the private keys needed to compute the master secret ms of that session:

(i) a non-Tested session between user u and user v using $prek_v^n$ where u is the initiator;

(ii) a non-Tested session between user u and some other user (possibly v or not) where u is the responder;

(iii) a session between a user other than u and user v using $prek_v^n$ where v is the responder.

In session type (i), the simulator does not know $\text{CDH}(g^\alpha, g^\beta)$. This would be an input to the KDF computation of the session key (in fact this is the value that the simulator needs to find in the GDH game). In session type (ii), the simulator does not know $\text{CDH}(g^\alpha, g^e)$ for unknown, potentially maliciously chosen, e. In session type (iii), the simulator does not know $\text{CDH}(g^\beta, g^e)$ for unknown, potentially maliciously chosen, e.

In each of these types of sessions, $\mathcal{B}_0$ will pick random keys $rk^1, ck^{\textbf{sym-ir}:0,0}$ rather them deriving than via $\text{HKDF}(ms)$. $\mathcal{B}_0$ maintains a list of all sessions in which random

keys have been substituted: the list contains the random session keys as well as the public keys that should have been used to compute each component of the master secret. $\mathcal{B}_0$ must also ensure that key values used are consistent with any queries that $\mathcal{A}$ makes to the random oracle HKDF. We are concerned about queries of the form $g^{x_1} \| g^{x_2} \| g^{x_3}$. Before answering any such query, $\mathcal{B}_0$ goes through each entry in the above list of sessions: for each entry in the list, it uses its DDH oracle to check if the public keys that should have been used to compute each component of the master secret match the corresponding component (g^{x_1}, g^{x_2} or g^{x_3}) of this random oracle query. For example, for session type (i) this amounts to querying the DDH oracle $\mathcal{O}_{\text{DDH}}(g^{\alpha}, g^{\beta}, g^{x_1})$ and possibly $\mathcal{O}_{\text{DDH}}(g^{e}, g^{\beta}, g^{x_2})$. If all components, when queried in the DDH oracle, return 1, then $\mathcal{B}_0$ uses the randomly chosen keys from that element of the list as the random oracle response; otherwise, $\mathcal{B}_0$ samples a new random value as the random oracle response. Similarly session types (ii) and (iii) can be simulated.

(While the explanation above starts from $\mathcal{B}_0$ picking random session keys when simulating a session and then ensuring random oracle queries are answered consistently, $\mathcal{B}_0$ must also do the reverse: when simulating a session, before picking random keys $\mathcal{B}_0$ analogously use its DDH oracle to check if this matches a previous random oracle query, to ensure correct simulation.)

Note the session type (i) is special: if $\mathcal{O}_{\text{DDH}}(g^{\alpha}, g^{\beta}, g^{x_1}) = 1$, then the adversary has found the solution to the GDH problem for us, and $\mathcal{B}_0$ can use g^{x_1} as its answer to the GDH challenger. Moreover, this is exactly when the event $abort_6$ occurs.

$$\Pr(abort_6) = \epsilon_{\text{GDH}}(\mathcal{B}_0) \ .$$

Game 7

In this game, the experiment replaces the session key in the Test session with a uniformly random key from the same space. Because of the abort in game 6, we know that the adversary never queried the random oracle HKDF on the input ms that was used to compute the session key $rk^1 \parallel ck^{\textbf{sym-ir}:0,0}$ of the Test session. Thus, in the random oracle model we can conclude the following:

$$\Pr(break_6) = \Pr(break_7) \ .$$

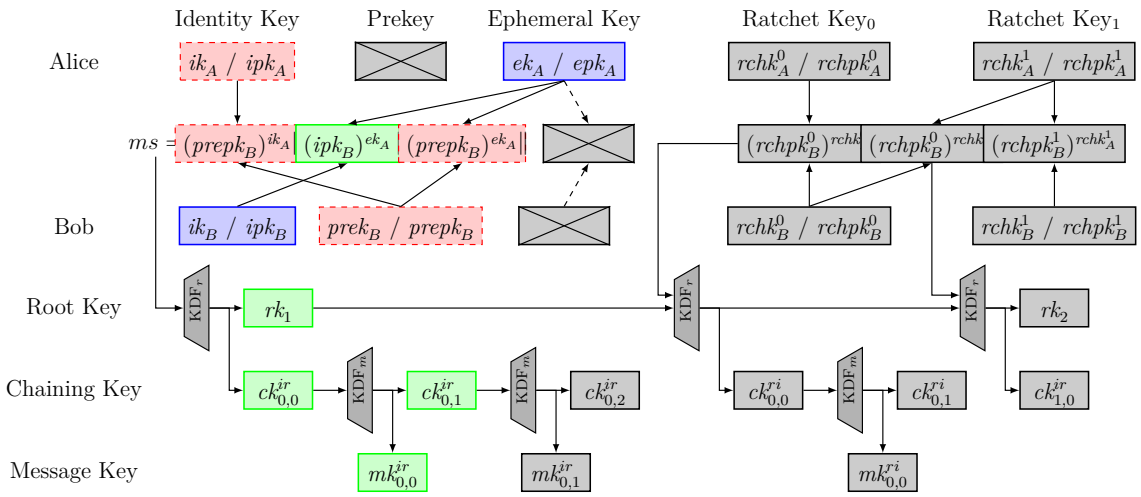


Figure B.4: A diagram showing the replacement of secrets in Game 5 of Case 1.2. Refer to Figure B.2 for the meanings of the coloured boxes.

Finally, since the session key is uniformly random and independent of the hidden bit, the adversary has no advantage in guessing the hidden bit and winning the experiment:

$$\Pr(break_7) = 1/2 \ .$$

C1.2: Case $\boldsymbol{type}[0] = \texttt{triple}$ and $\mathsf{clean}_{\mathtt{EL}}(u, i, 0)$

In this case the adversary $\mathcal{A}_{\mathtt{triple}}$ has issued a Test query $\mathsf{Test}(u, i, [0])$ such that $\mathsf{clean}_{\mathtt{EL}}(u, i, [0])$ is upheld. For Test sessions such that $\pi^i_u.role = \mathcal{I}$, this means that $\mathcal{A}_{\mathtt{triple}}$ has not issued $\mathsf{RevRand}(u, i, [0])$ and $\mathsf{RevLongTermKey}(v)$ where $v = \pi^i_u.peeripk$. For Test sessions such that $\pi^i_u.role = \mathcal{R}$, this means that $\mathcal{A}_{\mathtt{triple}}$ has not issued $\mathsf{RevLongTermKey}(u)$ and a $\mathsf{RevRand}(v, j, [0])$ such that $\pi^j_v.sid[0]$ matches the Test session $\pi^i_u.sid[0]$.

Game 4, 5, 6

The argument for this case is almost identical to that of the previous case, except we no longer need to guess the index of the long-term key of the responder or the ephemeral key of the initiator. The GDH challenge values g^α, g^β are inserted into the simulation in Game 5 in place of the ephemeral key of the initiator and the long-term key of the responder. Thus, we have the following inequality:

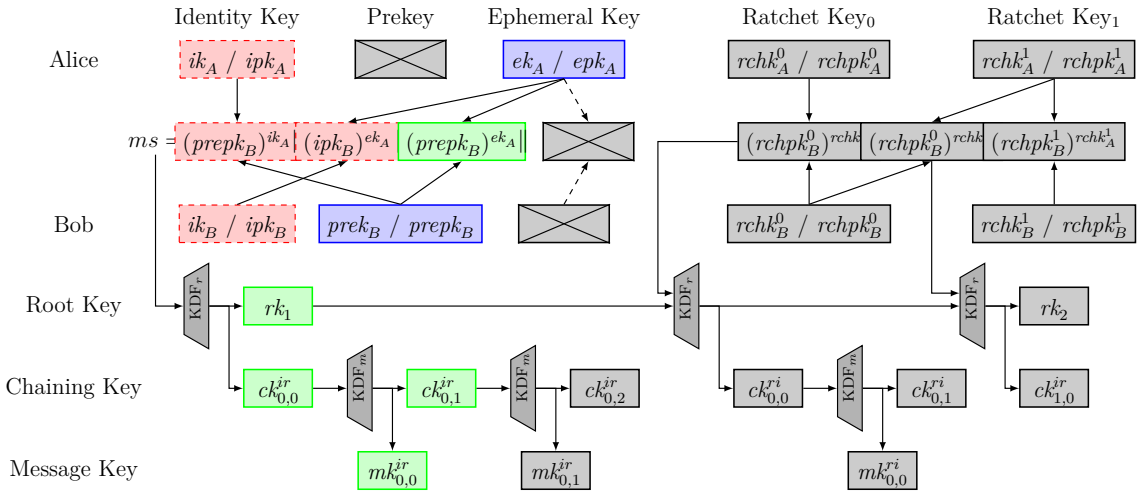


Figure B.5: A diagram showing the replacement of secrets in Game 6 of Case 1.3. Refer to Figure B.2 for the meanings of the coloured boxes.

$$\Pr(break_3) \leq 1/q + \epsilon_{\text{GDH}} + 1/2$$

C1.3: Case $\boldsymbol{type}[0] = \texttt{triple}$ and $\mathsf{clean}_{\texttt{EM}}(u, i, [0])$

Game 4, 5, 6, 7

In this case, the adversary $\mathcal{A}_{\texttt{triple}}$ has issued a Test query $\mathsf{Test}(u, i, [0])$ such that $\mathsf{clean}_{\texttt{EM}}(u, i, [0])$ is upheld. For Test sessions such that $\pi^i_u.role = \mathcal{I}$, this means that $\mathcal{A}_{\texttt{triple}}$ has not issued a $\mathsf{RevRand}(u, i, 0)$ and $\mathsf{RevMedTermKey}(v, \pi^i_u.\mathsf{peerpreid})$. Conversely, for Test sessions such that $\pi^i_u.role = \mathcal{R}$, this means that $\mathcal{A}_{\texttt{triple}}$ has not issued a $\mathsf{RevRand}(v, j, [0])$ such that $\pi^j_v.sid[0]$ matches the Test session $\pi^i_u.sid[0]$ and $\mathsf{RevMedTermKey}(u, \pi^i_u.prepk)$.

Again, this is analogous to before. We begin by guessing the index of the signed prekey of the responder, incurring a factor of $\mathsf{n_M}$. The Gap-DH challenge values g^α, g^β are inserted into the simulation in Game 6 in place of the ephemeral key of the initiator and the particular medium-term key of the responder used in the Test session. Thus, we have the following inequality;

$$\Pr(break_3) \leq \mathsf{n_M} \cdot (1/q + \epsilon_{\text{GDH}}) + 1/2$$

C2: Initial key exchange: $\textit{type}[0] = \texttt{triple+DHE}$

Recall that the initial key exchange can also have type `triple+DHE`, in which case cleanness requires that

$$\mathsf{clean}_{\texttt{LM}}(u,i) \vee \mathsf{clean}_{\texttt{EL}}(u,i,[0]) \vee \mathsf{clean}_{\texttt{EM}}(u,i,[0]) \vee \mathsf{clean}_{\texttt{EE}}(u,i,[0])$$

We now consider the case that the adversary has issued a Test query $\mathsf{Test}(u,i,[0])$ where the stage $\pi_u^i.type[0] = \texttt{triple+DHE}$. We note that the cases are the same as previously, with the additional case $\mathsf{clean}_{\texttt{EE}}(u,i,[0])$. As before, we define

- $E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\texttt{LM}}}$ to be the event that an adversary wins the multi-stage key indistinguishability game where $\mathcal{A}$ has issued a Test query $\mathsf{Test}(u,i,[0])$ and $\mathsf{clean}_{\texttt{LM}}(u,i)$ is upheld,
- $E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\texttt{EM}}}$ where $\mathcal{A}$ has issued a Test query $\mathsf{Test}(u,i,[0])$ and $\mathsf{clean}_{\texttt{EM}}(u,i,[0])$ is upheld,
- $E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\texttt{EL}}}$ where $\mathcal{A}$ has issued a Test query $\mathsf{Test}(u,i,[0])$ and $\mathsf{clean}_{\texttt{EL}}(u,i,[0])$ is upheld, and
- $E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\texttt{EE}}}$ where $\mathcal{A}$ has issued a Test query $\mathsf{Test}(u,i,[0])$ and $\mathsf{clean}_{\texttt{EE}}(u,i,[0],[0])$ is upheld.

The success probability of the adversary in the case $E^{\texttt{triple+DHE}}$ is bounded above by the maximum of $\Pr(E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\texttt{LM}}(u,i)})$, $\Pr(E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\texttt{EL}}(u,i,[0])})$, $\Pr(E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\texttt{EM}}(u,i,[0])})$, $\Pr(E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\texttt{EE}}(u,i,[0],[0])})$. The bounds for the previous summands are proved to be negligible under our cryptographic assumptions exactly as above, yielding the inequalities as desired. As before, the crucial proof step in each case is the Gap DH assumption. However, for this case it will also make a game hop like Game 3, where we also know Bob's unique matching session in advance. We can do this now because Bob has freshness in the handshake.

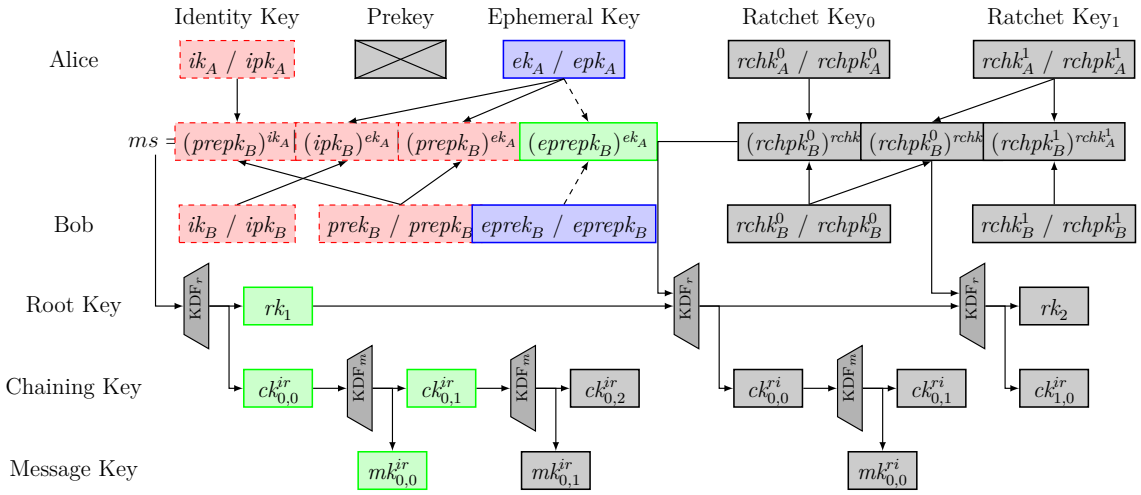


Figure B.6: A diagram showing the replacement of secrets in Game 6 of Case 2.4. Refer to Figure B.2 for the meanings of the coloured boxes.

C2.4: Case $\boldsymbol{type}[0] = \texttt{triple+DHE}$ and $\mathsf{clean}_{\mathsf{EE}}$

Game 4, 5, 6, 7

The final ephemeral-ephemeral case $E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\mathsf{EE}}}$ is analogous to previous cases except that in Game $G6$ ($E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\mathsf{EE}}}$), we need to replace the ephemeral values of both the initiator and the responder. (Since the simulator in $G4$ will never reuse ephemeral values in a different session, the simulation in this case is simpler and will not need to use its DDH oracle to maintain consistency.) We have to consider that the responder party generates a list of one-time ephemeral keys that new sessions (used in sessions executed by the responder) may use, and thus $G4$ now incurs a factor of $\mathsf{n_S}$. Thus, we have the following inequality:

$$\Pr(break_3) \leq \mathsf{n_S} \cdot (1/q + \epsilon_{\mathrm{GDH}}) + 1/2$$

C3: Asymmetric ratcheting, initial stage

We have now proved security of the initial key exchange, optionally including the ephemeral-ephemeral DH computation. We next move on to the asymmetric-ratcheting stages, in which Bob and Alice take turns generating new DH ephemerals and updating their root keys. The first asymmetric-ratcheting stage differs slightly from its successors

since it immediately follows the initial handshake, and we deal with it here now. Recall it is of type `asym-ri`, since it is performed when Bob wishes to send a message to Alice.

We consider an adversary $\mathcal{A}$ that issues a $\mathsf{Test}(u, i, s = [\textbf{asym-ri}{:}1])$ query, where stage s must have $type = \texttt{asym-ri}$. Note that the initial asymmetric stage is always of type `asym-ri` (messages from Alice to Bob before this stage are sent using the symmetric chain derived from the initial handshake), and thus in this section we do not need to consider *initial* stages of type `asym-ir`. We define

- $E^{\texttt{asym-ri}}$ to be the event that an adversary $\mathcal{A}$ wins the multi-stage key indistinguishability game by issuing a $\mathsf{Test}(u, i, s = [\textbf{asym-ri}{:}1])$ query,
- $E^{\texttt{asym-ri}}_{\mathsf{clean}_{\texttt{EE}}(u,i,[0])}$ to be the sub-event of $E^{\texttt{asym-ri}}$ satisfying $\mathsf{clean}_{\texttt{EE}}(u, i, [0], [0])$, and
- $E^{\texttt{asym-ri}}_{\mathsf{clean}_{\pi^i_u.type[0]}(u,i,[0])}$ to be the sub-event of $E^{\texttt{asym-ri}}$ satisfying

$$\mathsf{clean}_{\pi^i_u.type[0]}(u, i, [0]) \wedge \mathsf{clean}_{\mathsf{state}}(u, i, [\textbf{asym-ri}{:}1]).$$

It follows from our definition of freshness that

$$\Pr(E^{\texttt{asym-ri}}) \leq \max\left\{\Pr(E^{\texttt{asym-ri}}_{\mathsf{clean}_{\texttt{EE}}(u,i,[0])}), \Pr(E^{\texttt{asym-ri}}_{\mathsf{clean}_{\pi^i_u.type[0]}(u,i,[0])})\right\} \tag{B.1}$$

and we consider these two cases in turn, beginning with the case that $\mathsf{clean}_{\texttt{EE}}(u, i, [0], [0])$ is upheld.

C3.1: Case $s = [\textbf{asym-ri}{:}1]$, $\boldsymbol{type}[s] = \texttt{asym-ri}$ and $E^{\texttt{asym-ri}}_{\mathsf{clean}_{\texttt{EE}}}$

Game 4, 5, 6, 7

This case is dealt with similarly to case 2.4, with the only substantial differences being that the GDH challenge values are substituted into the ratchet keys $g^{rchk^u_{[0]}}$ and $g^{rchk^v_{[0]}}$ and we do not need to guess the index of the ratchet keys. Since we have a randomness-reveal query instead of queries for specific keys, the predicate $\mathsf{clean}_{\texttt{EE}}$ covers secrecy both of the handshake ephemerals and of the initial ratchet keys, which are generated at the same time. Thus, we have the following inequality:

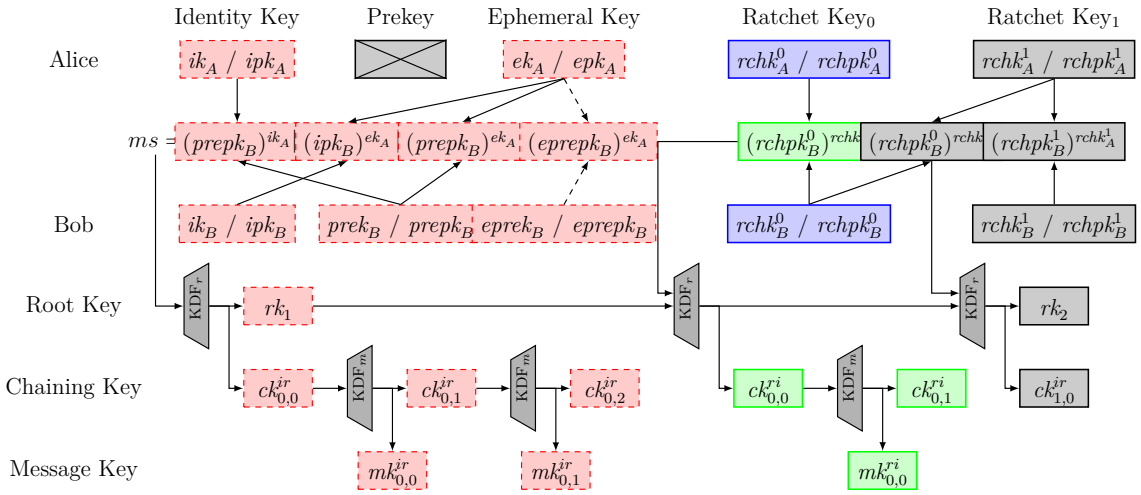


Figure B.7: A diagram showing the replacement of secrets in Game 6 of Case 3.1. Refer to Figure B.2 for the meanings of the coloured boxes.

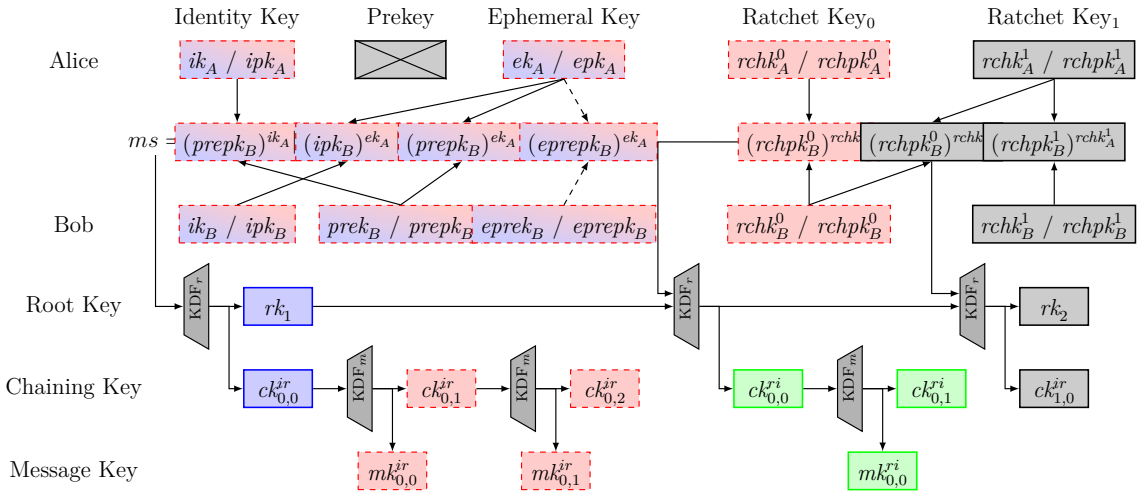


Figure B.8: A diagram showing the replacement of secrets in Game 7' of Case 3.2. Refer to Figure B.2 for the meanings of the coloured boxes.

$$\Pr(break_3) \leq 1/q + \epsilon_{\text{GDH}} + 1/2$$

C3.2: Case $s = [\textbf{asym-ri:1}]$, $\boldsymbol{type}[s] = \texttt{asym-ri}$ and $E^{\texttt{asym-ri}}_{\mathsf{clean}_{\pi^i_u.\boldsymbol{type}[0]}}(u,i,[0])$

In this case, cleanness comes from the initial key exchange (i.e. from one of its three or four disjuncts) and the fact that the adversary has not revealed the state linking the initial key exchange to this stage. The initial key exchange derives rk_1: we perform one game hop to replace that value with a uniformly random value; the game hop

is indistinguishable assuming the security of rk_1, which follows from Cases 1 and 2. Game 7′ is indicated below.

Game 7′

In this game, we replace the root key rk_1 derived in stage [0] by both the Test session and any matching peers with a value chosen uniformly at random. Thus, the advantage in this game is 0.

An adversary that can distinguish $G7'$ from its predecessor game can distinguish the root key from a random value. The root key was derived in the initial `triple` (or `triple+DHE`) handshake by applying KDF_r to the master secret ms. In Case 1 (or Case 2), we argued that all values derived from ms using HKDF were indistinguishable from random. Thus, an adversary that wins here contradicts the security of Case 1 (or Case 2). Recall that we denote with $\Pr(E^{\texttt{triple}})$ the probability of success for the adversary in breaking the key indistinguishability of Case 1, and with $\Pr(E^{\texttt{triple+DHE}})$ we denote the probability of success in breaking the key indistinguishability of Case 2. Given that only one of Case 1 or Case 2 applies (given how the initial key exchange is either of type `triple` or type `triple+DHE`), then the probability in distinguishing this change is $\max\{\Pr(E^{\texttt{triple}}), \Pr(E^{\texttt{triple+DHE}})\}$. Note that $\Pr(E^{\texttt{triple}}) \leq \Pr(E^{\texttt{triple+DHE}})$, given that $\Pr(E^{\texttt{triple}})$ is equal to the maximum of $(\Pr(E^{\texttt{triple}}_{\mathsf{clean}_{\texttt{LM}}(u,i)}), \Pr(E^{\texttt{triple}}_{\mathsf{clean}_{\texttt{EL}}(u,i,0)}), \Pr(E^{\texttt{triple}}_{\mathsf{clean}_{\texttt{EM}}(u,i,0)})$ Similarly, $\Pr(E^{\texttt{triple+DHE}})$ is equal maximum of $= \Pr(E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\texttt{LM}}(u,i)}), \Pr(E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\texttt{EL}}(u,i,[0])}), \Pr(E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\texttt{EM}}(u,i,[0])}), \Pr(E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\texttt{EE}}(u,i,[0],[0])})$. Note that the upper bounds on the first three subcases of both Case 1 and Case 2 are identical.

After replacing the root key rk_1, it is straightforward to see that it is impossible for the adversary to differentiate keys derived in this stage—chaining keys $ck^{\textbf{sym-ri}:x,0}$ and $ck^{\textbf{sym-ri}:x,1}$, messaging key $mk^{\textbf{sym-ri}:x,0}$, and intermediate value tmp from random: these are derived by applying KDF_r to rk_1 and then KDF_m to that result. Since both KDFs are modelled as random oracles, and the input to KDF_r is an independent uniformly random value, the adversary has no advantage is distinguishing this stage's session key from random. Thus, we have the following inequality:

$$\Pr(break_{7'}) = \Pr(E^{\texttt{triple+DHE}}) + 1/2$$

C4: Asymmetric ratcheting, non-initial stages

At this point we move on to arbitrary subsequent asymmetric stages. We assume that the initial handshake was of type `triple`, but the case of `triple+DHE` is analogous. The intuition for this part of the proof is essentially induction and post-compromise security:

- root keys provide security because they come from previous stages that are secure; or
- shared secrets derived from pairs of ephemeral keys provide security even if the root key at the time is compromised.

We first make a case distinction on the direction (`asym-ir` vs. `asym-ri`), and then deal with these cases in turn.

C4.1: Asymmetric ratcheting, $s = [\textbf{asym-ir:}x], x \geq 1$, $\textit{type}[s] =$ `asym-ir`

Definition 5.8 requires that one of the following conditions must be satisfied if $\mathsf{clean}_{\texttt{asym-ir}}(u, i, [\textbf{asym-ir}{:}x])$ is to hold.

- event $E^{\texttt{asym-ir}}_{\text{clean-prev}}$: $\mathsf{clean}_{\texttt{asym-ri}}(u, i, [\textbf{asym-ri}{:}x-1]) \wedge \mathsf{clean}_{\mathsf{state}}(u, i, [\textbf{asym-ir}{:}x])$
- event $E^{\texttt{asym-ir}}_{\text{clean-cur}}$: $\mathsf{clean}_{\texttt{EE}}(u, i, x-1, x-1)$

C4.1.1: Case $s = [\textbf{asym-ir:}x], x \geq 1$, $\textit{type}[s] =$ `asym-ir` and $E^{\texttt{asym-ir}}_{\textbf{clean-prev}}$

This case follows inductively like Case 3.2. This stage's message key (as well as the next root and chaining key) is derived by applying KDF_r to *tmp*, which was derived during stage $[\textbf{asym-ri}{:}x]$, and then KDF_m to the result. By an argument similar to

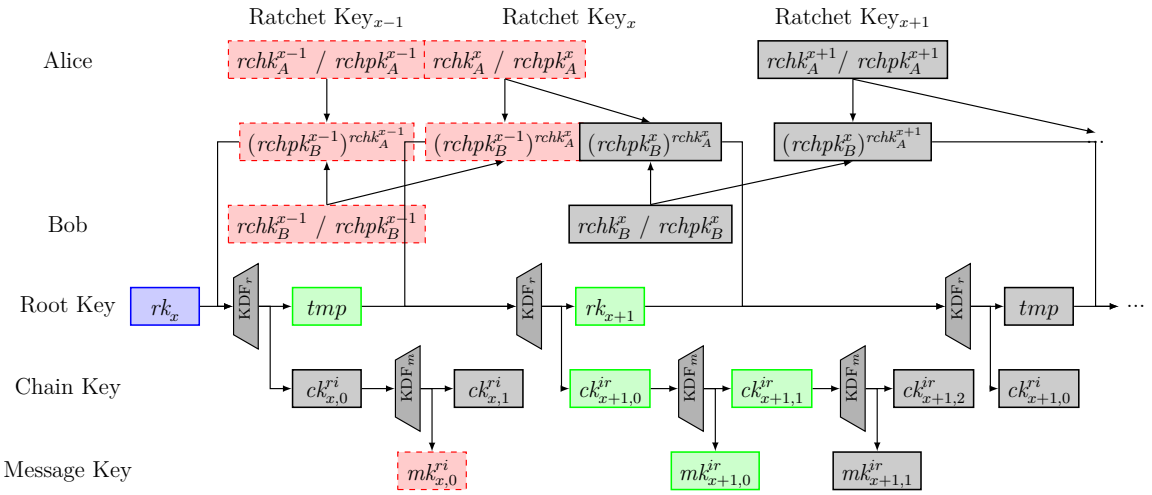


Figure B.9: A diagram showing the replacement of secrets in Game 7' of Case 4.1.1. Refer to Figure B.2 for the meanings of the different coloured boxes.

Case 3.2, we can replace tmp with a random key. Treating the KDF as a random oracle, this stage's message key, as well as the next root key rk_{x+1} and the symmetric chaining keys $ck^{\textbf{sym-ir}:x,0}$ and $ck^{\textbf{sym-ir}:x,1}$, are then indistinguishable from random. Thus, we have the following inequality:

$$\Pr(break_3) \leq 1/q + \Pr(E^{\texttt{triple+DHE}}) + \epsilon_{\text{GDH}} + 1/2$$

C4.1.2: Case $s = [\textbf{sym-ir}{:}x], x \geq 1$, $\boldsymbol{type}[s] = \texttt{sym-ir}$ and $E^{\texttt{sym-ir}}_{\textbf{clean-cur}}$

This case is analogous to Case 3.1, with key indistinguishability following from secrecy of the DH shared secret derived from ratchet keys. We first replace the ratchet public keys with challenge values from the Gap DH game, noting that $\mathsf{clean}_{\texttt{EE}}$ implies the existence of a unique session at Bob with the same sid as that of Alice's session. As before, an adversary that could distinguish this game from its predecessor allows us to answer the Gap DH challenge, violating that assumption. Indistinguishability of this stage's message key, as well as the next root and chaining keys enumerated in Case 4.4.1, then follows from applying the (random oracle) KDF to the (now independent) DH shared secret. Thus, we have the following inequality:

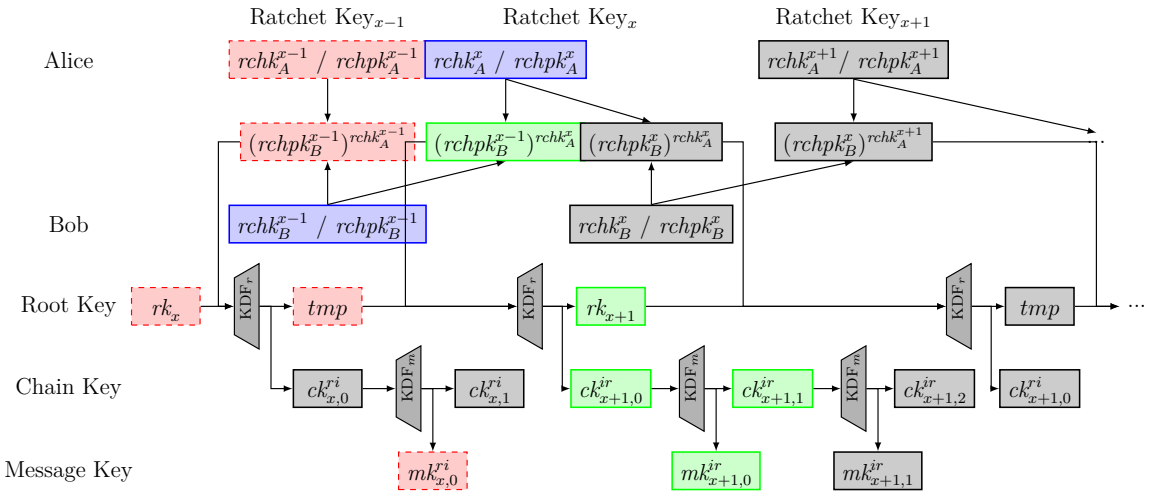


Figure B.10: A diagram showing the replacement of secrets in Game 7' of Case 4.1.2. Refer to Figure B.2 for the meanings of the different coloured boxes.

$$\Pr(break_3) \leq 1/q + 1/2 + \epsilon_{\text{GDH}}$$

C4.2: Asymmetric ratcheting, $s = [\textbf{asym-ri:}x], x > 1$, $\textbf{\textit{type}}[s] =$ `asym-ri`

Now we come to the case of non-initial asymmetric stages of type `asym-ri`. The proof here is nearly the same as in Case 4.1, except there is an extra KDF application: session keys derived by these stages are computed by first applying a KDF to derive an intermediate value *tmp*, and second applying another KDF to derive from *tmp* a session key.

Similarly, we partition our analysis into the following cases.

- event $E^{\texttt{asym-ri}}_{\text{clean-prev}}$: $\mathsf{clean}_{\texttt{asym-ir}}(u, i, [\textbf{asym-ir:}x])$
- event $E^{\texttt{asym-ri}}_{\text{clean-cur}}$: $\mathsf{clean}_{\texttt{EE}}(u, i, x, x-1)$

C4.2.1: Case $s = [\textbf{asym-ri:}x], x > 1$, $\textbf{\textit{type}}[s] =$ `asym-ri` and $E^{\texttt{asym-ri}}_{\textbf{clean-prev}}$

Once again the inductive argument here is analogous to Case 3.2: secrecy follows from the root key, and so we begin by replacing the root key with a random value.

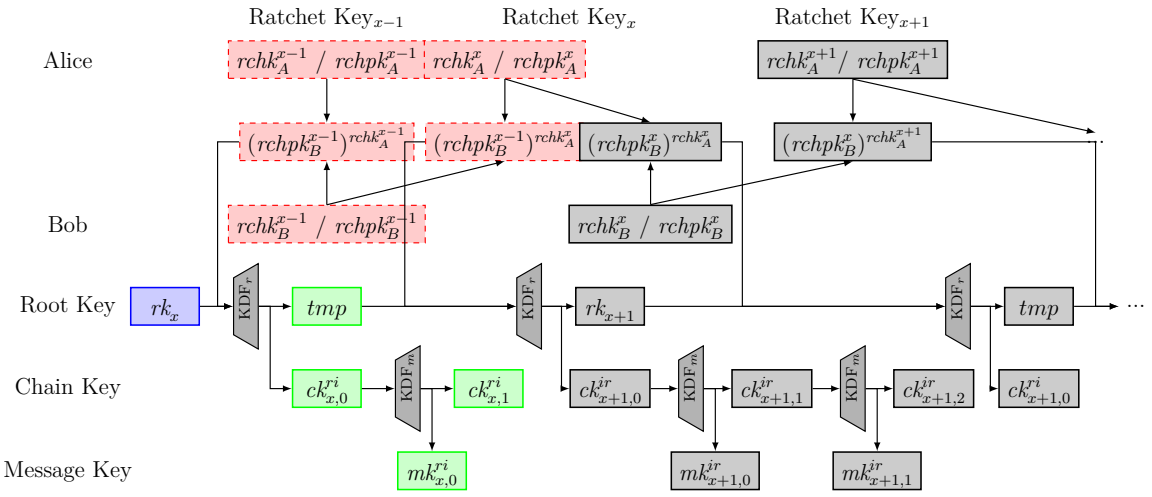


Figure B.11: A diagram showing the replacement of secrets in Game 7' of Case 4.2.1. Refer to Figure B.2 for the meanings of the different coloured boxes.

Detecting this change would violate the security properties of the previous stage, but after it the session key is easily proven indistinguishable from random. This stage's message key $mk^{\textbf{sym-ri}:x,0}$ as well as symmetric chaining keys $ck^{\textbf{sym-ri}:x,0}$ and $ck^{\textbf{sym-ri}:x,1}$ and intermediate value tmp, are all also then indistinguishable from random. Thus, we have the following inequality:

$$\Pr(break_3) \leq 1/q + \Pr(E^{\texttt{triple+DHE}}) + \epsilon_{\text{GDH}} + 1/2$$

C4.2.2: Case $s = [\textbf{asym-ri:}x], x > 1$, $\textbf{\textit{type}}[s] = \texttt{asym-ri}$ and $E^{\texttt{asym-ri}}_{\textbf{clean-cur}}$

For this case, we proceed similarly to Case 3.1. The DH shared secret can be shown indistinguishable under the Gap-DH assumption by replacing the ratchet public keys $rchk_u^x, rchk_v^{x-1}$ of the Test session and its matching peer with values from a GDH challenger. Indistinguishability of the stage's message key, symmetric chaining keys, and intermediate value tmp (as enumerated in case 4.2.1) all follow in turn from applying a (random oracle) KDF to (now independent) secret values. Thus, we have the following inequality:

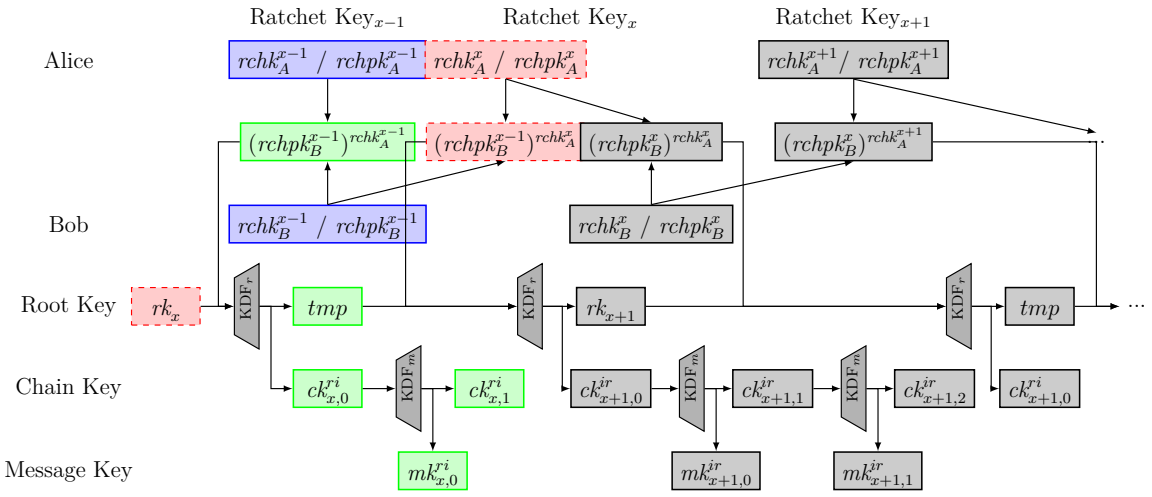


Figure B.12: A diagram showing the replacement of secrets in Game 7 of Case 4.2.2. Refer to Figure B.2 for the meanings of the different coloured boxes.

$$\Pr(break_3) \leq 1/q + \epsilon_{\text{GDH}} + 1/2$$

C5: Symmetric ratcheting: $\boldsymbol{type}[s] \in \{\texttt{sym-ir}, \texttt{sym-ri}\}$

We finally arrive at the case of Signal in the multistage key indistinguishability game against an adversary $\mathcal{A}$ that issues a $\mathsf{Test}(u, i, [\textbf{sym-ir}{:}x{,}y])$ or $\mathsf{Test}(u, i, [\textbf{sym-ri}{:}x{,}y])$ query against some some symmetric stage. In all subcases, we will show that the probability of winning is $1/2$

C5.1: Symmetric ratcheting, $s = [\textbf{sym-ir}{:}x{,}y], \boldsymbol{type}[s] = \texttt{sym-ir}$

We partition into the three different freshness conditions for the case [**sym-ir**:x,y]. We then cover the case of [**sym-ri**:x,y] similarly. The intuition is that for the first stage, the symmetric keys are derived from an asymmetric update and their secrecy follows from the previous cases. For subsequent stages, we have security due to the recursive nature of the freshness condition: we can replace the chain key used to derive the message key with randomness; if the simulation did not work, then the adversary could attack the previous stage, which is a contradiction to security of previous cases because the previous stage is fresh. In all symmetric stages, no new ephemeral keying material

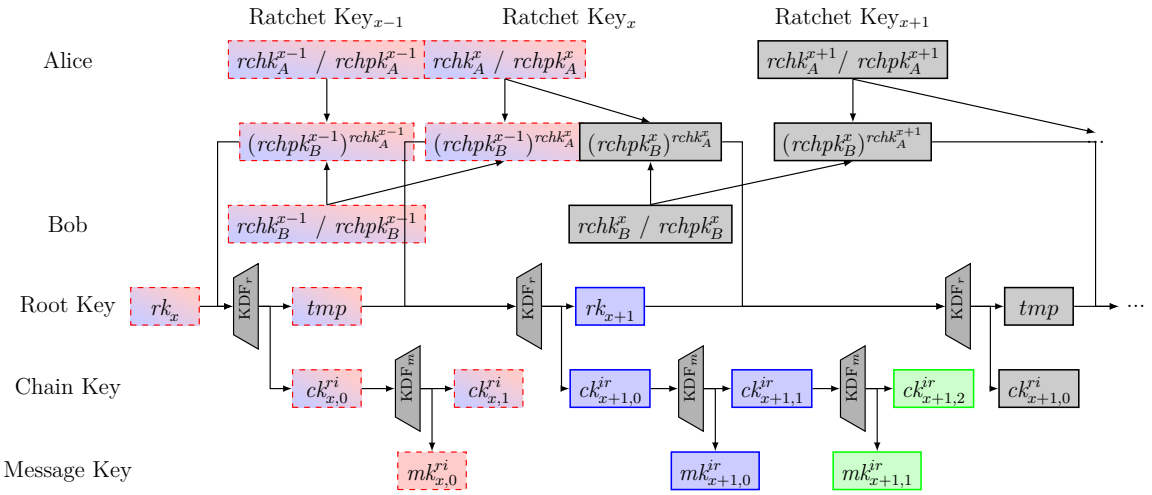


Figure B.13: A diagram showing the replacement of secrets in Game 7' of Case 5.1.2s. All other subcases follow a similar replacement strategy. Refer to Figure B.2 for the meanings of the different coloured boxes.

is introduced, so security depends solely on the chaining state not being leaked, which is guaranteed for these cases by $\mathsf{clean}_{\mathsf{state}}$.

(Recall that the case $y = 0$ is performed as part of the message key derivation in the previous asymmetric update so that the first symmetric stage derives key number 1.)

C5.1.1: Case $s = [\textbf{sym-ir:}x,y], x = 0, y = 1,$ $\boldsymbol{type}[s] = \texttt{sym-ir}$

This stage's messaging key is derived by applying a $\mathrm{KDF_m}$ to $ck^{\textbf{sym-ir}:0,1}$, which was derived during the initial `triple` or `triple+DHE` handshake. By Case 3.2, $ck^{\textbf{sym-ir}:0,1}$ is indistinguishable from random. Like the argument in Case 3.2, treating $\mathrm{KDF_m}$ as a random oracle, this message key of this stage, as well as the next chain key $ck^{\textbf{sym-ir}:0,2}$, are indistinguishable from random.

C5.1.2: Case $s = [\textbf{sym-ir:}x,y], x > 0, y = 1,$ $\boldsymbol{type}[s] = \texttt{sym-ir}$

This stage's messaging key is derived by applying a $\mathrm{KDF_m}$ to $ck^{\textbf{sym-ir}:x,1}$, which was derived during asymmetric-second-stage [**asym-ir**:x]. By Case 4.1, $ck^{\textbf{sym-ir}:x,1}$ is indistinguishable from random. Like the argument in case C.2, treating $\mathrm{KDF_m}$ as a

random oracle, this message key of this stage, as well as the next chain key $ck^{\textbf{sym-ir}:x,2}$, are indistinguishable from random.

C5.1.3: Case $s = [\textbf{sym-ir}{:}x,y], x \geq 0, y.1$, $\textbf{\textit{type}}[s] = \texttt{sym-ir}$

This stage's messaging key is derived by applying a KDF_m to $ck^{\textbf{sym-ir}:x,y}$, which was derived during symmetric stage $[\textbf{sym-ir}{:}x]y - 1$. By Case 5.1.1, 5.1.2, or induction on Case 5.1.3, $ck^{\textbf{sym-ir}:x,y-1}$ is indistinguishable from random. Like the argument in case 3.2, treating KDF_m as a random oracle, this message key of this stage, as well as the next chain key $ck^{\textbf{sym-ir}:x,y+1}$, are indistinguishable from random.

C5.2: Symmetric ratcheting, $s = [\textbf{sym-ri}{:}x,y]$, $\textbf{\textit{type}}[s] = \texttt{sym-ri}$

These cases are analogous to Case 5.1, by symmetry: cleanness is defined in the same recursive manner for both `sym-ir` and `sym-ri` stages, except that the base cases differ. Thus, the initial game hops are analogous to those in the `asym-ir` and `asym-ri` respectively, and the subsequent inductive argument is analogous to Case 5.1.3. □

B.2 A standard model proof of Signal

One drawback of the proof as it currently stands is that it sits within the random oracle model. As we have already mentioned in Chapter 2, this assumption may be unsettling for some.

Therefore, in this section, we outline the process in which one could attempt a standard model proof of the Signal Protocol in our security model. This is not a straightforward task; a naive attempt would be by retaining the current structure of our proof and replacing Gap-DH assumptions with a pair of DDH and PRF security assumptions. However, this approach does not fit. Similarly to previous proofs of TLS [58, 78], we find ourselves relying instead on the PRF-ODH assumption. There are many different variants of the PRF-ODH assumption throughout the literature; see [34] for a detailed exposition of these variants. However, the crux of the assumption is as follows. Given g^u and g^v, computing a function $/PRF(g^{uv}, x)$ should be hard,

even with choice of x and an oracle that computes values like $\text{PRF}(g^{uw}), x')$ and $\text{PRF}(g^{wv}, x')$ for chosen $w \neq u, v$.

Roughly speaking, the reason a PRF-ODH assumption needs to be used is because there are long-lived keys used in key computations in Signal that must be simulated when they are substituted out in the game hops. Before, we used a random oracle, which, by definition, gives random replies that the simulator could just choose randomly but consistently, to simulate these key computations. Now, we need a Diffie–Hellman oracle from the PRF-ODH assumption to carry out the simulation. One can hopefully see that because Signal computes keys using a PRF on Diffie–Hellman values, it seems at least plausible that some version of a PRF-ODH assumption may work for a proof.

Readers may ask why we cannot simply retain the current structure of our proof and replace the Gap-DH assumptions with a single PRF-ODH step. As discussed in Chapter 5 (and wholly unlike TLS), the Diffie–Hellman values used in Signal can be long-term, medium-term or ephemeral. These keys are also used in different combinations in key derivations. This difference requires that we not use a single PRF-ODH assumption, but a range of PRF-ODH assumptions that are parameterised by how many times the challenger will generate $\text{PRF}(g^{uv'}, x)$ and $\text{PRF}(g^{u'v}, x)$ values upon being queried for a "left" or "right" PRF-ODH oracle $(g^{v'}, x)$ or $(g^{u'}, x)$ (where $g^{u'}$ and $g^{v'}$ are not one of the DH challenge values (g^u, g^v) given by the challenger). We give a formal definition below from the work from [34].

Definition B.1 (Generic PRF-ODH assumption)**.** *Let $\mathbb{G}$ be a cyclic group of order q with generator g. Let* $\text{PRF} : \mathbb{G} \times \{0,1\}^* \to \{0,1\}^\lambda$ *be a pseudo-random function that takes as key input an element $k \in \mathbb{G}$ and an arbitrary-length salt value $x \in \{0,1\}^*$ as input, and outputs a value $y \in \{0,1\}^\lambda$.*

*We define a generalised security notion lr-*PRF-ODH*, which is parameterised by $l, r \in \{\mathsf{n}, \mathsf{s}, \mathsf{m}\}$, indicating how often the adversary is able to query a certain left oracle or right oracle (denoted* ODH_u *and* ODH_v *respectively) where* n *indicates that no query is allowed,* s *indicates that a single query is allowed, and* m *indicates that arbitrarily many queries are allowed to the respective oracle. Consider the lr-*PRF-ODH *security experiment depicted in Figure B.14.*

$\underline{\mathsf{Exp}^{lr\text{-}\mathsf{PRF\text{-}ODH}}_{\mathrm{PRF},\mathbb{G},g,q}(\mathcal{A})}$:

1: $\mathsf{b} \xleftarrow{\$} \{0,1\}$, $c_u \leftarrow 0$, $c_v \leftarrow 0$
2: $u \xleftarrow{\$} \mathbb{Z}_q$, $\mathbf{X} \leftarrow \emptyset$
3: **if** $l = \mathsf{m}$ **then**
4: $\quad x^* \leftarrow \mathcal{A}(\mathbb{G}, g, q, g^u)^{\mathsf{ODH}_u}$
5: **else**
6: $\quad x^* \leftarrow \mathcal{A}(\mathbb{G}, g, q, g^u)$
7: $v \xleftarrow{\$} \mathbb{Z}_q$
8: **if** $\{g^{uv}, x^*\} \in \mathbf{X}$ **then**
9: $\quad$ **return** $\bot$
10: $y_0 \leftarrow \mathrm{PRF}(g^{uv}, x^*)$, $y_1 \xleftarrow{\$} \{0,1\}^\lambda$
11: $\mathbf{X} \leftarrow \mathbf{X} \cup \{g^{uv}, x^*\}$
12: $b' \leftarrow \mathcal{A}(y_b)^{\mathsf{ODH}_u, \mathsf{ODH}_v}$
13: **return** $(b' = b)$

$\underline{\mathsf{Exp}^{lr\text{-}\mathsf{sPRF\text{-}ODH}}_{\mathrm{PRF},\mathbb{G},g,q}(\mathcal{A})}$:

1: $\mathsf{b} \xleftarrow{\$} \{0,1\}$, $c_u \leftarrow 0$, $c_v \leftarrow 0$
2: $u, v \xleftarrow{\$} \mathbb{Z}_q$, $\mathbf{X} \leftarrow \emptyset$
3: **if** $(l = \mathsf{m}) \wedge (r = \mathsf{m})$ **then**
4: $\quad \mathcal{A}(\mathbb{G}, g, q, g^u, g^v)^{\mathsf{ODH}_u, \mathsf{ODH}_v} \to x^*$
5: **if** $(l = \mathsf{m}) \wedge (r \neq \mathsf{m})$ **then**
6: $\quad \mathcal{A}(\mathbb{G}, g, q, g^u, g^v)^{\mathsf{ODH}_u} \to x^*$
7: **if** $(l \neq \mathsf{m}) \wedge (r = \mathsf{m})$ **then**
8: $\quad \mathcal{A}(\mathbb{G}, g, q, g^u, g^v)^{\mathsf{ODH}_v} \to x^*$
9: **else**
10: $\quad \mathcal{A}(\mathbb{G}, g, q, g^u, g^v) \to x^*$
11: **if** $(g^{uv}, x^*) \in \mathbf{X}$ **then**
12: $\quad$ **return** $\bot$
13: $y_0 \leftarrow \mathrm{PRF}(g^{uv}, x^*), y_1 \xleftarrow{\$} \{0,1\}^\lambda$
14: $\mathbf{X} \leftarrow \mathbf{X} \cup \{g^{uv}, x^*\}$
15: $b' \leftarrow \mathcal{A}(y_b)^{\mathsf{ODH}_u, \mathsf{ODH}_v}$
16: **return** $(b' = b)$

$\underline{\mathsf{ODH}_u(S, x, \mathbf{X}, c_u)}$:

1: **if** $(l = \mathsf{n}) \vee (\{S^u, x\} \in \mathbf{X}) \vee (S \notin \mathbb{G})$ **then**
2: $\quad$ **return** $\bot$
3: **if** $(l = \mathsf{s}) \wedge (c_u > 0)$ **then**
4: $\quad$ **return** $\bot$
5: $c_u \leftarrow c_u + 1, \mathbf{X} \leftarrow \mathbf{X} \cup \{S^u, x\}$
6: **return** $\mathrm{PRF}(S^u, x)$

$\underline{\mathsf{ODH}_v(S, x, \mathbf{X}, c_v)}$:

1: **if** $(r = \mathsf{n}) \vee (\{S^v, x\} \in \mathbf{X}) \vee (S \notin \mathbb{G})$ **then**
2: $\quad$ **return** $\bot$
3: **if** $(r = \mathsf{s}) \wedge (c_v > 0)$ **then**
4: $\quad$ **return** $\bot$
5: $c_v \leftarrow c_v + 1, \mathbf{X} \leftarrow \mathbf{X} \cup \{S^v, x\}$
6: **return** $\mathrm{PRF}(S^v, x)$

Figure B.14: The security experiments for the generic PRF-ODH assumption, and the symmetric PRF-ODH assumption. Note that both experiments make use of identical ODH_u and ODH_v oracles.

We say that the adversary $\mathcal{A}$ wins the lr-PRF-ODH game if $b' = b$ and define the following advantage function:

$$\mathsf{Adv}^{lr\text{-}\mathsf{PRF\text{-}ODH}}_{\mathrm{PRF},\mathbb{G}}(\mathcal{A}) := |\Pr(\mathsf{Exp}^{lr\text{-}\mathsf{sPRF\text{-}ODH}}_{\mathrm{PRF},\mathbb{G},g,q}(\mathcal{A}) = 1) - \frac{1}{2}|$$

However, in [34] it is shown that the existence of an algebraic efficient black-box reduction from the sn/ns-PRF-ODH problem to a decisional-Diffie–Hellman-augmented (DDHa) problem would imply that either the DDHa problem or the decisional-square Diffie–Hellman problem is not hard. The DDHa assumption is a class of assumptions,

roughly stating that the adversary cannot efficiently win the following hybrid game: between either the DDH problem or some other independent cryptographic problem (such as finding a collision in a hash), attempt to win either game. The decisional-square DH problem states that it is hard to distinguish (g, g^x, g^{x^2}) from (g, g^x, g^y). This in turn, shows that the existence of a standard-model instantiation of any generic PRF-ODH problem (excepting nn-PRF-ODH) is impossible, assuming the aforementioned problems are indeed hard. So a standard model proof of the Signal Protocol using generic PRF-ODH based assumptions could be moot.

To add to the difficulty, these left-and-right generic PRF-ODH assumption variants do not allow the adversary to query both sides of the DH keyshares multiple times before the challenger generates the secret value, which would be the case in the replacement of the long-term and medium-term secrets (refer to Case 1.1), which means that we would need to further modify the generic PRF-ODH assumption to the needs of our particular Signal Protocol proof. We call this a *symmetric* generic PRF-ODH problem, which we define below.

Definition B.2 (Symmetric PRF-ODH assumption)**.** *Let $\mathbb{G}$ be a cyclic group of order q with generator g. Let* $\mathrm{PRF} : \mathbb{G} \times \{0,1\}^* \to \{0,1\}^\lambda$ *be a pseudo-random function that takes as key input an element $k \in \mathbb{G}$ and an arbitrary-length salt value $x \in \{0,1\}^*$ as input, and outputs a value $y \in \{0,1\}^\lambda$.*

We define a symmetric security notion lr-sPRF-ODH, which is parameterised by $l, r \in \{\mathsf{n}, \mathsf{s}, \mathsf{m}\}$, indicating how often the adversary is able to query a certain left oracle or right oracle (denoted ODH_u and ODH_v respectively) where n *indicates that no query is allowed,* s *indicates that a single query is allowed, and* m *indicates that polynomially many queries are allowed to the respective oracle. Consider the security game $\mathsf{Exp}^{lr\text{-sPRF-ODH}}_{\mathrm{PRF},\mathbb{G},g,q}(\mathcal{A})$ described in Figure B.14. We say that the adversary $\mathcal{A}$ wins the lr-sPRF-ODH game if $b' = b$ and define the following advantage function:*

$$\mathsf{Adv}^{lr\text{-sPRF-ODH}}_{\mathrm{PRF},\mathbb{G},\mathcal{A}}(\lambda) := |\Pr(\mathsf{Exp}^{lr\text{-sPRF-ODH}}_{\mathrm{PRF},\mathbb{G},g,q}(\mathcal{A}) = 1) - \frac{1}{2}|$$

Assuming a sequence of groups in dependency of the security parameter, we say that a pseudo-random function PRF *with keys from* $(\mathbb{G})$ *provides lr-sPRF-ODH security*

(for $l, r \in \{\mathsf{n}, \mathsf{s}, \mathsf{m}\}$*) if for any efficient adversary* $\mathcal{A}$ *the advantage* $\mathsf{Adv}^{lr\text{-}\mathsf{sPRF\text{-}ODH}}_{\mathrm{PRF},\mathbb{G},\mathcal{A}}(\lambda)$ *is negligible in* λ*,* $log(q)$*.*

There is, however, an advantage to this effort: it would being clarity to which of the cases would be easier for the adversary to break. In the work by Brendel et. al. the relations between the variants of lr-PRF-ODH are shown, and thus the security of each of the cases is able to be compared concretely. In our current proof, the adversary seemingly does not have an advantage targeting one particular case over another. Now we have all the tools we would require to consider how the security proof in each case would work. We consider each case below and explain which flavour of PRF-ODH is required and why this would be the case.

Case 1.1. In Game 6 of Case 1.1 we know that the long-term identity key of party u and the medium-term key of the peer v have not been compromised by the adversary, and thus we can replace the key shares ipk_u, $prepk_v$ and the computed root key rk_1 and first-stage chain key $ck^{ir}_{0,0}$ with PRF-ODH challenge values. Since both of these Diffie–Hellman key shares can be used in multiple sessions, and (potentially) may be used before the Test session has initialised, we require many ODH_u and ODH_v queries at the start of the experiment before the challenge salt value x is computed. Thus, we require the mm-sPRF-ODH assumption, the strongest variant of the PRF-ODH problem. In this case, we treat the keys ipk_u and $prepk_v$ as the keys to the PRFODH problem (which is now internal to the mm-sPRF-ODH game) and the following (potentially revealed) secrets ipk_v and ek_u values as the salt value x that is queried to the mm-sPRF-ODH challenger. The challenge value y_b output by the challenger is then used to replace the rk_1, $ck^{ir}_{0,0}$ values in both the Tested session and the peer session that is used in computing the message key $mk^{ir}_{0,0}$ that was Tested by the adversary. To ensure that the message key $mk^{ir}_{0,0}$ is indistinguishable from random, we need an additional PRF game to replace the computation of $mk^{ir}_{0,0}$ from KDF_m, and use the output from the PRF challenger to replace $mk^{ir}_{0,0}$. Thus, an adversary capable of distinguishing these changes would also be capable of breaking the PRF security of KDF_m, or the $\mathsf{mm-sPRF\text{-}ODH}$ security of $\mathrm{HKDF}, \mathbb{G}$.

Case 1.2. In Game 6 of Case 1.2 we know that the predicate $\mathsf{clean}_{\mathtt{EL}}(u, i, [0])$ is upheld, which means that the ephemeral key of the initiator and the identity-key of the responder have not been compromised by the adversary, and thus we can replace the key shares epk_u, ipk_v, and the computed root key rk_1 and first-stage chain key $ck^{ir}_{0,0}$ with PRF-ODH challenge values. Since only the identity-key of the responder can be used in multiple sessions, and (potentially) may be used before the Test session has initialised, we require many ODH_u queries at the start of the experiment before the challenge salt value x is computed. Thus, we require the mn-PRF-ODH assumption, a non-symmetric variant of the PRF-ODH problem. In this case, we treat the values ipk_v, ek_u as the keys to the PRF-ODH problem, which is now internal to the mn-PRF-ODH game, and the following (potentially revealed) secrets $prepk_v$, ik_u as the salt value x that is queried to the mm-PRF-ODH challenger. The challenge value y_b output by the challenger is then used to replace the rk_1, $ck^{ir}_{0,0}$ values in both the Tested session and the peer session that is used in computing the message key $mk^{ir}_{0,0}$ that was Tested by the adversary. To ensure that the message key $mk^{ir}_{0,0}$ is indistinguishable from random, we need an additional PRF game to replace the computation of $mk^{ir}_{0,0}$ from KDF_m, and use the output from the PRF challenger to replace $mk^{ir}_{0,0}$. Thus, an adversary capable of distinguishing these changes would also be capable of breaking the PRF security of KDF_m, or the mn-PRF-ODH security of $\mathrm{HKDF}, \mathbb{G}$.

Case 1.3. In Game 6 of Case 1.3 we know that the predicate $\mathsf{clean}_{\mathtt{EM}}(u, i, [0])$ is upheld, which means that the ephemeral key of the initiator and the medium-term key of the responder have not been compromised by the adversary, and thus we can replace the keyshares epk_u, $prepk_v$ and the computed root key rk_1 and first-stage chain key $ck^{ir}_{0,0}$ with PRF-ODH challenge values. Since only the signed-prekey of the responder can be used in multiple sessions, and (potentially) may be used before the Test session has initialised, we require many ODH_v queries at the start of the experiment before the challenge salt value x is computed. However, an adversary may potentially reuse the ephemeral-key in later sessions with compromised long-term keys. Thus, we require the mn-PRF-ODH assumption,

a non-symmetric variant of the PRF-ODH problem. In this case, we treat the values $prepk_v$, ek_u as the key to the PRF-ODH problem, which is now internal to the mn-PRF-ODH game, and the following (potentially revealed) secrets ipk_v, ik_u values as the salt value x that is queried to the mn-PRF-ODH challenger. The challenge value y_b output by the challenger is then used to replace the rk_1, $ck^{ir}_{0,0}$ values in both the Tested session and the peer session that is used in computing the message key $mk^{ir}_{0,0}$ that was Tested by the adversary. To ensure that the message key $mk^{ir}_{0,0}$ is indistinguishable from random, we need an additional PRF game to replace the computation of $mk^{ir}_{0,0}$ from KDF_m, and use the output from the PRF challenger to replace $mk^{ir}_{0,0}$. Thus, an adversary capable of distinguishing these changes would also be capable of breaking the PRF security of KDF_m, or the mn-PRF-ODH security of $\text{HKDF}, \mathbb{G}$.

Case 2.1. This is treated identically to Case 1.1, with the same bounds and game hops.

Case 2.2 This is treated identically to Case 1.2, with the same bounds and game hops.

Case 2.3 This is treated identically to Case 1.3, with the same bounds and game hops.

Case 2.4. In Game 6 of Case 2.4 we know that the predicate $\mathsf{clean}_{\texttt{EE}}(u, i, [0])$ is upheld, which means that the ephemeral key of the initiator and the ephemeral key of the responder have not been compromised by the adversary, and thus we can replace the key shares epk_u, epk_v and the computed root key rk_1 and first-stage chain key $ck^{ir}_{0,0}$ with PRF-ODH challenge values. Since both keys are ephemerally generated and only used a single time, we require the sn-PRF-ODH assumption, the weakest non-standard model variant of the PRF-ODH problem. In this case, we treat the epk_v, ek_u as the key to the PRF-ODH problem, which is now internal to the sn-PRF-ODH game, and the following (potentially revealed) secrets $prepk_v$, ik_v and ik_u values as the salt value x that is queried to the sn-PRF-ODH challenger.

The challenge value y_b output by the challenger is then used to replace the rk_1, $ck^{ir}_{0,0}$ values in both the Tested session and the peer session that is used in computing the message key $mk^{ir}_{0,0}$ that was Tested by the adversary. To ensure that the message key $mk^{ir}_{0,0}$ is indistinguishable from random, we need an additional PRF game to replace the computation of $mk^{ir}_{0,0}$ from KDF_m, and use the output from the PRF challenger to replace $mk^{ir}_{0,0}$. Thus, an adversary capable of distinguishing these changes would also be capable of breaking the PRF security of KDF_m, or the sn-PRF-ODH security of $\text{HKDF}, \mathbb{G}$.

Case 3.1. In Game 6 of Case 3.1 we know that the predicate $\mathsf{clean}^{asym\text{-}ri}_{\texttt{EE}}$ is upheld, which means that the ratchet key of the initiator and the ratchet key of the responder have not been compromised by the adversary, and thus we can replace the key shares $rchpk^0_u$, $rchpk^0_v$ and the computed temporary value x and asymmetric-stage chain key $ck^{ri}_{0,0}$ with PRF-ODH challenge values. Since both keys are ephemerally generated and only used a single time, we require the sn-PRF-ODH assumption, the weakest non-standard model variant of the PRF-ODH problem. In this case, we treat the values $rchpk^0_v$, $rchk^0_u$ as the key to the PRF-ODH problem, which is now internal to the sn-PRF-ODH game, and the (potentially revealed) root key rk_1 of the first stage as the salt value x that is queried to the sn-PRF-ODH challenger. The challenge value y_b output by the challenger is then used to replace the x, $ck^{ri}_{0,0}$ values in both the Tested session and the peer session that is used in computing the message key $mk^{ri}_{0,0}$ that was Tested by the adversary. To ensure that the message key $mk^{ri}_{0,0}$ is indistinguishable from random, we need an additional PRF game to replace the computation of $mk^{ri}_{0,0}$ from KDF_m, and use the output from the PRF challenger to replace $mk^{ri}_{0,0}$. Thus, an adversary capable of distinguishing these changes would also be capable of breaking the PRF security of KDF_m, or the sn-PRF-ODH security of $\text{HKDF}, \mathbb{G}$.

Case 3.2. In Game 6 of Case 3.2 we know that the predicate $\mathsf{clean}_{\pi^i_u.type[0]}(u, i, [0])$ is upheld, which means that initial key exchange stage has some Diffie–Hellman key share pair that has not been corrupted, and that the adversary has not

revealed the state linking the initial key exchange to this stage. Depending on which clean predicate that was upheld in the first stage, the replacement of Diffie–Hellman values is done as in Case 1.1, Case 1.2, Case 1.3 or Case 2.4. We know from these case analysis that the root key rk_1 is indistinguishable from random, and thus we are able to replace this value with a random value $rk_{1'}$ and note that an adversary capable of distinguishing this change can break the security of Case 1.1, Case 1.2, Case 1.3 or Case 2.4. We then use PRF game hops in a standard way to replace the derivation of the x, $ck^{ri}_{0,0}$ values in both the Tested session and the peer session that is used in computing the message key $mk^{ri}_{0,0}$ which was Tested by the adversary. To ensure that the message key $mk^{ri}_{0,0}$ is indistinguishable from random, we need an additional PRF game to replace the computation of $mk^{ri}_{0,0}$ from KDF_m, and use the output from the PRF challenger to replace $mk^{ri}_{0,0}$. Thus, an adversary capable of distinguishing these changes would also be capable of breaking the PRF security of KDF_m, or the mm-sPRF-ODH security of $\mathrm{HKDF}, \mathbb{G}$ (as in Case 1.1), or the mn-PRF-ODH security of $\mathrm{HKDF}, \mathbb{G}$ (as in Cases 1.2 and 1.3), or the sn-PRF-ODH security of $\mathrm{HKDF}, \mathbb{G}$ (as in Case 2.4).

Case 4.1.1. In Game 6 of Case 4.1.1 we know that $\mathsf{clean}_{\texttt{asym-ri}}(u, i, [\textbf{asym-ri}{:}x - 1])$ is upheld. This means that either:

- the previous stage's ratchet keys have not been compromised by the adversary (in which case analysis follows from Case 3.1)
- the previous stage's state has not been compromised by the adversary (in which case analysis follows from Case 3.2)

In a similar way, then, we follow those cases to replace the appropriate uncompromised Diffie–Hellman key shares with challenge values from a PRFODH game. Thus, an adversary capable of distinguishing these changes would also be capable of breaking the PRF security of KDF_m, or the mm-sPRF-ODH security of $\mathrm{HKDF}, \mathbb{G}$ (as in Case 1.1), or the mn-PRF-ODH security of $\mathrm{HKDF}, \mathbb{G}$ (as in

Cases 1.2 and 1.3), or finally the sn-PRF-ODH security of HKDF, $\mathbb{G}$ (as in Cases 2.4 and 3.1).

Case 4.1.2. In Game 6 of Case 4.1.2 we know that the predicate $\mathsf{clean}_{\mathtt{EE}}(u, i, x-1, x-1)$ is upheld, which means that the ratchet keys from the previous stage have not been compromised by the adversary and analysis follows from Case 3.1, with the same bounds and game hops. In particular, this means that the advantage of the adversary in breaking the key indistinguishability of the Tested session key is bound by the PRF security of KDF_r and KDF_m, or the sn-PRF-ODH security of HKDF, $\mathbb{G}$.

Case 4.2.1. In Game 6 of Case 4.1.1 we know that $\mathsf{clean}_{\mathtt{asym\text{-}ri}}(u, i, [\textbf{asym-ir}{:}x-1])$ is upheld. Note that similarly to the proof of Case 4.2.1, this follows identically to Case 4.1.1 with an additionally application of a PRF game to account for the intermediate computation of a *tmp* value.

Case 4.2.2. In Game 6 of Case 4.2.2 we know that the predicate $\mathsf{clean}_{\mathtt{EE}}(u, i, x-1, x-1)$ is upheld, which means that the previous stages ratchet keys have not been compromised by the adversary and analysis follows identically to Case 4.1.2, with an additional PRF game to account for the intermediate computation of a *tmp* value. In particular, this means that the advantage of the adversary in breaking the key indistinguishability of the Tested session key is bound by the PRF security of KDF_r and KDF_m, or the sn-PRF-ODH security of HKDF, $\mathbb{G}$.

Case 5. In this case, and all subcases, the analysis follows from Case 4, with additional PRF game hops to inductively replace chain keys that (via the cleanness predicate $\mathsf{clean}_{\mathtt{sym}}$) have not been compromised by the adversary.

From this vantage point, we can now compare the cases concretely. For instance, it is clear that the advantage of the adversary in breaking Case 1.1 (where the long-term identity key of the initiator and the medium-term signed prekey of the responder have not been trivially compromised by the adversary) is quantitatively higher than the advantage of the adversary in breaking Case 2.4 (where the ephemeral

key of the initiator and the one-time prekey of the responder have not been trivially compromised by the adversary).

Bibliography

[1] Martin Abadi and Phillip Rogaway. Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption). In *IFIP International Conference on Theoretical Computer Science (TCS)*, volume 15, pages 103–127. Springer, 2002.

[2] Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-Based Authenticated Key Exchange in the Three-Party Setting. In *International Workshop on Public Key Cryptography*, pages 65–84. Springer, 2005.

[3] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, 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 SIGSAC Conference on Computer and Communications Security (CCS)*, pages 5–17, 2015.

[4] Nadhem AlFardan and Kenneth Paterson. Lucky Thirteen: Breaking the TLS and DTLS Record Protocols. In IEEE Symposium on Security and Privacy (S&P), pages 526–540, 2013.

[5] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, Alex Halderman, Viktor Dukhovni, Emilia Käsper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt. DROWN: Breaking TLS Using SSLv2. In *USENIX Security Symposium*, pages 689–706, 2016.

[6] Christoph Bader, Dennis Hofheinz, Tibor Jager, Eike Kiltz, and Yong Li. Tightly-Secure Authenticated Key Exchange. In *Theory of Cryptography Conference*, pages 629–658, 2015.

[7] Christoph Bader, Tibor Jager, Yong Li, and Sven Schäge. On the Impossibility of Tight Cryptographic Reductions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 273–304. Springer, 2016.

[8] 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*, pages 146–166. Springer, 2014.

[9] David Basin and Cas Cremers. Modeling and Analyzing Security in the Presence of Compromising Adversaries. In *IEEE European Symposium on Research in Computer Security (ESORICS)*, pages 340–356. Springer, 2010.

[10] David Basin and Cas Cremers. Know your Enemy: Compromising Adversaries in Protocol Analysis. In *ACM Transactions on Information and System Security*, volume 17, page 7. ACM, 2014.

[11] David Basin, Cas Cremers, and Marko Horvat. Actor Key Compromise: Consequences and Countermeasures. In *IEEE Computer Security Foundations Symposium (CSF)*, pages 244–258, 2014.

[12] Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. An Uninstantiable Random-Oracle-Model Scheme for a Hybrid-Encryption Problem. In *Advances in Cryptology, EUROCRYPT*, pages 171–188. Springer, 2004.

[13] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols. In *ACM Symposium on Theory of Computing*, pages 419–428, 1998.

[14] Mihir Bellare, Tadayoshi Kohno, and Victor Shoup. Stateful Public-Key Cryptosystems: How to Encrypt With one 160-bit Exponentiation. In *ACM Conference on Computer and Communications Security (CCS)*, pages 380–389, 2006.

[15] Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM Conference on Computer and Communications Security (CCS)*, pages 62–73, 1993.

[16] Mihir Bellare and Phillip Rogaway. Entity Authentication and Key Distribution. In *Advances in Cryptology, CRYPTO*, pages 232–249. Springer, 1994.

[17] Daniel J Bernstein. Curve25519: New Diffie–Hellman Speed Records. In *International Workshop on Public Key Cryptography*, pages 207–228. Springer, 2006.

[18] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-Speed High-Security Signatures. *Journal of Cryptographic Engineering*, pages 1–13, 2012.

[19] Daniel J Bernstein, Tanja Lange, and Ruben Niederhagen. Dual EC: A Standardized Back Door. IACR Cryptology ePrint Archive: Report 2015/767, available online: `https://eprint.iacr.org/2015/767`, 2015.

[20] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. A Messy State of the Union: Taming the Composite State Machines of TLS. In IEEE Symposium on Security and Privacy (S&P), pages 535–552, 2015.

[21] Karthikeyan Bhargavan, Antoine Delignat Lavaud, Cédric Fournet, Alfredo Pironti, and Pierre Yves Strub. Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS. In IEEE Symposium on Security and Privacy (S&P), pages 98–113, 2014.

[22] Karthikeyan Bhargavan and Gaëtan Leurent. Transcript Collision Attacks: Breaking Authentication in TLS, IKE, and SSH. In *Network and Distributed System Security Symposium (NDSS)*, 2016.

[23] Simon Blake-Wilson and Alfred Menezes. Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol. In *Public Key Cryptography*, pages 154–170. Springer, 1999.

[24] Bruno Blanchet. A Computationally Sound Mechanized Prover for Security Protocols. In *IEEE Transactions on Dependable and Secure Computing*, volume 5, pages 193–207, 2008.

[25] Bruno Blanchet. Automatic Verification of Correspondences for Security Protocols. In *Journal of Computer Security*, volume 17, pages 363–434, 2009.

[26] Daniel Bleichenbacher. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. In *Advances in Cryptology, CRYPTO*, pages 1–12. Springer, 1998.

[27] Tobias Boelter. WhatsApp Retransmission Vulnerability. `https://tobi.rocks/2016/04/whats-app-retransmission-vulnerability/`, 2017.

[28] Dan Boneh and Matt Franklin. Identity-Based Encryption from the Weil Pairing. In *Advances in Cryptology, CRYPTO*, pages 213–229. Springer, 2001.

[29] Dan Boneh and Jonathan Katz. Improved Efficiency for CCA-Secure Cryptosystems Built Using Identity-Based Encryption. In *Cryptographers' Track at the RSA Conference*, pages 87–103. Springer, 2005.

[30] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-Record Communication, or, why Not to Use PGP. In *ACM Workshop on Privacy in the Electronic Society*, pages 77–84, 2004.

[31] Kevin D Bowers, Ari Juels, Ronald L Rivest, and Emily Shen. Drifting Keys: Impersonation Detection for Constrained Devices. In *IEEE INFOCOM*, pages 1025–1033, 2013.

[32] Colin Boyd, Cas Cremers, Michèle Feltz, Kenneth Paterson, Bertram Poettering, and Douglas Stebila. ASICS: Authenticated Key Exchange Security Incorporating Certification Systems. In *IEEE European Symposium on Research in Computer Security (ESORICS)*, pages 381–399. Springer, 2013.

[33] Colin Boyd and Anish Mathuria. *Protocols for Authentication and Key Establishment*. Springer Science & Business Media, 2013.

[34] Jacqueline Brendel, Marc Fischlin, Felix Günther, and Christian Janson. PRF-ODH: Relations, Instantiations, and Impossibility Results. In *Advances in Cryptology, CRYPTO*, pages 651–681. Springer, 2017.

[35] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Dynamic Group Diffie–Hellman Key Exchange Under Standard Assumptions. In *Advances in Cryptology, EUROCRYPT 2002*, pages 321–336. Springer, 2002.

[36] Christina Brzuska, Marc Fischlin, Nigel Smart, Bogdan Warinschi, and Stephen C Williams. Less is More: Relaxed yet Composable Security Notions for Key Exchange. In *International Journal of Information Security*, volume 12, pages 267–297. Springer, 2013.

[37] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen Williams. Composability of Bellare-Rogaway Key Exchange Protocols. In *ACM Conference on Computer and Communications Security (CCS)*, pages 51–62, 2011.

[38] Jan Camenisch and Victor Shoup. Practical Verifiable Encryption and Decryption of Discrete Logarithms. In *Advances in Cryptology, CRYPTO*, pages 126–144. Springer, 2003.

[39] Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 136–145, 2001.

[40] Ran Canetti, Oded Goldreich, and Shai Halevi. The Random Oracle Methodology, Revisited. In *Journal of the ACM (JACM)*, pages 557–594, 2004.

[41] Ran Canetti and Hugo Krawczyk. Analysis of Key-Exchange Protocols and Their use for Building Secure Channels. In *Advances in Cryptology, EUROCRYPT 2001*, pages 453–474. Springer, 2001.

[42] Paul Chadwick. Flawed Reporting About WhatsApp. The Guardian, available online: `https://www.theguardian.com/technology/commentisfree/2017/jun/28/flawed-reporting-about-whatsapp`, 2017.

[43] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A Formal Security Analysis of the Signal Messaging Protocol. In IEEE European Symposium on Security and Privacy (EuroS&P), pages 451–466, 2017.

[44] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. On Post-Compromise Security. In *IEEE Computer Security Foundations Symposium (CSF), pages 164–178*, 2016.

[45] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On Ends-to-Ends Encryption: Asynchronous Group Messaging with Strong Security Guarantees. IACR Cryptology ePrint Archive Report: 2017/666, available online: `https://eprint.iacr.org/2017/666`, 2017.

[46] Common Vulneriblities and Exposures (CVE) website. `https://www.cve.mitre.org/`.

[47] Confide. `https://getconfide.com/`, 2017.

[48] Ronald Cramer and Victor Shoup. Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public-Key Encryption. In *Advances in Cryptology, EUROCRYPT*, pages 45–64. Springer, 2002.

[49] Cas Cremers and Michèle Feltz. One-Round Strongly Secure Key Exchange with perfect Forward Secrecy and Deniability. IACR Cryptology ePrint Archive Report: 2011/300, available online: `https://eprint.iacr.org/2011/300`, 2011.

[50] Cas Cremers and Michèle Feltz. Beyond eCK: perfect forward Secrecy Under Actor Compromise and Ephemeral-Key Reveal. In *IEEE European Symposium on Research in Computer Security (ESORICS)*, pages 734–751. Springer, 2012.

[51] Cas Cremers and Michèle Feltz. On the Limits of Authenticated Key Exchange Security with an Application to Bad Randomness. Cryptology ePrint Archive Report: 2014/369, available online: `https://eprint.iacr.org/2014/369`, 2014.

[52] Cas Cremers, Marko Horvat, Sam Scott, and Thyla van der Merwe. Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication. In IEEE Symposium on Security and Privacy (S&P), pages 470–485, 2016.

[53] Daniel Kahn Gillmor. [TLS] OPTLS: Signature-less TLS 1.3. `https://www.ietf.org/mail-archive/web/tls/current/msg14423.html`, 2014.

[54] Jean Paul Degabriele, Anja Lehmann, Kenneth Paterson, Nigel Smart, and Mario Strefler. On the Joint Security of Encryption and Signature in EMV. In *Cryptographers' Track at the RSA Conference*, pages 116–135. Springer, 2012.

[55] Alexander W Dent. A Note On Game-Hopping Proofs. IACR Cryptology ePrint Archive Report: 2006/260, available online: `https://eprint.iacr.org/2006/260`, 2006.

[56] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. Deniable Authentication and Key Exchange. In *ACM Conference on Computer and Communications Security (CCS)*, pages 400–409, 2006.

[57] Whitfield Diffie and Martin E Hellman. New Directions in Cryptography. In *IEEE Transactions on Information Theory*, volume 22, pages 644–654, 1976.

[58] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A Cryptographic Analysis of the TLS 1.3 Handshake Protocol Candidates. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1197–1210, 2015.

[59] Electronic Frontier Foundation Secure Messaging Scorecard. `https://www.eff.org/secure-messaging-scorecard`, 2014.

[60] Taher ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology, CRYPTO*, pages 10–18. Springer, 1985.

[61] Marc Fischlin and Felix Günther. Multi-Stage Key Exchange and the Case of Google's QUIC Protocol. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1193–1204, 2014.

[62] 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), pages 457–472, 2016.

[63] Eiichiro Fujisaki, Tatsuaki Okamoto, David Pointcheval, and Jacques Stern. RSA-OAEP is Secure Under the RSA Assumption. In *Advances in Cryptology, CRYPTO*, pages 260–274. Springer, 2001.

[64] Manisha Ganguly. WhatsApp Design Feature Means Some Encrypted Messages Could be Read by Third Party. The Guardian, available online: `https://www.theguardian.com/technology/2017/jan/13/whatsapp-design-feature-encrypted-messages`, 2017.

[65] 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 *USENIX Security Symposium*, pages 655–672, 2016.

[66] Martin Geisler and Nigel Smart. Distributing the Key Distribution Centre in Sakai–Kasahara Based Systems. In *IMA International Conference on Cryptography and Coding*, pages 252–262. Springer, 2009.

[67] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to Construct Random Functions. In *IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 464–479, 1984.

[68] Shafi Goldwasser and Yael Tauman Kalai. Cryptographic Assumptions: A Position Paper. IACR Cryptology ePrint Archive Report: 2015/907, available online: `https://eprint.iacr.org/2015/907`, 2015.

[69] Shafi Goldwasser and Silvio Micali. Probabilistic Encryption. In *Journal of Computer and System Sciences*, volume 28, pages 270–299. Elsevier, 1984.

[70] Matthew D Green and Ian Miers. Forward Secure Asynchronous Messaging from Puncturable Encryption. In IEEE Symposium on Security and Privacy (S&P), pages 305–320, 2015.

[71] Michael Groves. MIKEY-SAKKE: Sakai-Kasahara Key Encryption in Multimedia Internet KEYing (MIKEY). In *IETF RFC 6509*, 2012.

[72] Michael Groves. Sakai-Kasahara Key Encryption (SAKKE). In *IETF RFC 6508*, 2012.

[73] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message Franking via Committing Authenticated Encryption. In *Advances in Cryptology, CRYPTO*, pages 66–97. Springer, 2017.

[74] Felix Günther, Britta Hale, Tibor Jager, and Sebastian Lauer. 0-RTT Key Exchange with Full Forward Secrecy. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 519–548. Springer, 2017.

[75] Peter Gutmann. Key Management Through Key Continuity (KCM). IETF Internet Draft, available online: `https://tools.ietf.org/id/draft-gutmann-keycont-01.txt`, 2008.

[76] Mike Hamburg. Ed448-Goldilocks, a New Elliptic Curve. IACR Cryptology ePrint Archive Report: 2015/625, available online: `https://eprint.iacr.org/2015/625`, 2015.

[77] Dennis Hofheinz and Victor Shoup. GNUC: A New Universal Composability Framework. *In Journal of Cryptology*, pages 1–86, 2013.

[78] 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*, pages 273–293. Springer, 2012.

[79] Tibor Jager, Jörg Schwenk, and Juraj Somorovsky. On the Security of TLS 1.3 and QUIC Against Weaknesses in PKCS#1 v1.5 Encryption. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1185–1196, 2015.

[80] Mike Just and Serge Vaudenay. Authenticated Multi-Party Key Agreement. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 36–49. Springer, 1996.

[81] Joe Kilian and Phillip Rogaway. How to Protect DES Against Exhaustive Key Search. In *Journal of Cryptology, CRYPTO*, volume 1109, pages 252–267. Springer, 1996.

[82] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. Automated Verification for Secure Messaging Protocols and their Implementations: A Symbolic and Computational Approach. In IEEE European Symposium on Security and Privacy (EuroS&P), 2017.

[83] Neal Koblitz and Alfred Menezes. Another Look at "Provable Security". II. Cryptology ePrint Archive: Report 2006/229, available online: `https://eprint.iacr.org/2006/229`, 2006.

[84] Neal Koblitz and Alfred Menezes. Another Look at "Provable Security". In *Journal of Cryptology*, volume 20, pages 3–37. Springer, 2007.

[85] Neal Koblitz and Alfred Menezes. Another Look at Security Definitions. IACR Cryptology ePrint Archive Report: 2011/343, available online: `https://eprint.iacr.org/2011/343`, 2011.

[86] Neal Koblitz and Alfred Menezes. The Random Oracle Model: A Twenty-Year Retrospective. In *Designs, Codes and Cryptography*, volume 77, pages 587–610. Springer, 2015.

[87] Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the Security of TLS-DH and TLS-RSA in the Standard Model. IACR Cryptology ePrint Archive Report: 2013/367, available online: `https://eprint.iacr.org/2013/367`, 2013.

[88] Hugo Krawczyk. HMQV: A High-Performance Secure Diffie–Hellman Protocol. In *Advances in Cryptology, CRYPTO*, pages 546–566. Springer, 2005.

[89] Hugo Krawczyk. Cryptographic Extraction and Key Derivation: The HKDF Scheme. In *Advances in Cryptology, CRYPTO*, pages 631–648. Springer, 2010.

[90] Hugo Krawczyk. [TLS] OPTLS: Signature-less TLS 1.3. `https://www.ietf.org/mail-archive/web/tls/current/msg14592.html`, 2014.

[91] Hugo Krawczyk, Kenneth Paterson, and Hoeteck Wee. On the Security of the TLS Protocol: A Systematic Analysis. In *Advances in Cryptology, CRYPTO*, pages 429–448. Springer, 2013.

[92] Hugo Krawczyk and Hoeteck Wee. The OPTLS Protocol and TLS 1.3. In IEEE European Symposium on Security and Privacy (EuroS&P), pages 81–96.

[93] Caroline Kudla and Kenneth Paterson. Modular Security Proofs for Key Agreement Protocols. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 549–565. Springer, 2005.

[94] Kaoru Kurosawa and Yvo Desmedt. A New Paradigm of Hybrid Encryption Scheme. In *Advances in Cryptology, CRYPTO*, pages 345–359. Springer, 2004.

[95] Brian LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger Security of Authenticated Key Exchange. In *International Conference on Provable Security (ProvSec)*, pages 1–16. Springer, 2007.

[96] Adam Langley. Pond. `https://github.com/agl/pond`, 2014.

[97] Steven Levy. The Open Secret. Wired, available online: `https://www.wired.com/1999/04/crypto/`, 1999.

[98] Yong Li and Sven Schäge. No-Match Attacks and Robust Partnering Definitions—Defining Trivial Attacks for Security Protocols is Not Trivial. In *ACM Conference on Computer and Communications Security (CCS)*, pages 1343–1360, 2017.

[99] Line. https://line.me/en/, 2011.

[100] Gavin Lowe. An attack on the Needham–Schroeder Public-Key Authentication Protocol. In *Information Processing Letters*, volume 56, pages 131–133. Elsevier, 1995.

[101] Moxie Marlinspike. There is no WhatsApp "backdoor". https://signal.org/blog/there-is-no-whatsapp-backdoor/, 2017.

[102] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In *International Conference on Computer Aided Verification*, pages 696–701. Springer, 2013.

[103] Alfred Menezes and Berkant Ustaoglu. On Reusing Ephemeral Keys in Diffie–Hellman Key Agreement Protocols. In *International Journal of Applied Cryptography*, volume 2, pages 154–158, 2010.

[104] Messenger Secret Conversations Technical White Paper. https://fbnewsroomus.files.wordpress.com/2016/07/secret_conversations_whitepaper-1.pdf, 2016.

[105] Kevin Milner, Cas Cremers, Jiangshan Yu, and Mark Ryan. Automatically Detecting the Misuse of Secrets: Foundations, Design Principles, and Applications. In *IEEE Computer Security Foundations Symposium (CSF), pages 203–216*, 2017.

[106] Vinnie Moscaritolo, Gary Belvin, and Phillip Zimmermann. Silent Circle Instant Messaging Protocol (Github repository). https://github.com/SilentCircle/silent-text/tree/master/Documentation, 2012.

[107] National Vulneriblity Database (NVD) website. https://www.nvd.nist.gov/.

[108] Matus Nemec, Marek Sys, Petr Svenda, Dusan Klinec, and Vashek Matyas. The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli. In *ACM Conference on Computer and Communications Security (CCS)*, 2017.

[109] Ashley Parker and Philip Rucker Rucker. Upheaval is Now Standard Operating Procedure Inside the White House. The Washington Post, available online: https://www.washingtonpost.com/politics/upheaval-is-now-standard-operating-procedure-inside-the-white-house/2017/02/13/d65dee58-f213-11e6-a9b0-ecee7ce475fc_story.html?hpid=hp_rhp-banner-main_whitehouse-0719pm%3Ahomepage%2Fstory&utm_term=.12761eca3fbc, 2017.

[110] Kenneth Paterson, Jacob Schuldt, Martijn Stam, and Susan Thomson. On the Joint Security of Encryption and Signature, Revisited. *Advances in Cryptology, ASIACRYPT*, pages 161–178, 2011.

[111] Trevor Perrin. The XEdDSA and VXEdDSA Signature Schemes. Open Whisper Systems, available online: https://whispersystems.org/docs/specifications/xeddsa/, 2016.

[112] Trevor Perrin and Moxie Marlinspike. Double Ratchet Algorithm (GitHub) wiki). Open Whisper Systems, available online: https://github.com/trevp/double_ratchet/wiki, 2016.

[113] Trevor Perrin and Moxie Marlinspike. `libsignal-protocol-java`. Open Whisper Systems, GitHub repository commit hash `4a7bc1667a68c1d8e6af0151be30b84b94fd1e38`, available online: `https://github.com/WhisperSystems/libsignal-protocol-java`, 2016.

[114] Trevor Perrin and Moxie Marlinspike. The Double Ratchet Algorithm (specification). Open Whisper Systems, available online: `https://whispersystems.org/docs/specifications/doubleratchet/`, 2016.

[115] Trevor Perrin and Moxie Marlinspike. The X3DH Key Agreement Protocol. Open Whisper Systems, available online: `https://whispersystems.org/docs/specifications/x3dh/`, 2016.

[116] Joel Reardon, David Basin, and Srdjan Capkun. SoK: Secure Data Deletion. In IEEE Symposium on Security and Privacy (S&P), pages 301–315, 2013.

[117] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. IETF Internet-Draft, draft-ietf-tls-tls13-14, available online: `http://www.ietf.org/internet-drafts/draft-ietf-tls-tls13-14.txt`.

[118] Phillip Rogaway. Authenticated-Encryption with Associated-Data. In *ACM Conference on Computer and Communications Security (CCS)*, pages 98–107, 2002.

[119] Ryuichi Sakai and Masao Kasahara. ID Based Cryptosystems with Pairing on Elliptic Curve. IACR Cryptology ePrint Archive: Report 2003/054, available online: `https://eprint.iacr.org/2003/054`, 2003.

[120] Michael Scott. Distributing SAKKE Private Key Generation. `https://www.miracl.com/hs-fs/hub/230906/file-20129876-pdf/downloads/certivox_labs_d-pkg_for_sakke.pdf`, 2012.

[121] Victor Shoup. OAEP Reconsidered. In *Advances in Cryptology, CRYPTO*, pages 239–259. Springer, 2001.

[122] Victor Shoup. Sequences of Games: A Tool for Taming Complexity in Security Proofs. IACR Cryptology ePrint Archive: Report 2004/332, available online: `https://eprint.iacr.org/2004/332`, 2004.

[123] Telegram. `https://telegram.org/`, 2013.

[124] Telegram Blog: 100,000,000 Monthly Active Users. `https://telegram.org/blog/100-million`, 2016.

[125] Threema. `https://threema.ch/en`, 2012.

[126] Sean Turner. The Application/PKCS #10 Media Type. In *IETF RFC 5967*, 2010.

[127] Nik Unger and Ian Goldberg. Deniable Key Exchanges for Secure Messaging. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1211–1223, 2015.

[128] Viber. `https://www.viber.com/`, 2010.

[129] WhatsApp Encryption Overview Technical White Paper. `https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf`, 2016.

[130] Wickr. `https://www.wickr.com/`, 2012.

[131] Malcolm Williamson. Non-Secret Encryption Using a Finite Field. CESG, available online: `https://www.gchq.gov.uk/sites/default/files/document_files/nonsecret_encryption_finite_field_0.pdf`, 1974.

[132] Malcolm Williamson. Thoughts on Cheaper Non–Secret Encryption. CESG, available online: `http://cryptocellar.org/cesg/cheapnse.pdf`, 1976.

[133] Wire. `https://wire.com/en/`, 2014.

[134] Scott Yilek, Eric Rescorla, Hovav Shacham, Brandon Enright, and Stefan Savage. When Private Keys are Public: Results from the 2008 Debian OpenSSL Vulnerability. In *ACM Internet Measurement Conference*, pages 15–27, 2009.

[135] Tatu Ylonen and Chris Lonvick. The Secure Shell (SSH) Connection Protocol. In *IETF RFC 4254*, 2006.

[136] Phil Zimmermann, Alan Johnston, and Jon Callas. ZRTP: Media Path Key Agreement for Unicast Secure RTP. In *IETF RFC 6189*, 2011.

[137] Philip Zimmermann. *The Official PGP User's Guide*. MIT Press, 1995.